Drug Disease Model Resources

ddmore

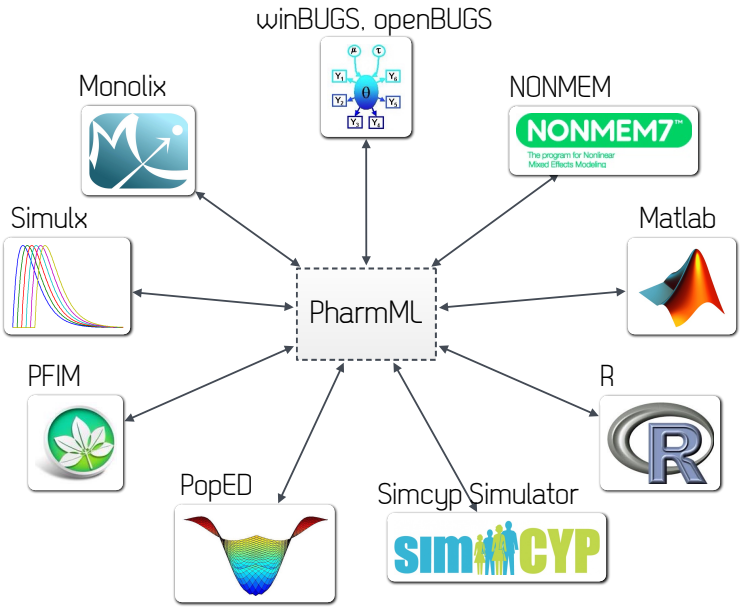
2^{nd} Public Release

Pharmacometrics Markup Language (PharmML)

Language Specification for Version 0.6

Maciej J Swat[1], Sarala Wimalaratne[1], Niels Rode Kristensen[2], Florent Yvon[1], Stuart Moodie[1,3], Nicolas Le Novère[1,4]

[1] *EMBL - European Bioinformatics Institute, Cambridge, UK*,
[2] *Novo Nordisk A/S, Bagsværd, Denmark*,
[3] *Eight Pillars Ltd, Edinburgh, UK*,
[4] *Babraham Institute, Cambridge, UK*



January 29, 2015
1^{st} public version, 0.2.1, was released on November 21, 2013

Contents

Chapter 1

What is PharmML?

1.1 The Problem

The principal problem that PharmML addresses is the reliable exchange of pharmacometric models between software tools. This is illustrated in Figure 1.1, where PharmML is the exchange medium for pharmacometric models for the main modelling and simulation (M&S) tools in the field. This goal has been successfully realised in the field of systems biology, neurobiology or data mining.

In systems biology for example software tools exchange models using the Systems Biology Markup Language (SBML; `http://www.sbml.org`) [Hucka et al., 2003] and many published models can be found in the BioModels Database (`http://www.ebi.ac.uk/biomodels-main/`) [Li et al., 2010]. Modellers don't worry about the content of an SBML file; they rely on the fact that when they exchange it between the modelling tools, it just works. Crucial to its success has been an active community of tool developers and modellers who have supported and used it during that time. Equally important has been the provision of sophisticated software libraries (libSBML and JSBML) that take away much of the pain a software tool developer would otherwise experience supporting what is now quite a complex standard. It is a virtuous circle. Users demand their modelling tools support SBML. Developers provide reliable SBML support using libSBML, which enables them to give their users what they want. The more tools that support SBML, the more useful it becomes. The cost of supporting SBML is not negligible but quality libraries like libSBML make the cost acceptable.

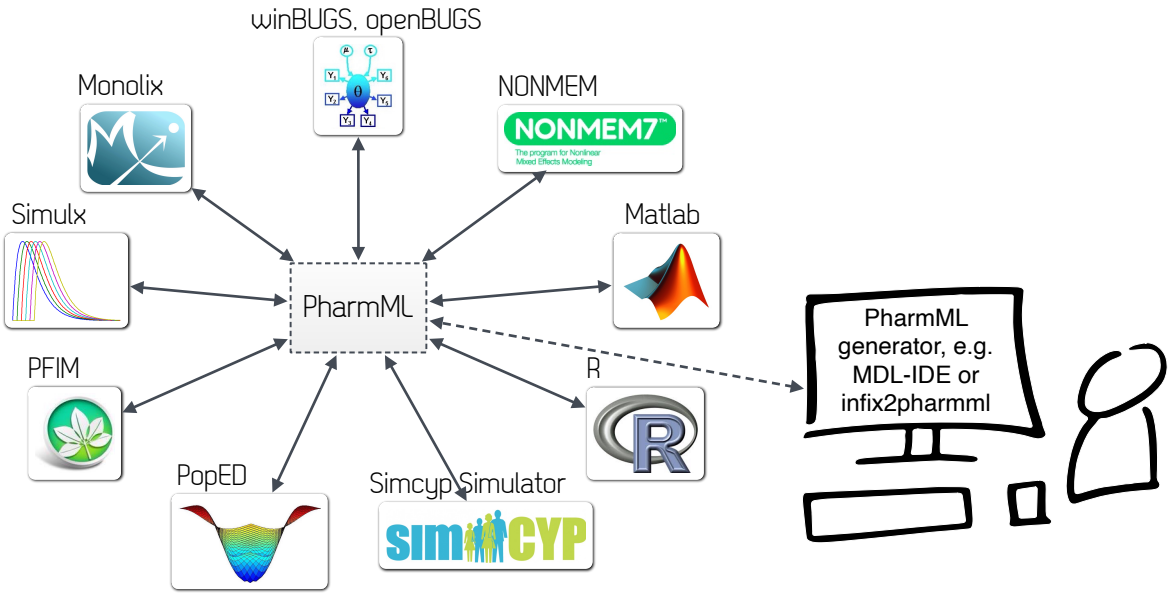


Figure 1.1: Interoperability platform to exchange models via PharmML.

This lesson has not been ignored by the pharmacometrics community and in fact a number of years ago

the NLME consortium (a consortium of pharmaceutical companies now all part of DDMoRe) started to work on a very similar standard to PharmML. This resulted in early drafts of an XML based exchange language, called PharML, but work on it was unfortunately discontinued and the standard has never been used and validated. There are a number of other exchange standards in related modelling fields, which we have drawn on in the development of our work to varying degrees, including:

CellML Supports the exchange and storage of computer based mathematical models of biological systems [Nielsen and Halstead, 2004].

NeuroML Supports the exchange and description of models "to describe the biophysics, anatomy and network architecture of neuronal systems at multiple scales"[1] [Gleeson et al., 2010].

NineML Describes neuronal networks in a "simulator independent language"[2] that is design to interact with NeuroML [Gorchetchnikov et al., 2010].

SED-ML Encodes simulation experiments of SBML and CellML models "to ensure exchangeability and reproducibility of simulation experiments"[3] [Waltemath et al., 2011].

PharmML is supposed to be the solution in pharmacometrics to the problem. An XML based language that will be able to encode models from NONMEM, MONOLIX, BUGS and related tools. We intend this to be a community standard nucleated around the members of the DDMoRe consortium. In addition we are developing a software library (libPharmML[4]) to help tool providers incorporate support of PharmML and to facilitate its general adoption in the field.

1.2 The Solution

Having described the problem, we here articulate the *kind* of solution we wanted PharmML to be. Developing a language as complex as PharmML is a difficult undertaking and we wanted to make sure that we had some firm principles in place to help when designing the language. We've set these aims and objectives below. PharmML should:

describe the mathematics of a model The language should not include information about the authorship of a model, its update history, or the nature of the disease process or drug that is being modelled. These aspects will be captured by the annotation of the PharmML document and are out of scope of this specification.

describe the task(s) associated with a model The task(s), such as simulation or estimation, to be performed with a model should be encoded in the language.

be declarative The language should describe *what* information is present in a model and *what* the associated task(s) are. It should not describe *how* the information is organised, or *how* the task(s) should be performed.

be platform independent Language elements specific to a particular modelling tool should not be included. For example it should not describe a structural model using a name specific to PREDPP in NONMEM.

serve as an exchange format for the DDMoRe infrastructure The language should either support features required by the infrastructure or provide extension mechanisms so that additional information can be associated with the PharmML document.

provide support for ontological annotation The language should provide a mechanism for it to be annotated with information that is useful to describe the model, but which is beyond the scope of the PharmML document itself.

enable custom extension Provide an extensibility mechanism so that software tools can associate additional, possibly tool specific information, with a PharmML document.

reuse existing standards where appropriate Where an established information standard exists that can be used to represent information within the PharmML document, we should adopt it.

[1] Quoted from `http://www.neuroml.org` on 15 Mar 2013.
[2] Quoted from `http://software.incf.org/software/nineml` on 15 Mar 2013.
[3] Quoted from `http://sed-ml.org` on 15 Mar 2013.
[4] `https://sourceforge.net/projects/libpharmml.ddmore.p`

1.3 PharmML & SO – workflow support and more

PharmML covers the input for a pharmacometric task, i.e. the model, trial design, an according task description and experimental data. Another requirements at the start of the project was to provide additionally a structure to store any type of numerical results coming from a target tool. This is now an ongoing effort and the first draft specification of the so called Standardised Output (SO) is already in use within the framework.

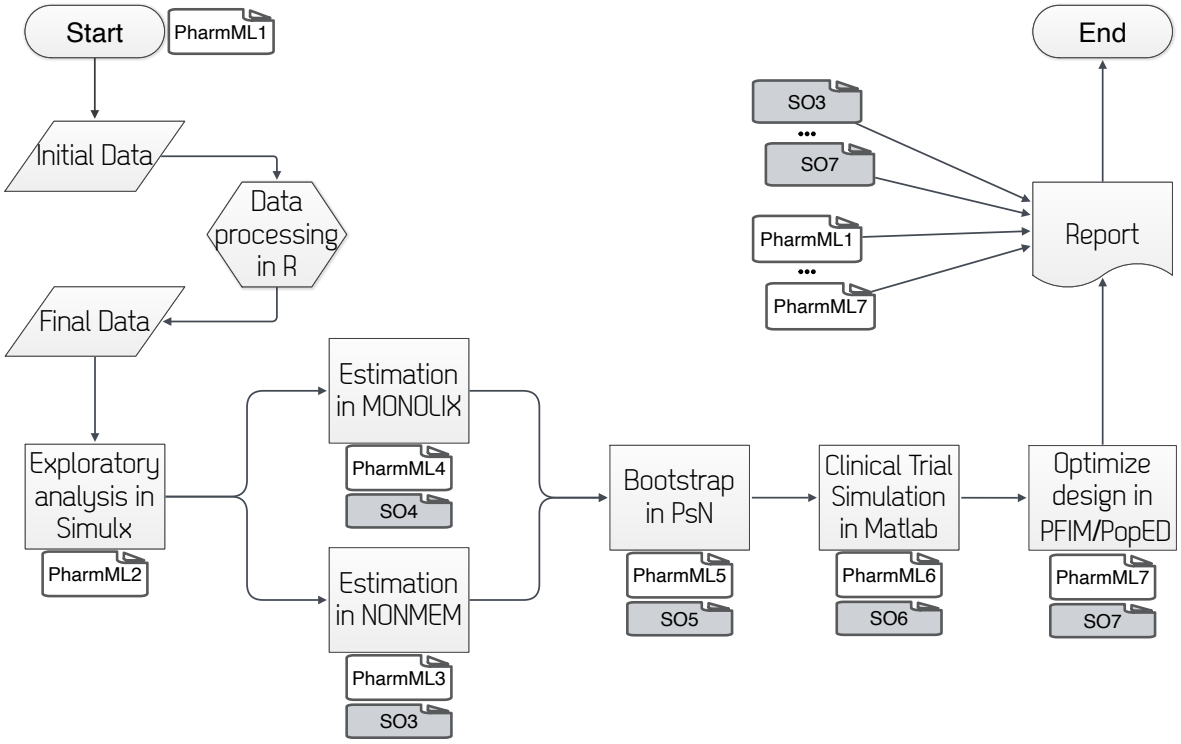


Figure 1.2: PharmML and Standardised Output (SO) supporting a typical workflow in Pharmacometrics featuring major target tools of the DDMoRe platform. Here, it starts with data processing in R, which can consist of data formatting, merging and/or missing-data imputation. After that an exploratory analysis is carried out in Simulx, followed by estimation using either Monolix or NONMEM. Subsequent steps are bootstrapping using PsN, clinical trial simulation in Matlab and finally Optimal Design in either PFIM or PopED. At every step of the workflow, the PharmML model can be stored and the results following each step can be recorded in the corresponding SO file. Documenting workflows in such a detailed way can potentially simplify reporting and ensures reproducibility.

A first public release of the SO is planned in a few months. Together, PharmML and SO are expected to facilitate:

- Smooth and error-free transmission of models between tools
- Use of complex workflows via standardised model and output definitions, see for example Figure 1.2
- Easier reporting and bug tracking
- Improved interaction with regulatory agencies regarding modelling and simulation
- Reuse of existing model resources, e.g. BioModels database
- Development of new tools and methods
- Expanding the community developing/applying pharmacometric models.

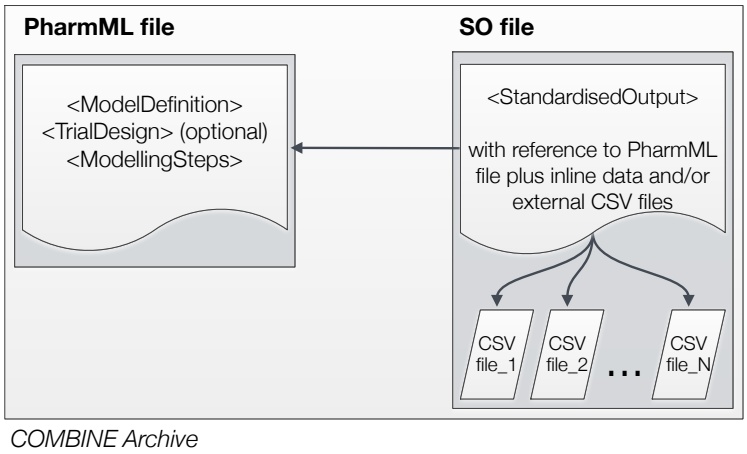


Figure 1.3: PharmML and Standardised Output (SO) relationship.

1.4 Creating PharmML coded models

As indicated in Figure 1.1 once a model is encoded in PharmML it can be shared with any compatible target tool to perform simulation, estimation or optimal design. Modellers will be able to write models using a human readable language also developed within DDMoRe, the Modelling Description Language, MDL (`http://ddmore.eu/mdl`). To facilitate its use, an Integrated Development Environment tool, MDL-IDE, is available, within which the model is automatically translated to PharmML and can be passed to PharmML compatible tools. Development of the MDL and the MDL-IDE is still on-going, but initial results are very promising.

Alternatively, in cases when only the Structural Model is required, modellers can already use the web-editor infix2pharmml (`http://infix2pharmml.sourceforge.net`) to edit complete PharmML models.

1.5 The DDMoRe Consortium

The Drug Disease Model Resources (DDMoRe) consortium aims to promote collaborative drug and disease modelling and simulation research. Its aim is to develop tools and standards that will help the consortium members and later the wider scientific community achieve this goal. Providing PharmML is a key goal of the consortium as it underpins a number of related deliverables of the consortium. In particular:

- The DDMoRe infrastructure in which PharmML is used to exchange models between the different modelling tools.
- The DDMoRe model repository in which PharmML will be used to upload and export models to and from the repository. It will also serve as the storage medium for the repository.
- The DDMoRe library of reference models and data-sets, which will provide models in several therapeutic areas. These models will be encoded using PharmML.

The contribution of the DDMoRe consortium members in guiding and reviewing the standard has been invaluable. As the standard evolves their role in using and then promoting the standard to the wider community will be indispensable.

Peer review is therefore important in the development of PharmML and to date we have hosted a number of face-to-face meetings since the development of PharmML commenced in August 2011. These meetings were:

- A number of DDMoRe consortium meetings in Leiden and Hoofddrop, in the years 2012–2014.
- The DDMoRe technical workshop hosted by Novo Nordisk in Copenhagen, 28–30 Jan 2013.
- Two PharmML technical workshops hosted by University of Pavia, November 2013 & 2014.

1.6 How PharmML was developed

PharmML was designed and implemented by a relatively small group of individuals, but its development has very much been a collaborative process. At the beginning of this project we had a number of development guidelines that we adhered to. We aimed to:

- start with a limited scope and expand the functionality we encode over time.
- drive development using use cases which reflect the current scope.
- test the implemented use cases by generating executable models.
- have frequent review meetings with experts to make sure we are on the right track.
- use existing technology standards if it is possible and reasonable to do so.
- use existing information standards if applicable to avoid re-inventing the wheel.
- make sure the standard is in a form and uses names and terms that make sense to the expert community.

In the second half of 2014 an Interoperability Group within the DDMoRe consortium was appointed to facilitate the testing of PharmML and the interoperability platform driven by it. This typically includes

- Creating a MDL coded model in the MDL-IDE (see Figures 1.1 and 1.2 for an schematic representation of this and the following steps).
- Translating from MDL to PharmML.
- Translating from PharmML to a target tool language, e.g. MLXTRAN for use in Monolix/Simulx or NMTRAN for NONMEM.
- Performing a task in a target tool and the exporting the results into SO.

Initial results are very promising, with a number of models being successfully processed though this pipeline providing a proof-of-concept for the interoperability concept. Figure 1.4 gives an overview of the development timeline, more details can be find in the detailed changelog in chapter 10.

1.7 Imperative or Declarative?

In developing PharmML we have designed it to be a declarative language (see section 1.2). While we feel this is the best approach to take for an information exchange langage, it does present us with number of challenges when dealing with NONMEM, the leading tool in the field. Despite having a specifically defined language (NM-TRAN [Beal et al., 2009]), NONMEM offers a lot of flexibility to the user, and experienced users can make NONMEM do things that it was never designed to do, to a large extent because the imperative approach used in NONMEM facilitates this.

The challenge is to convert from the imperative to the declarative language, because there are many ways to do the same thing in the imperative language. Therefore, generating a PharmML document from a NONMEM control stream is challenging.

1.8 The Evolution of PharmML

This document represents the second public release of PharmML. The first one was released on 21^{st} November, 2013, [Moodie et al., 2013].

Any piece of software is upgraded as users request new features and developers find better ways to do the same thing. A successful standard is no different and with success in mind we expect PharmML to evolve and change further as it is subject to the same influences. To manage this process we have adopted the following strategy to record versions of the PharmML specification.

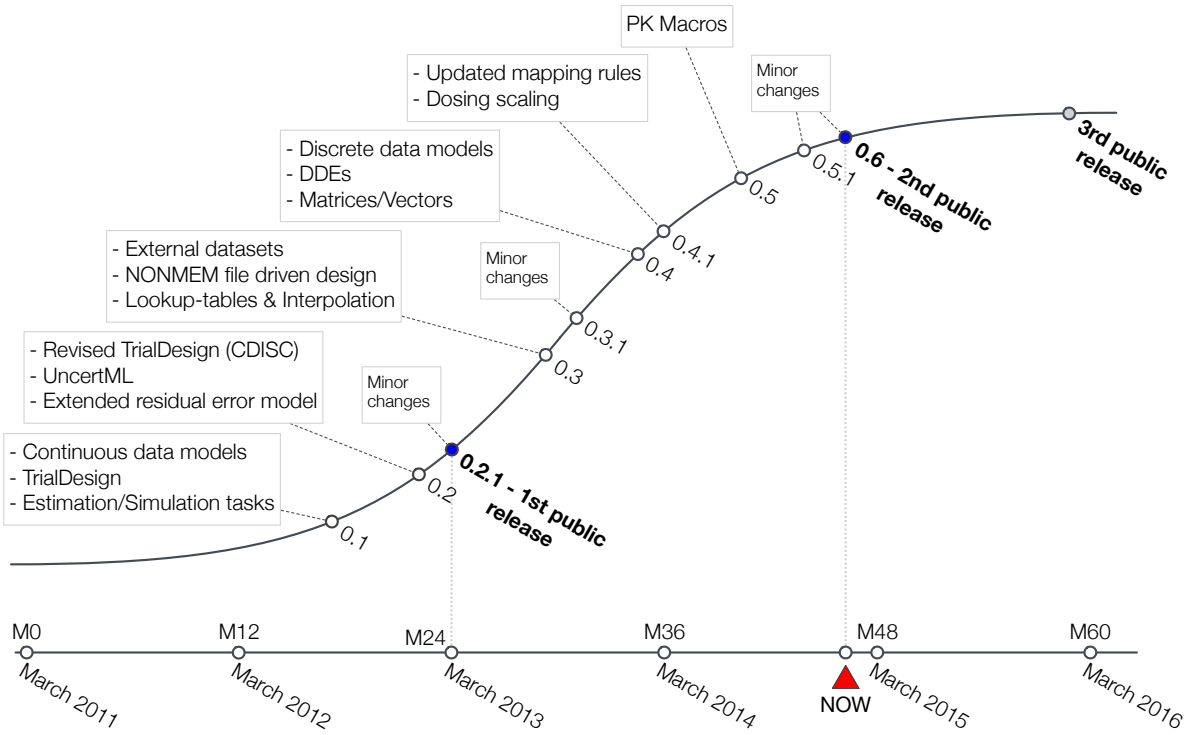


Figure 1.4: Timeline of the PharmML development process, deliverable schedule and version features.

Version number To record changes in the specification we will use the following three level numbering system, of the form $x.y.z$. The levels correspond to the following types of revision:

x **major** Significant new features or radical change of design.

y **minor** New features or evolutionary design changes.

z **patch** Error corrections.

1.9 Feedback

User and developer feedback is important. Typically, this feedback will be in the form of a specific issue: either to report defects identified in the specification or to request new features. Either way the specific issues can be submitted to the tracker at: `https://github.com/pharmml/pharmml-spec/issues`. In some cases it is practical to raise an issue that is broader than a specific issue or requires some discussion within the community. Here the contact is the PharmML forum at `http://pharmml.org`.

Chapter 2

Scope of PharmML

2.1 Introduction

The scope of pharmacometric models is very wide, as such models can be empirical as well as mechanistic; describe continuous as well as discrete data types; and be deterministic, stochastic or a mixture of both. It is a challenging endeavour to accommodate this variety of possibilities under one computational standard, therefore it is indispensable to split the task into multiple steps of subsequent specifications and to define precisely the scope of every release.

In this chapter we define the scope of the functionality in PharmML, i.e. what information a PharmML document can represent. As is common practise in software engineering we have described the functionality as a set of "features". It is important to remember that this specification is the second public release of PharmML and over remaining time we expect additional features to be provided by future releases of the language.

The language is organised into three sections, which we believe naturally describe the logical organisation of a pharmacometric model and its associated tasks. Consequently we have grouped the features to match this organisation. The sections are:

Model Definition A description of the model, incl. the structural model, the model parameters, relevant covariates, the variability components, and the observations.

Trial Design A description of the design of a clinical trial associated with the model (for example a trial from which data is available to estimate the parameters of the model or a trial to be simulated with the model).

Modelling Steps A description of steps or tasks performed with the model. Typically this describes how the model has been used, for example to estimate its parameters or to perform a simulation.

2.2 Model Definition

2.2.1 Structural Model

PharmML can encode:

- A structural model defined by a set of algebraic equations. Typically the explicit solution to a simple PK model, or a dose-response model.
- A structural model defined by a system of ODEs with initial conditions.
- Delay Differential Equations (DDEs).
- Compartmental PK models implemented using PK macros.

2.2.2 Covariate Model

PharmML can encode:

- Continuous covariates. These can be sampled from a probability distribution, used with an applied transformation and interpolated.
- Categorical covariates. These can also be sampled from a probability distribution.

2.2.3 Parameter Model

PharmML can encode:

- Gaussian models with linear or non-linear covariate model.
- General, equation based model such as those described in [Keizer and Karlsson, 2011].
- Random effects at arbitrary levels of variability.
- Correlation of the random effects, described by a correlation or covariance matrix.

2.2.4 Variability Model

PharmML supports the following levels of variability:

- Between-Subject Variability (BSV). Aka inter-individual variability (IIV).
- Inter-Occasion Variability (IOV). Such as within-subject variability.
- Higher levels of variability above BSV. Such as variability between countries or centres.
- Lower levels of variability below IOV. Such as variability between sub-occasions within occasions.

2.2.5 Observations Model

PharmML supports the following observation model:

- Continuous data models. A residual error model applied to one or more observation variables..
- Discrete data models such as
  - Count data models. Such as Poisson, negative binomial, zero-inflated Poisson models etc.
  - Nominal and ordered categorical models. Logistic regression, proportional odds models etc.
  - Time-to-event models.

2.3 Trial Design

2.3.1 Dataset based

PharmML supports external NONMEM/Monolix datasets which can be sourced to provide the full study design and the according experimental records.

2.3.2 `<TrialDesign>` based

PharmML can encode the following features of a trial design *explicitly*:

- Bolus and infusion dosing.
- Multiple dosing regimens including mixed bolus and infusion.
- Single, Repeated and Steady state dosing.
- Dosing to more than one compartment.
- Simple, parallel and cross-over designs.
- Washout periods.
- Occasions – defined by time interval within a treatment epoch.
- Trials with different centres or other levels of organisation above study arms

2.4 Modelling Steps

PharmML can encode the following features related to the task(s) associated with a model:

- Estimation utilising the maximum likelihood principle.
- Simulation of the model.

2.5 General

Metadata annotation Provides support to enable metadata descriptions of the PharmML document.

Extension mechanism Provides support to enable the extension of the PharmML document.

Chapter 3

Mathematical Representation

3.1 Introduction

This chapter deals with the mathematical description of models which can be encoded in PharmML. The figure 3.2 visualises the task every modeller is faced with – find a model (solid line) which explains the experimental data (diamonds). A perfect mathematical model, given error free experimental measurements, would explain the underlying mechanism and therefore fit every measurement. The complexity of the human body makes detailed mechanistic representation impossible, so the models we use are only approximations of a physiological system under consideration with multiple sources of uncertainty we have to account for, see Figure 3.1.

$$\underset{\text{Data}}{\text{Experimental}} = \underset{\text{Prediction}}{\text{Model}} + \underset{\text{Error}}{\text{Residual}}$$

Figure 3.1: A basic equation visualising the relationship between experimental data and model prediction.

Additional aspect is the difference between individual and population approach. In the former case, usually when dealing with animal or preclinical studies, frequently sampled data is available and one can estimate individual's PK and PD parameters, see Figure 3.2. In clinical practice however, the situation is quite different, Figure 3.3. More data records in total are available but it's often impossible to estimate subject specific parameters. Instead, the data provides valuable information on the inter-individual variability.

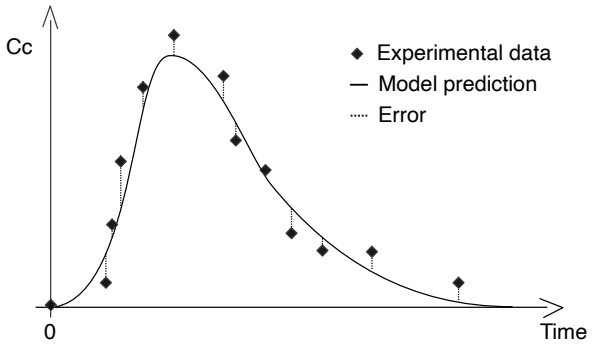


Figure 3.2: Frequently sampled individual PK data. The black diamonds stand for experimental data, the solid line for the time course of concentration as predicted by a mathematical model.

3.2 Non-linear mixed effect models

The approach which proved to be very effective for the analysis of population data is that of **nonlinear mixed effect models**, NLME [Bonate, 2011, Lavielle, 2014]. The nonlinearity means that it can handle

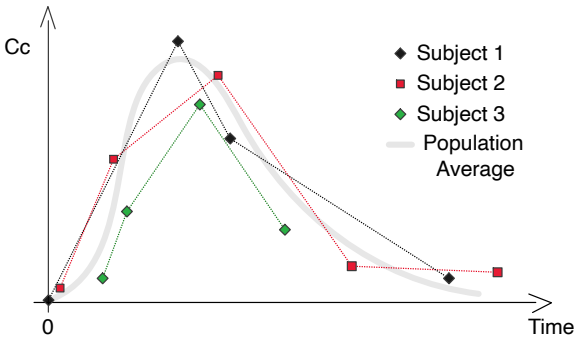


Figure 3.3: Population PK data – for three subjects with different observation times and varying characteristics, such as area under the curve, maximum values, time of the maximum etc. The grey line of the population average is to be estimated along with individual estimates.

virtually any type of structural model, usually highly non-linear. Additionally one can consider population or subject related factors, known *a priori* or collected during the study, which can be divided into two groups:

- **fixed effects** – population averages, e.g. typical/population value for volume, and other group or even subject specific explanatory variables, such as treatment groups, gender or weight, see covariate model in section 3.6.5.
- **random effects** – subject/occasion specific, e.g. *inter-individual* or *inter-individual within subject within centre variability*, see section 3.5 on variability.

The notation of fixed versus random effects might be at first confusing as the former account for population, group but also subject characteristics. As explained in the parameter model section, the values of individual characteristics are subject specific features but the parameters assigned to them are identical for a group or population.
The structure of this chapter is the following, in section 3.3 we formulate a general nonlinear mixed effects model for continuous data, section 3.4 introduces the structural model, section 3.5 discusses the variability as nested hierarchy, section 3.6 is about the parameter model, with discussion on correlation of random effects, covariate model and a comparison of equivalent representations of the parameter model and section 3.7.1.1 is about the residual error model.

3.3 Continuous data model

A general nonlinear mixed effects model for continuous data for N subjects and n_i measurements per subject i reads as follows, [Lavielle, 2014]:

$$\underbrace{y_{ij}}_{\substack{\text{Experimental}\\ \text{data}}} = \underbrace{f(x_{ij},\psi_i)}_{\substack{\text{Model}\\ \text{prediction}}} + \underbrace{g(x_{ij},\psi_i,\xi)\,\epsilon_{ij}}_{\text{Error}} \qquad 1 \le i \le N, \quad 1 \le j \le n_i \tag{3.1}$$

with

- y_{ij} – j^{th} observation for subject i
- f – structural model prediction
- x_{ij} – regression variables, e.g. *time* or *concentration*
- ψ_i – individual parameters
- ϵ_{ij} – residual error
- g – standard deviation of the residual error
- ξ – parameters of the residual model

With ϵ_{ij} being normal distributed with mean 0 and variance 1, y_{ij} is also normally distributed with mean $f(x_{ij},\psi_i)$ and the standard deviation $g(x_{ij},\psi_i,\xi)$.

3.4 Structural model

This section deals with the first term of the right hand side in eq.3.1

$$f(x_{ij}, \psi_i)$$

i.e. the model prediction.

It can be formulated as a simple algebraic equation (e.g. Hill equation) or complex physiology-based PK model implemented as system of ODEs. When defined in such framework, this deterministic model for an individual will later be embedded in a statistical model. Other approaches, such as SDE-based structural models are not supported in this specification.

Example As an example of a structural model we consider a combined PK/PD model,

- PK – oral one-compartmental model
- PD – turnover model, so called I_{max} model

with the following model parameters ka, V, CL, $Imax$, $IC50$, Rin and $kout$.

$$\begin{aligned}
k &= CL/V \\
\frac{dAd}{dt} &= -ka \times Ad \\
\frac{dAc}{dt} &= ka \times Ad - k \times Ac \\
\frac{dE}{dt} &= Rin \times \left(1 - \frac{Imax \times Cc}{Cc + IC50}\right) - kout \times E \\
\text{Initial condition: } E(t=0) &= Rin/kout \\
Ad(t=0) &= DoseSize \\
Ac(t=0) &= 0; \\
Cc &= Ac/V
\end{aligned}$$

Alternative formulation 1 The PK model can, in this case, be formulated as an algebraic equation because an analytic solution exists, i.e.

$$C(t) = \frac{D}{V}\frac{ka}{ka-k}\left(e^{-k(t-t_D)} - e^{-ka(t-t_D)}\right)$$

Alternative formulation 2 PK macros offer an equation-free option to implement PK compartmental models. For details see chapter5.

3.4.1 Delayed Differential Equations (DDE)

Additionally to the ODEs and algebraic equations, version 0.5.1 of PharmML offers the possibility to use Delayed Differential Equations. Similarly to initial conditions required for an ODE, the so-called *history* has to be defined. If a model is defined for times $t \geq t_0$, the history specifies the values of the model variables for the time $\leq t_0$. The following simple example, from [Lavielle and Lixoft Team, 2014], shows how such model and the according history can be formulated.

$$\begin{aligned}
\frac{dA}{dt} &= r - c \times A \times B - k \times A \\
\frac{dB}{dt} &= c \times A \times B - A(t-d)
\end{aligned}$$

with history $A_0 = r/k$ and $B_0 = 0$ for $t \leq 5$. d is the discrete delay parameter.

3.5 Nested hierarchy as the random variability structure

This section describes the variability structure of the random effects and the related naming convention. It is largely based on the discussions and conclusions from the DDMoRe Copenhagen focus meeting (Section 1.5). Accordingly, in the following we will distinguish:

- (related to the observations) – *residual variability*, also known as *intra-individual variability* and
- (related to the parameters) – *inter-individual* and *inter-occasion variabilities*

The former is described in the section 3.7.1.1, while the latter is described in this section.

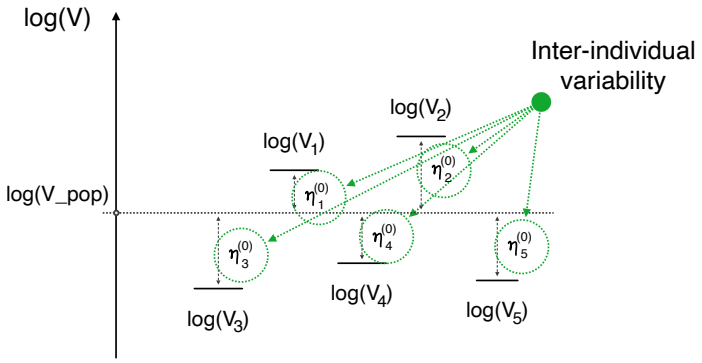


Figure 3.4: Inter-individual variability typically occurring in an experiment, here $\log(V_{i=1\ldots5})$ i.e. values for five subjects, varying around a typical value $\log(V_{pop})$, are shown.

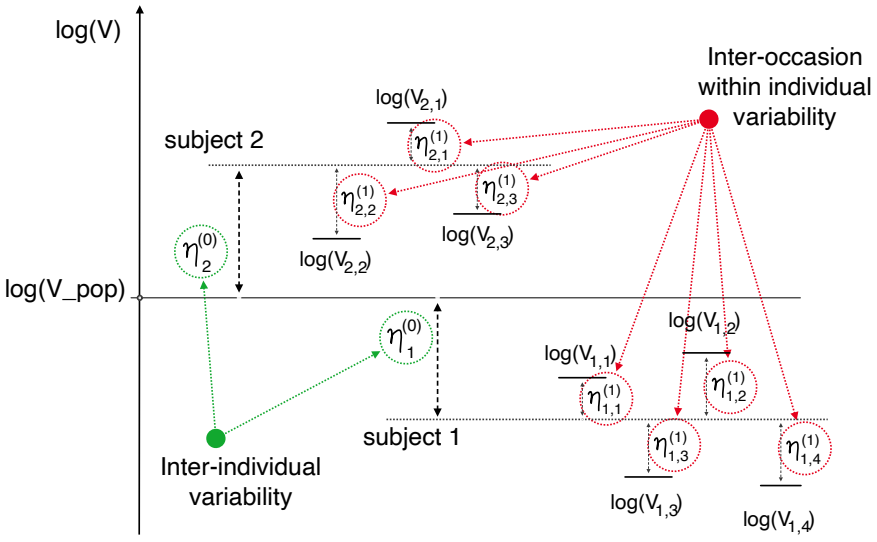


Figure 3.5: Subject variability level, (0), and within-subject (or occasion) variability level, (1), typically occurring in an experiment, with index i for subjects and k for occasion. Here two subjects only are visualised, each of them having four or three occasions, respectively.

3.5.1 Motivation

One way to look at variability is to consider the following simple experiment: in this experiment, we estimate the volume of distribution in five subjects. Following a drug administration we collect blood samples over a time interval and estimate each subject's PK parameters. The result will be a set of five individual estimates such as those in Figure 3.4. The values vary around a certain typical/population value. It is apparent that

the only variability source is the fact that these are different persons, i.e. we have a rough estimate of the so called *inter-individual* variability.
As an extension of this setup, we can now consider different number of occasions, when the PK parameters are estimated for each subject. If we restrict the discussion to two subjects only, each of them having three or four occasions, respectively, we can illustrate the results such those in Figure 3.5. Repeatedly performing the same experiment for each subject is equivalent to create an additional level of variability, the *inter-occasion within individual* variability.
Similarly, one can add e.g. 'country' or 'study centre' as new variability levels. If a clinical trial has been conducted in various countries or centres, it is reasonable to ask if the geographic location influences the outcome of the study.

3.5.2 General case

As a generalisation of the examples described above, one can derive the *nested hierarchy* (also known as *inclusion hierarchy*) of the variability structure of random effects. It can be visualised as a tree or alternatively using a Venn diagram, see Figures 3.6 and 3.7.
The tree representation consists of *nodes* and *links* or *edges*. It has the advantage that it visualises the whole structure explicitly from the top level, the *root* node, down to lowest level of the variability. It provides immediate insights needed to understand or to verify the setup of a trial design. However, in case of a very complex structure, with high number of levels and/or subjects, it can become very large, making the tree difficult to represent in a typical document. It this case showing only partial branches will be more helpful, e.g. Figure 3.6. On the contrary, the Venn diagram visualises the levels only, and it might be more suited for the complex cases. Usually, the variability structure consists of only one or two levels, e.g. *individual* or {*individual*, *occasion*}, see examples below.

The *root*, i.e. the top node in the tree structure, stands for the population/typical value of a parameter. Following the current nomenclature, every subsequent variability level is either 'positive' or 'negative' dependent on its position relative to the reference 'subject level', denoted as 0 – the level 'zero'. Each level has a covariance matrix associated with it, i.e.

- Ω^{-n} – for levels above the reference level – their names will vary according to the nature of the levels. For example the variability on country level is called 'between-country variability'.
- Ω^{0} – reference level[1], also called BSV (between subject variability) or IIV (inter-individual variability).
- Ω^{+n} – for levels below the reference level – called IOV (inter-occasion variability) or WSV (within-subject variability).

The number of levels will vary dependent on the nature of the study. Cases without or with only positive/negative levels are possible. Please note that PharmML doesn't require numbers to be assigned to the various levels of variability. Instead the user can define meaningful identifiers.

Example 1 This example handles the simplest scenario, with only one level of variability: *subject*–level, see Figure 3.8. The following symbols are used

- i – subject index, $1 \leq i \leq N$

with N_l – number of subjects.
The typical parameter model, without covariate, reads as follows:

$$\log(V_i) = \log(V_{pop}) + \eta_i^{(0)}$$

or alternatively:

$$V_i = V_{pop}\, e^{\eta_i^{(0)}}$$

with $\eta_i^{(0)} \sim \mathcal{N}\big(0, \Omega^{(0)}\big)$.

[1]Target tools such as NONMEM or Monolix assume the subject level as the reference level by default. PharmML doesn't make such assumption and the *zero* level can be explicitly defined and is required if more then one level exist.

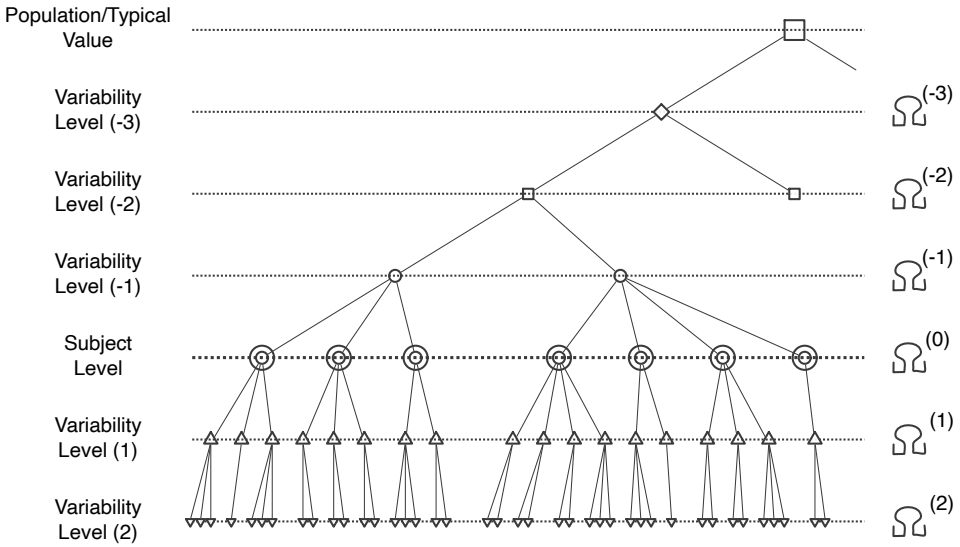


Figure 3.6: General nested hierarchy of the variability structure – as tree. Note that PharmML doesn't require or use numbers to be assigned to the various levels of variability. Instead the user can define meaningful identifiers such as *subject* or *iiv* for level (0), *occasion* or *iov* for level (1), *study* for level (-1) etc.

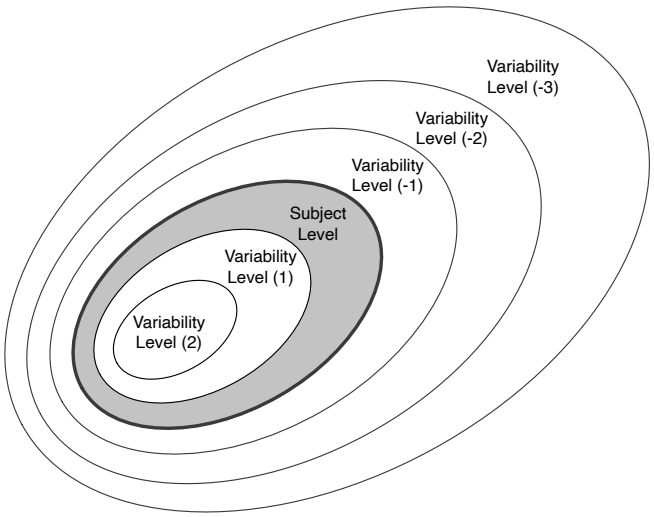


Figure 3.7: General nested hierarchy of the variability structure – as Venn diagram.

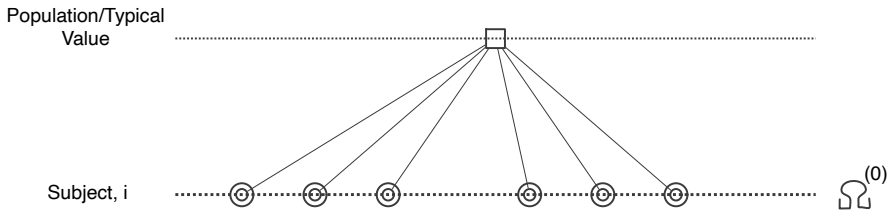


Figure 3.8: Example 1 – single level of variability: *subject* – (0) level.

Example 2 In this example there are three levels of variability: {*centre, subject, occasion*}, see Figure 3.9. Following symbols are used:

- l – centre index, $1 \leq l \leq L$

- i – subject index, $1 \leq i \leq N_l$
- k – occasion index, $1 \leq k \leq N_{li}$

with

- L – number of centres
- N_l – number of subjects in centre l
- N_{li} – number of occasions in subject i in centre l

The parameter model, without covariate, reads as follows:

$$\log(V_{lik}) = \log(V_{pop}) + \eta_l^{(-1)} + \eta_{li}^{(0)} + \eta_{lik}^{(+1)}$$

or alternatively:

$$V_{lik} = V_{pop}\, e^{\eta_l^{(-1)}}\, e^{\eta_{li}^{(0)}}\, e^{\eta_{lik}^{(+1)}}$$

with

$$\eta_l^{(-1)} \sim \mathcal{N}\big(0, \Omega^{(-1)}\big), \quad \eta_{li}^{(0)} \sim \mathcal{N}\big(0, \Omega^{(0)}\big), \quad \eta_{lik}^{(+1)} \sim \mathcal{N}\big(0, \Omega^{(+1)}\big)$$

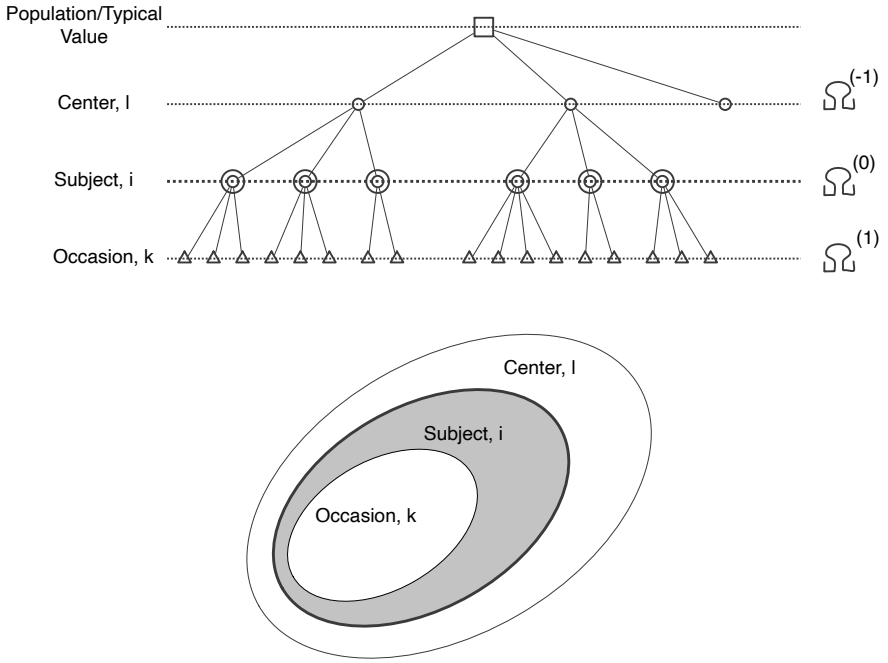


Figure 3.9: Example 1 – three levels of variability: {*centre (+1), subject (0), occasion (-1)*}

Example 3 In this example there are four levels of variability: {*country, centre, subject, occasion*}, see Figure 3.10. The symbol list is extended by one for 'country' as follows:

- m – country index, $1 \leq m \leq M$
- l – centre index, $1 \leq l \leq N_m$
- i – subject index, $1 \leq i \leq N_{ml}$
- k – occasion index, $1 \leq k \leq N_{mli}$

with

- M – number of countries
- N_m – number of centres in country m
- N_{ml} – number of subjects in centre l in country m
- N_{mli} – number of occasions in subject i in centre l in country m

The parameter model reads as follows:

$$\log(V_{mlik}) = \log(V_{pop}) + \eta_m^{(-2)} + \eta_{ml}^{(-1)} + \eta_{mli}^{(0)} + \eta_{mlik}^{(+1)}$$

or alternatively:

$$V_{mlik} = V_{pop}\, e^{\eta_m^{(-2)}} e^{\eta_{ml}^{(-1)}}\ e^{\eta_{mli}^{(0)}}\ e^{\eta_{mlik}^{(+1)}}$$

with

$$\eta_m^{(-2)} \sim \mathcal{N}\big(0, \Omega^{(-2)}\big), \quad \eta_{ml}^{(-1)} \sim \mathcal{N}\big(0, \Omega^{(-1)}\big), \quad \eta_{mli}^{(0)} \sim \mathcal{N}\big(0, \Omega^{(0)}\big), \quad \eta_{mlik}^{(+1)} \sim \mathcal{N}\big(0, \Omega^{(+1)}\big)$$

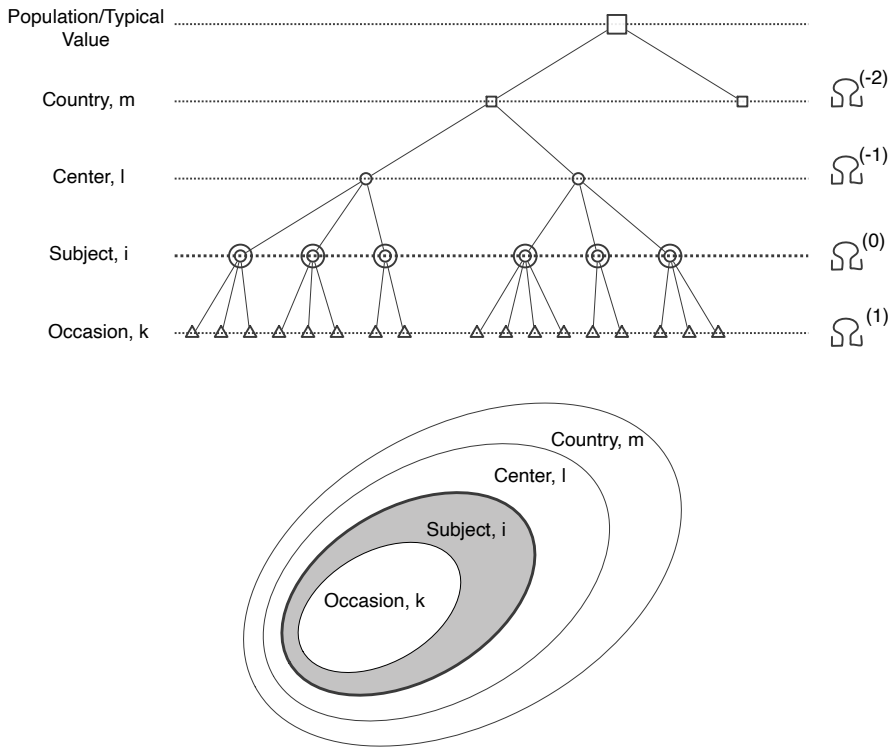


Figure 3.10: Example 2 – four levels of variability: {*country (-2), centre (-1), subject (0), occasion (+1)*}

3.6 Parameter model

The following section outlines the parameter model. We consider three types of parameter model:

- Type 1. Implicit model

$$\psi_i = H(\beta, C_i, \eta_i)$$

This is the most general form of a parameter model with no constraints on the function H. It is an implicit equation as it doesn't allow an easy interpretation of its elements in contrast to the two following forms.

- Type 2. Gaussian model with general covariate model

$$h(\psi_i) = H(\beta, C_i) + \eta_i$$

Here the parameter is normally distributed up to a transformation h with a general covariate model and additive random effects.

- Type 3. Gaussian model with linear covariate model

$$h(\psi_i) = h(\psi_{pop}) + \beta\, C_i + \eta_i$$

This is a special case of the models above which allows for the most detailed interpretation as explained in the following section.

with

- ψ_i – individual parameter
- ψ_{pop} – typical or population mean parameter
- η_i – random effect(s)
- β – fixed effect(s)
- C_i – covariate(s)
- H – arbitrary function
- h – function which transforms the model on both sides, e.g. log, logit, probit.

3.6.1 Discussion and examples of Type 1 models

This model type is the most flexible one, able to accommodate

- multiple fixed effects
- multiple random effects and
- an arbitrary (nonlinear) covariate model.

Example Let's consider a complex clearance model as introduced in [Beal et al., 2006], which contains

- four fixed effects $\theta_1, \cdots, \theta_4$
- three continuous covariates $WT, AGE, SECR$
- one categorical covariate ICU
- three random effects $\eta_{1i,met} \sim \mathcal{N}(0, \omega^2_{1,met}), \eta_{2i,met} \sim \mathcal{N}(0, \omega^2_{2,met}), \eta_{3i,ren} \sim \mathcal{N}(0, \omega^2_{3,ren})$

The model is composed of

1. the average metabolic clearance which reads

$$CL_{met_{average}} = WT \times \frac{\theta_1 - \theta_2 \times Cpss_2}{\theta_3 + Cpss_2}$$

extended with random effects representing a patient being from an ICU (intensive care unit) or else

$$CL_{i,met} = CL_{met_{average}} + (1 - ICU)\, \eta_{1,i} + ICU\, \eta_{2,i}$$

i.e.

$$CL_{i,met} = \begin{cases} CL_{met_{average}} + \eta_{1,i} & \text{for} \quad ICU = 0 \quad \text{i.e. patient not from ICU} \\ CL_{met_{average}} + \eta_{2,i} & \text{for} \quad ICU = 1 \quad \text{else} \end{cases}$$

2. and average renal clearance which reads

$$CL_{ren_{average}} = \theta_4 \times RF \quad \text{with} \quad RF = WT \times \frac{1.66 - 0.011 \times AGE}{SECR}$$

so the clearance for subject i amounts to

$$CL_{i,ren} = CL_{ren_{average}}(1 + \eta_{3,i})$$

The complete model, combining (1) and (2), for an individual's clearance then reads

$$CL_i = CL_{i,met} + CL_{i,ren}.$$

This model, although fully flexible, is difficult to break into meaningful sub-components. This is an entirely different situation for the following model types, where clearly defined sub-components can be separately stored and annotated.

3.6.2 Discussion and examples of Type 2 models

Here, we consider normally distributed parameters, up to a transformation h, i.e. normal, log-normal or logit-normally distributed with identity, the natural logarithm or the logit as transformation, respectively.

Compared to the Type 1 parameter model, the Type 2 parameter model has a more structured additive form:

$$h(\psi_i) = \underbrace{H(\beta, C_i)}_{\substack{\text{non-linear covariate} \\ \text{model}}} + \underbrace{\eta_i^{(0)} + \eta_{ik}^{(+1)} + \dots}_{\substack{\text{IIV and other} \\ \text{levels of variability}}}$$

Accordingly a model for an individual parameter consists of

- the left-hand transformation, h
- a non-linear covariate model, i.e. any function, H, of fixed effects, β, and categorical or continuous covariates, C_i, e.g. *Sex* or *Weight*, and
- random effects, η, for *inter-individual, inter-occasion* and/or other levels of variability (see section 3.5).

Example The following example is taken from the 'Fisher/Shafer NONMEM Workshop', and in NMTRAN code reads

```
WTE = THETA(1) * WT / (THETA(2)+ WT)
V = (THETA(3) + WTE) * EXP(ETA(1))
```

After taking the logarithm of both sides we get

$$\log(V_i) = \log\left(\theta_3 + \frac{\theta_1 \times WT_i}{\theta_2 + WT_i}\right) + \eta_{V,i}.$$

3.6.3 Discussion and examples of Type 3 models

Here, we again consider normally distributed parameters, up to a transformation h, i.e. normal, log-normal or logit-normally distributed with identity, the natural logarithm or the logit as transformation, respectively.

The Type 3 parameter model has a very convenient fully additive form, which separates all of the sub-components, making it very easy to understand and process:

$$h(X_i) = h(X_{pop}) + \underbrace{\beta\, C_i}_{\substack{\text{linear covariate} \\ \text{model}}} + \underbrace{\eta_i^{(0)} + \eta_{ik}^{(+1)} + \dots}_{\substack{\text{IIV and other} \\ \text{levels of variability}}}$$

Accordingly a model for an individual parameter consists of

- a parameter transformation, h
- a typical or population mean value of the parameter, X_{pop}

- a linear covariate model, $\beta\, C_i$, with
  - fixed effects, β, and
  - categorical or continuous covariates, C_i, e.g. *Sex* or *Weight*
- random effects, η, for *inter-individual, inter-occasion* and/or other levels of variability (section 3.5).

See Figure 3.11 for an example of the linear relationship between a parameter and a continuous covariate and one, *inter-individual*, level of variability.

Example Let's consider volume, V, as a log-normally distributed parameter with two covariates *Sex* and *Weight* and with three levels of variability as discussed in Example 3 in section 3.5 (see Figure 3.9), which can be represented by the equation:

$$V_{lik} = V_{pop} e^{\beta_{V,1} 1_{Sex_i=F}} \left(\frac{W_i}{70}\right)^{\beta_{V,2}} e^{\eta_{l,V}^{(-1)}} e^{\eta_{li,V}^{(0)}} e^{\eta_{lik,V}^{(+1)}}$$

or alternatively as

$$\underbrace{\log(V_{lik})}_{\substack{\text{transformed}\\ \text{individual value}}} = \underbrace{\log(V_{pop})}_{\substack{\text{transformed}\\ \text{typical value}}} + \underbrace{\beta_{V,1} 1_{Sex_i=F}}_{\substack{\text{categorical}\\ \text{covariate model}\\ \text{for Sex}}} + \underbrace{\beta_{V,2} \log\left(\frac{W_i}{70}\right)}_{\substack{\text{continuous}\\ \text{covariate model}\\ \text{for Weight}}} + \underbrace{\eta_{l,V}^{(-1)}}_{\substack{\text{inter-centre}\\ \text{variability}}} + \underbrace{\eta_{li,V}^{(0)}}_{\substack{\text{inter-}\\ \text{individual}\\ \text{within centre}\\ \text{variability}}} + \underbrace{\eta_{lik,V}^{(+1)}}_{\substack{\text{inter-occasion}\\ \text{within individual}\\ \text{within centre}\\ \text{variability}}}$$

with

$$\eta_{l,V}^{(-1)} \sim \mathcal{N}\big(0, \Omega^{(-1)}\big), \quad \eta_{li,V}^{(0)} \sim \mathcal{N}\big(0, \Omega^{(0)}\big), \quad \eta_{lik,V}^{(+1)} \sim \mathcal{N}\big(0, \Omega^{(+1)}\big).$$

The equation for V_{lik} represented in the additive form is clearly easier to understand and one can read out the following information from it:

- the parameter transformation, the natural logarithm, *log*
- the typical volume, V_{pop}
- the two linear covariate models, $\beta_{V,1}C_1$ and $\beta_{V,2}C_2$ with
  - a fixed effect for the categorical covariate, $\beta_{V,1}$
  - a categorical covariate, $1_{Sex_i=F}$
  - a fixed effect for the continuous covariate, $\beta_{V,2}$
  - a continuous covariate, $C_2 = \log(W/W_{pop})$ with $W_{pop} = 70$
- multiple random effects
  - a random effect above the subject level for *inter-centre* variability, $\eta_{l,V}^{(-1)}$
  - a random effect at the subject level for *inter-individual within centre* variability, $\eta_{li,V}^{(0)}$
  - a random effect below the subject level for *inter-occasion within individual within centre* variability, $\eta_{lik,V}^{(+1)}$.

3.6.4 Correlation of random effects

Correlation of random effects means that the transformed parameters. e.g. $\log(V_i)$ and $\log(CL_i)$ are correlated as well (although the relationship is not straightforward; see also the discussion below on correlation and covariates). There are two alternative ways to define the correlation, using either

- a correlation matrix, R, or
- a variance-covariance matrix, Ω.

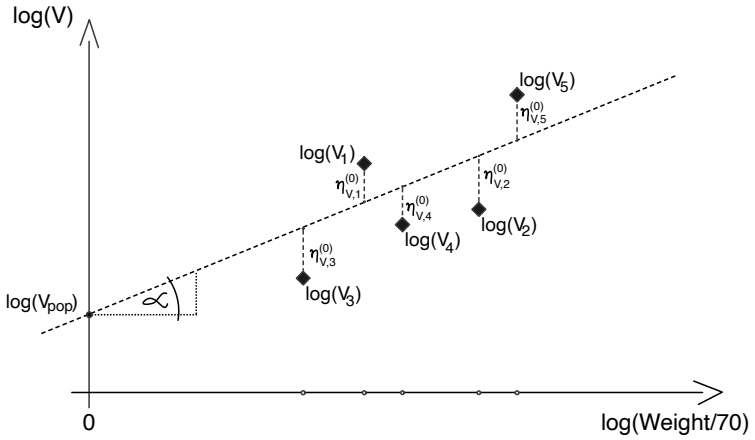


Figure 3.11: The linear relationship between the parameter and a continuous covariate after application of appropriate transformations $V_i \longrightarrow \log(V_i)$ and $W \longrightarrow \log(W/70)$ with $\beta_{V,2} = \tan\alpha$, the slope of the regression line, and $\log(V_{pop})$ as the y-axis intercept.

Correlation matrix In this case it is sufficient to define the non-zero correlation coefficients, e.g. $\rho_{V,CL}$. All other off-diagonal correlation coefficients will be assumed to be equal to 0. For a simple one-compartment oral PK model with parameters ka, V, CL, and a correlation between CL and V the full correlation matrix reads as follows

$$R = \begin{pmatrix} 1 & 0 & 0 \\ & 1 & \rho_{V,CL} \\ & & 1 \end{pmatrix}$$

Variance-covariance matrix Alternatively, the variance-covariance matrix for the model

$$\Omega = \begin{pmatrix} \omega_{ka}^2 & \omega_{ka,V} & \omega_{ka,CL} \\ & \omega_V^2 & \omega_{V,CL} \\ & & \omega_{CL}^2 \end{pmatrix} = \begin{pmatrix} \omega_{ka}^2 & 0 & 0 \\ & \omega_V^2 & \omega_{V,CL} \\ & & \omega_{CL}^2 \end{pmatrix}$$

is providing the necessary information due to the reletionship

$$\mathrm{Cov}(p_i,p_j) = \sigma_i\sigma_j\mathrm{Corr}(p_i,p_j) = \sigma_i\sigma_j\ \rho_{i,j} \quad \text{i.e.} \quad \omega_{V,CL} = \omega_V\omega_{CL}\rho_{V,CL}$$

in which case it is enough to define Ω to cover the full correlation structure.

3.6.5 Covariate model

The covariate model accounts for systematic or known subject characteristics such as treatment group, gender or body weight. Accordingly, the model can be defined for discrete and continuous covariates and is the place where one category of fixed effects is defined (the other being the population averages, e.g. V_{pop}). Of course, the values of individual characteristics (weight or sex) are subject specific but the parameters assigned to them are identical for a group or population.

As described in the example above the contribution of the continuous covariate $Weight$ to the parameter value is formulated as $\beta_{V,2}\log(W_i/70)$ (see figure 3.11). The figure illustrates the linearity after the appropriate transformation of the parameter, $V_i \longrightarrow \log(V_i)$ and $W \longrightarrow \log(W/70)$ with $\beta_{V,2} = \tan\alpha$, the slope of the regression line, and $\log(V_{pop})$ as the y-axis intercept.

In the estimation case the values for the covariate are provided for each individual. In the case of a simulation (see example 9.2.2) its probability distribution has to be estimated. The information we have to provide is summarised in the following

Continuous covariate model

$$\begin{aligned} Covariates &= Weight \\ CovariatesType &= Continuous \\ CovariatesPopDistribution\{1\} &\sim \text{Normal}(pop_{Weight}, \omega_{Weight}) \\ \text{with} \quad pop_{Weight} &= 70.07 \\ \omega_{Weight} &= 14.09 \\ CovariatesTransf &= \log(Weight/70) \end{aligned}$$

Analog information has to be provided in the case of a categorical covariate, such as *Sex* and is summarised for a simple example in the following

Categorial covariate model

$$\begin{aligned} Covariates &= Sex \\ CovariatesType &= Categorical \\ CategoriesNumber &= 2 \\ Categories &= \{F, M\} \\ RefCategory &= F \\ RefCategProbability &= 14/36 \end{aligned}$$

3.6.6 Equivalent representations of the parameter model

Every parameter model represented in the Type 3 format discussed before has at least three mathematically equivalent representation forms, which will be presented and discussed in terms of advantages and disadvantages in the following. It is important to understand these different representation forms, as they explain the different forms of notation used in different software tools. Here, we concentrate on NONMEM and MONOLIX only.

3.6.6.1 Log-Normal distributed

For a **log-normal** distributed parameter, e.g. V, the equivalent representations read

$$\begin{aligned} &(1)\eta_{i,V} \sim \mathcal{N}(0, \omega_V); \quad V_i = V_{pop}\, e^{\eta_{i,V}} \\ &(2)\eta_{i,V} \sim \mathcal{N}(0, \omega_V); \quad \log(V_i) = \log(V_{pop}) + \eta_{i,V} \\ &(3)\log(V_i) \sim \mathcal{N}\big(\log(V_{pop}), \omega_V\big) \end{aligned}$$

for a typical value V_{pop} and standard deviation ω_V as described in [Lavielle, 2014].
The typical NMTRAN code for a log-normally distributed parameter is ([Smith and Holford, 2012])

```
GRPV=THETA(1)
V=GRPV*EXP(ETA(1))
```

and in MLXTRAN ([Lavielle and Lixoft Team, 2012])

```
# as explicit equation
eta_V ~ normal(0, omega_V)
V = V_pop*exp(eta_V)

# or using short notation
V = {distribution=lognormal, typical=V_pop, sd=omega_V}
```

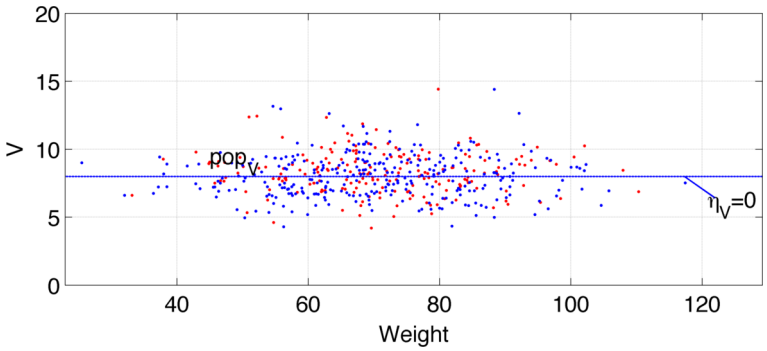


Figure 3.12: Log-normally distributed 'V' with $V_{pop} = 8$ and $\omega_V = 0.2$

3.6.6.2 Log-Normal distributed with a continuous covariate

For a **log-normal** distributed parameter, e.g. V, with body weight, W, as covariate the equivalent representations read

$$(1) \quad \eta_{i,V} \sim \mathcal{N}(0, \omega_V); \quad V_i = V_{pop} \left(\frac{W_i}{70}\right)^{\beta} e^{\eta_{i,V}}$$

$$(2) \quad \eta_{i,V} \sim \mathcal{N}(0, \omega_V); \quad \log(V_i) = \log(V_{pop}) + \beta \log\left(\frac{W_i}{70}\right) + \eta_{i,V}$$

$$(3) \quad \log(V_i) \sim \mathcal{N}\Big(\log(V_{pop}) + \beta \log\Big(\frac{W_i}{70}\Big), \omega_V\Big)$$

The typical NMTRAN code for a log-normally distributed parameter with weight as covariate is

```
GRPV=THETA(1)*(WT/70)**THETA(2)
V=GRPV*EXP(ETA(1))
```

and in MLXTRAN

```
# as explicit equation
V_pop = V_pop*(weight/70)^beta_V
eta_V ~ normal(0, omega_V)
V = V_pop*exp(eta_V)

# or using short notation
V = {distribution=lognormal, typical=V_pop, covariate=lw70, coefficient=beta_V, sd=omega_V}
```

with $lw70 \equiv \log(W/70)$.

3.6.6.3 Logit-Normal distributed

For a **logit-normal** distributed parameter, e.g. $Imax$, the equivalent representations read

$$(1) \quad \eta_{i,Imax} \sim \mathcal{N}(0, \omega); \quad Imax_i = \frac{\left[\frac{Imax_{pop}}{1-Imax_{pop}}\, e^{\eta_{i,Imax}}\right]}{1 + \left[\frac{Imax_{pop}}{1-Imax_{pop}}\, e^{\eta_{i,Imax}}\right]}$$

$$(2) \quad \eta_{i,Imax} \sim \mathcal{N}(0, \omega); \quad \text{logit}(Imax_i) = \text{logit}(Imax_{pop}) + \eta_{i,Imax}$$

$$(3) \quad \text{logit}(Imax_i) \sim \mathcal{N}\big(\text{logit}(Imax_{pop}), \omega\big)$$

Equation (1) can be rewritten using '*logit*' as follows

$$Imax_i = \frac{\exp\left(logit(Imax_{pop}) + \eta_{i,Imax}\right)}{1 + \exp\left(logit(Imax_{pop}) + \eta_{i,Imax}\right)}$$

$$\Leftrightarrow Imax_i = \frac{1}{1 + \exp\left(-logit(Imax_{pop}) - \eta_{i,Imax}\right)}$$

The last form is used for a typical NMTRAN implementation of a logit-normally distributed parameter

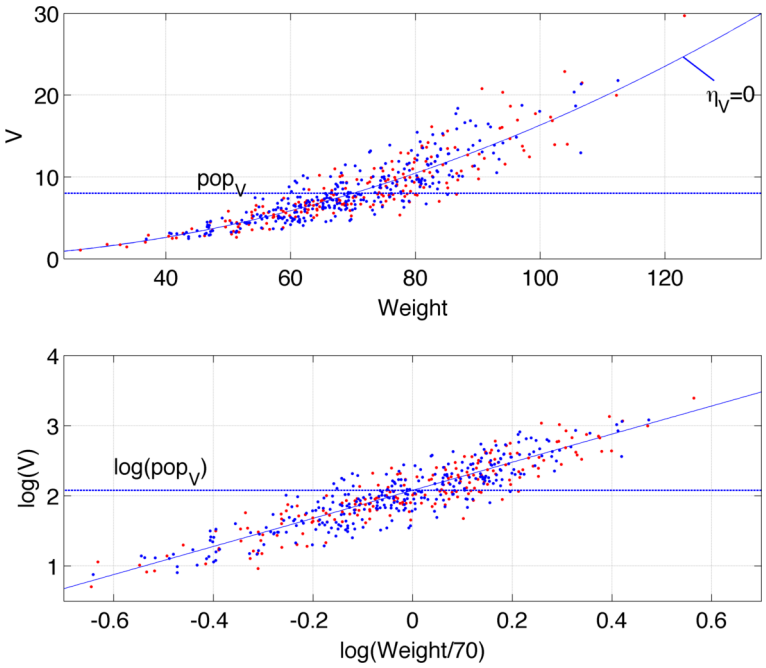


Figure 3.13: Log-normally distributed 'V' with 'Weight' as covariates.

```
LGTIMAX=LOG(POP_IMAX/(1−POP_IMAX)) + ETA(IMAX)
IMAX=1/(1+EXP(−LGTIMAX))
```

and in MLXTRAN

```
# as explicit equation
eta_Imax ~ normal(0, omega_Imax)
logitImaxi = log(pop_Imax/(1−pop_Imax)) + eta_Imax
Imaxi = 1/(1 + exp(−logitImaxi))

# or using short notation
Imax = {distribution=logitnormal, typical=Imax_pop, sd=omega_Imax}
```

3.6.6.4 Logit-Normal distributed with a continuous covariate

For a **logit-normal** distributed parameter with *Weight* as **covariate** we have

$$(1)\quad \eta_{i,Imax} \sim \mathcal{N}(0,\omega); \quad Imax_i = \frac{\left[\frac{Imax_{pop}}{1-Imax_{pop}}\left(\frac{W_i}{70}\right)^{\beta} e^{\eta_{i,Imax}}\right]}{1+\left[\frac{Imax_{pop}}{1-Imax_{pop}}\left(\frac{W_i}{70}\right)^{\beta} e^{\eta_{i,Imax}}\right]}$$

$$(2)\quad \eta_{i,Imax} \sim \mathcal{N}(0,\omega); \quad \text{logit}(Imax_i) = \text{logit}(Imax_{pop}) + \beta \log\left(\frac{W_i}{70}\right) + \eta_{i,Imax}$$

$$(3)\quad \text{logit}(Imax_i) \sim \mathcal{N}\left(\text{logit}(Imax_{pop}) + \beta \log\left(\frac{W_i}{70}\right), \omega\right)$$

The first equation can be rewritten as follows

$$Imax_i = \frac{\exp\left(logit(Imax_{pop}) + \beta \log\left(\frac{W_i}{70}\right) + \eta_{i,Imax}\right)}{1+\exp\left(logit(Imax_{pop}) + \beta \log\left(\frac{W_i}{70}\right) + \eta_{i,Imax}\right)}$$

$$\Leftrightarrow Imax_i = \frac{1}{1+\exp\left(-logit(Imax_{pop}) - \beta \log\left(\frac{W_i}{70}\right) - \eta_{i,Imax}\right)}$$

The last form is used for a typical NMTRAN implementation of a logit-normally distributed parameter with covariate

```
LGTIMAX=LOG(POP_IMAX/(1−POP_IMAX)) + BETA∗LOG(WT/70) + ETA(IMAX)
IMAX=1/(1+EXP(−LGTIMAX))
```

and in MLXTRAN

```
# as explicit equation
eta_Imax ~ normal(0, omega_Imax)
logitImaxi = log(pop_Imax/(1−pop_Imax)) + beta∗lw70 + eta_Imax
Imaxi = 1/(1 + exp(−logitImaxi))

# or using short notation
Imax = {distribution=lognormal, typical=Imax_pop, covariate=lw70, coefficient=beta_Imax, sd=omega_Imax}
```

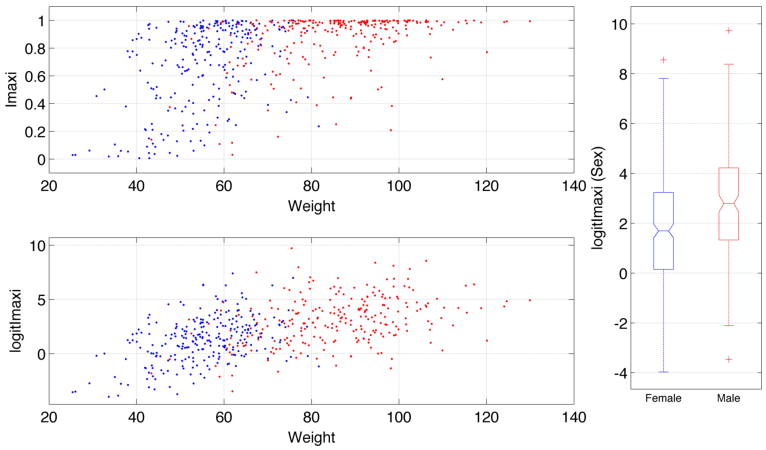


Figure 3.14: Logit-normally distributed 'Imax' with 'Weight' as covariate.

3.6.6.5 Log-Normal distributed with complex variability structure

In this example we consider representations of type (1) and (2) only. A typical parameter model with a continuous covariate, W, for three levels of variability e.g. {centre, subject, occasion} (this will be explained in detail in next section), see Figure 3.9, reads as follows

$$(1) \quad V_{lik} = V_{pop} \left(\frac{W_i}{70}\right)^{\beta} e^{\eta_{l,V}^{(-1)}} e^{\eta_{li,V}^{(0)}} e^{\eta_{lik,V}^{(+1)}}$$

$$(2) \quad \log(V_{lik}) = \log(V_{pop}) + \beta \log\left(\frac{W_i}{70}\right) + \eta_{l,V}^{(-1)} + \eta_{li,V}^{(0)} + \eta_{lik,V}^{(+1)}$$

with

$$\eta_l^{(-1)} \sim \mathcal{N}\left(0, \Omega^{(-1)}\right), \quad \eta_{li}^{(0)} \sim \mathcal{N}\left(0, \Omega^{(0)}\right), \quad \eta_{lik}^{(+1)} \sim \mathcal{N}\left(0, \Omega^{(+1)}\right)$$

with l – centre index, i – subject index, k – occasion index.

3.7 Observation model

Figure 3.15 shows the classification of continuous and discrete data models in PharmML 0.5.1. The defining characteristic of data is their distribution which will be described first

- Continuous data
  - usually described by the normal distribution.
- Discrete data
  - Count data – non-negative integers – described by the Poisson or a related distribution.
  - Categorical data – finite set of integer values, nominal or ordered – described by the Bernoulli or the categorical distribution.

– Time-to-event data – time until an event occurs, can be unique or repeated – described by survival or hazard function.

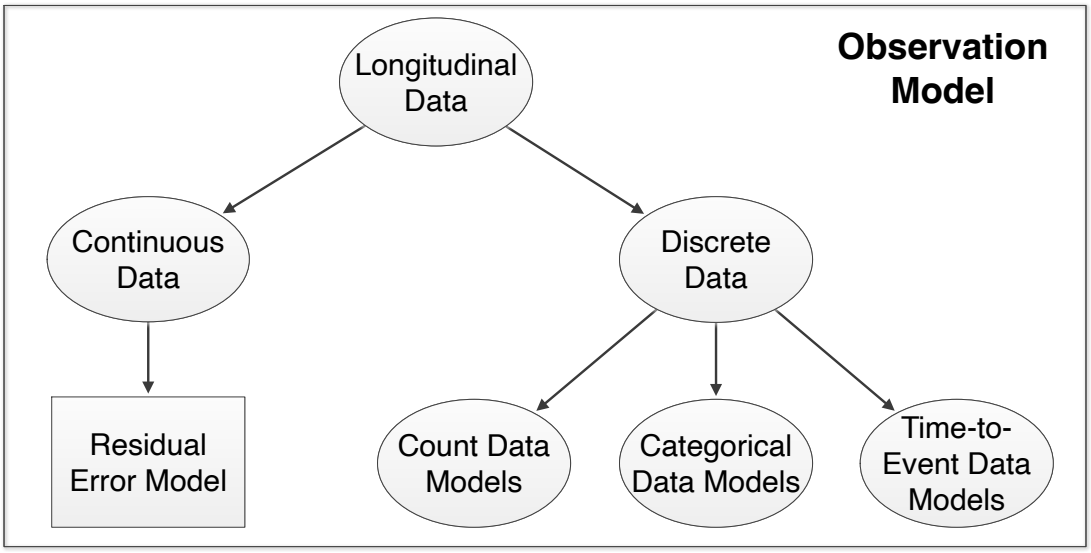


Figure 3.15: Observation model for continuous and discrete data. The residual error model of continuous data is described by a continuous distribution (e.g. normal distribution), while the error distribution of discrete data is described by a discrete distribution (e.g. binomial for binary data).

3.7.1 Contiuous data

An essential component of the continuous observation model is the residual error model.

3.7.1.1 Residual error model

In this section we consider different forms of the residual error, i.e. this section is about g in the term

$$g(x_{ij}, \psi_i, \xi)\epsilon_{ij}$$

of eq.3.1 with $\epsilon_{ij} \sim N(0,1)$, i.e. a standard normally distributed random variable. We distinguish between

- models for **untransformed** data

$$\underbrace{y_{ij}}_{\substack{\text{Experimental}\\ \text{data}}} = \underbrace{f(x_{ij}, \psi_i)}_{\substack{\text{Model}\\ \text{prediction}}} + \underbrace{g(x_{ij}, \psi_i, \xi_i)\ \epsilon_{ij}}_{\substack{\text{Residual}\\ \text{error}}}$$

- **transform-both-sides** models

$$\underbrace{u(y_{ij})}_{\substack{\text{Transformed}\\ \text{experimental}\\ \text{data}}} = \underbrace{u\big(f(x_{ij}, \psi_i)\big)}_{\substack{\text{Transformed}\\ \text{model}\\ \text{prediction}}} + \underbrace{g(x_{ij}, \psi_i, \xi_i)\ \epsilon_{ij}}_{\substack{\text{Residual}\\ \text{error}}}$$

- and **implicit** models

$$\underbrace{u(y_{ij})}_{\substack{\text{Transformed}\\ \text{experimental data}}} = \underbrace{U\big(f(x_{ij}, \psi_i), \xi_i, \epsilon_{1,ij}, \epsilon_{2,ij}, \dots\big)}_{\substack{\text{Transformed}\\ \text{model prediction}}}$$

The *untransformed* form is a special case of the *transform-both-sides* form with $u \equiv Id$, i.e. the identity transformation. Then for models of both types with ϵ_{ij} being normally distributed with mean 0 and variance 1, $u(y_{ij})$ is also normally distributed with mean $u(f(x_{ij}, \psi_i))$ and the standard deviation $g(x_{ij}, \psi_i, \xi_i)$. Possible extensions to the basic models are

- when more than one random variable is applied, i.e. multiple ϵ's,
- when more than one type of measurement or observation is defined, or
- when variability, as discussed in section 3.5, is applied to parameters of the residual error model (see section 3.7.1.2 for details).

3.7.1.2 Incorporating variability on the residual error model parameters

In analogy to the nested hierarchical structure for the variability on the individual parameters, variability on residual error model parameters can be defined using the same structure. By doing so, no new structure is necessary to account for any inter-individual and/or inter-occasion variability of the residual error model parameters.

This allows PharmML to cover the so-called 'ETA-on-EPS' approach – e.g. IIV on the residual error model parameters or in other words varying residual error magnitude between individuals, see Figure 3.16. For example, if an additive residual error model and a log-normal distribution for a is assumed, then the parameter model reads

$$\log(a_i) = \log(a_{pop}) + \eta_a, \quad \eta_a \sim \mathcal{N}(0, \omega_a^2)$$

and the observation model reads

$$y_{ij} \sim \mathcal{N}(f_{ij}, a_i^2): \quad y_{ij} = f_{ij} + a_i \epsilon_{ij}, \quad \epsilon_{ij} \sim \mathcal{N}(0, 1).$$

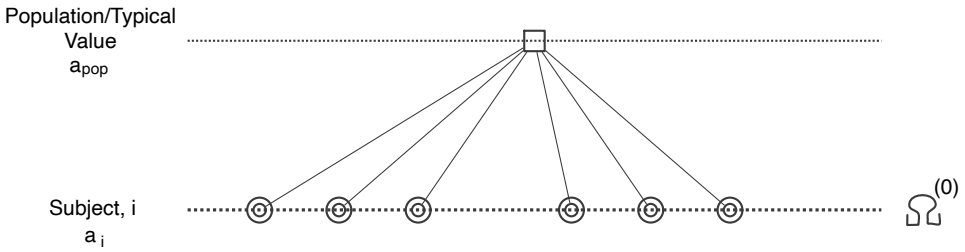


Figure 3.16: Inter-individual variability of the residual error parameter a. The nested hierarchical structure is identical to that of structural model parameters.

3.7.1.3 Residual error model examples

Currently, there is no library of residual error models but this might change in the future. All of the following residual error model examples and their different versions can be implemented in the present version of PharmML:

- Constant/additive:

$$\begin{aligned} y_{ij} &= f_{ij} + a\,\epsilon_{ij}; \quad \epsilon_{ij} \sim N(0,1) \\ \text{or} \quad y_{ij} &= f_{ij} + \epsilon_{ij}; \quad \epsilon_{ij} \sim N(0,\sigma^2) \end{aligned}$$

- Proportional or constant CV (CCV):

$$\begin{aligned} y_{ij} &= f_{ij} + b f_{ij}\,\epsilon_{ij}; \quad \epsilon_{ij} \sim N(0,1) \\ \text{or} \quad y_{ij} &= f_{ij}(1+\epsilon_{ij}); \quad \epsilon_{ij} \sim N(0,\sigma^2) \end{aligned}$$

- Combined additive and proportional 1:

$$y_{ij} = f_{ij} + (a + b f_{ij})\,\epsilon_{ij}; \quad \epsilon_{ij} \sim N(0,1)$$

- Combined additive and proportional 2:

$$\begin{aligned} & y_{ij} = f_{ij} + \sqrt{a^2 + b^2 f_{ij}^2}\, \epsilon_{ij}; \quad \epsilon_{ij} \sim N(0,1) \\ \text{or} \quad & y_{ij} = f_{ij} + a\, \epsilon_{1,ij} + b f_{ij}\, \epsilon_{2,ij}; \quad \epsilon_{1,ij} \sim N(0,1); \quad \epsilon_{2,ij} \sim N(0,1); \\ \text{or} \quad & y_{ij} = f_{ij}(1 + \epsilon_{1,ij}) + \epsilon_{2,ij}; \quad \epsilon_{1,ij} \sim N(0,\sigma_1^2); \quad \epsilon_{2,ij} \sim N(0,\sigma_2^2); \end{aligned}$$

- Power error model:

$$y_{ij} = f_{ij} + b\, f_{ij}^c\, \epsilon_{ij}; \quad \epsilon_{ij} \sim N(0,1)$$

- Combined additive and power error model 1:

$$y_{ij} = f_{ij} + (a + b f_{ij}^c)\, \epsilon_{ij}; \quad \epsilon_{ij} \sim N(0,1)$$

- Combined additive and power error model 2:

$$y_{ij} = f_{ij} + a\epsilon_{1,ij} + b f_{ij}^c \epsilon_{2,ij}; \quad \epsilon_{1,ij} \sim N(0,1); \quad \epsilon_{2,ij} \sim N(0,1)$$

- Two (or more) types of measurements error model:

$$y_{ij} = f_{ij} + \text{ASY}_j \epsilon_{1,ij} + (1 - \text{ASY}_j)\epsilon_{2,ij}; \quad \epsilon_{1,ij} \sim N(0,\sigma_1^2); \quad \epsilon_{2,ij} \sim N(0,\sigma_2^2)$$

- Two (or more) types of observations error model:

$$\begin{aligned} & y_{ij} = \text{TYP}_{ij} f_{1,ij} + (1 - \text{TYP}_{ij}) f_{2,ij} + \text{TYP}_{ij}\epsilon_{1,ij} + (1 - \text{TYP}_{ij})\epsilon_{2,ij}; \\ & \epsilon_{1,ij} \sim N(0,\sigma_1^2); \quad \epsilon_{2,ij} \sim N(0,\sigma_2^2) \end{aligned}$$

Main sources: [Beal et al., 2006] and [Inria POPIX, 2013].

Note 1 In the list above models are pulled together which have the same variance function.

Note 2 Models listed above are the most popular ones in use but the present PharmML structure allows for implementation of virtually any user-defined model. See section ref:XYZ for more examples and PharmML implementation.

3.7.2 Discrete data

The central piece of information to be provided when modeling discrete data is their distribution. While it is possible to encode in PharmML virtually any probability mass or density function explicitly, in few cases we can use the UncertML [UncertML Team, 2014] and its extensive collection of distributions.

The following sections are about the minimal information necessary to be implemented for most common types of discrete data models. It is important to note that we tried to stay generic rather then to provide the support for encoding of models in a particular tool.

3.7.2.1 Count data

An example for count data is the number of events of certain types, such as seizures, heart attacks and is typically modelled by the Poisson distribution. In the case of so called *over-dispersed* data, i.e. when the variability is greater then what one would expect based on typical data sample, we have to resort to more complex models, such as negative binomial or zero-inflated Poisson model, see 3.7.2.1. Typically the minimal information to be provided consists of:

- Type of observed variable – discrete/count
- Count variable, e.g. y
- Parameters (see also Table 3.1)

  - Rate parameter λ, also called the Poisson 'intensity' $\lambda_{ij} = \lambda(t_{ij}, \psi_{ij})$, can be
    * constant (*homogenous Poisson process*)

$$\lambda(t_{ij}, \psi_i) = \lambda_i$$

    * a function of time (*non-homogenous Poisson process*)

$$\lambda(t_{ij}, \psi_i) = \lambda_{0,i} + a_i t_{ij}$$

    * a function of additional regression variables, e.g. drug concentration linked to λ via an Imax-model

$$\lambda(t_{ij}, \psi_i) = \lambda_{0,i}\Big(1 - Imax_i \frac{C_i(t_{ij})}{IC_{50,i} + C_i(t_{ij})}\Big)$$

  - Overdispersion parameter, τ (NB model)
  - Dispersion parameter, δ (GP model)
  - Mixture probability, π ($PMIX_2$ model)
  - Zero probability, p_0 (ZIP model)

- Probability mass function (PMF) with a link function. For example one can define
  - the untransformed PMF, e.g.

$$P(y_{ij} = k; \lambda) = \frac{\lambda^k \exp(-\lambda))}{k!}$$

  - or log(Poisson-PMF)

$$\log(P(y_{ij} = k; \lambda)) = -\lambda + k \log(\lambda) - \log(k!)$$

- Link function – one from the list: {*identity*, *log*, *logit*, *probit*}.
- Markovian dependence, see Sec.3.7.2.4.

The following table gives an overview of common count data models encodable in PharmML.

Short name	Full Model	Distribution parameters	Dependance	Markov order
		equi-dispersed data		
PS	Poisson	λ	–	–
PMAK	Poisson with Markov elements	λ_i	Markov	any order is supported
		over-dispersed data		
NB	Negative Binomial	λ, τ	–	–
ZIP	Zero-Inflated Poisson	λ, p_0	–	–
GP	Generalized Poisson	λ, δ	–	–
$PMIX_2$	Poisson with Mixture Distribution	$\lambda_1, \lambda_2, \pi$	–	–

Table 3.1: Overview of most popular count data models as used in pharmacometrics, based on [Plan et al., 2009]. All of them are encodable in PharmML.

Alternative models Poisson model listed above applies to equi-dispersed data only. There is a number of alternative models which can handle over-dispersed data. Note, that the according PMF or log(PMF) has to be explicitly encoded in PharmML, see [Plan et al., 2009]:

- Zero-inflated Poisson model, ZIP

$$P(y_{ij}=k;\lambda,p_0)=\begin{cases} p_0+(1-p_0)e^{-\lambda} & \text{if} \quad k=0 \\ (1-p_0)\frac{e^{-\lambda}\lambda^k}{k!} & \text{if} \quad k>0 \end{cases}$$

  with $\lambda>0$ and $p_0\in[0,1]$

- Generalised Poisson model, GP

$$P(y_{ij}=k;\lambda,\delta) \quad = \quad \frac{\lambda(\lambda+k\delta)^{k-1}e^{-\lambda-k\delta}}{k!}$$

  with $\lambda>0$ and $\delta\in[0,1]$

- Poisson model with mixture distribution, PMIX

$$P(y_{ij}=k;\pi,\lambda_1,\lambda_2) \quad = \quad \pi\frac{e^{-\lambda_1}\lambda_1^k}{k!}+(1-\pi)\frac{e^{-\lambda_2}\lambda_2^k}{k!}$$

  with $\lambda_1,\lambda_2>0$ and $\pi\in[0,1]$

- Negative Binomial model, NB

$$P(y_{ij}=k;\lambda,\tau) \quad = \quad \frac{\Gamma\left(k+\frac{1}{\tau}\right)}{k!\times\Gamma\left(\frac{1}{\tau}\right)}\times\left(\frac{1}{1+\tau\times\lambda}\right)^{\frac{1}{\tau}}\times\left(\frac{\lambda}{\frac{1}{\tau}+\lambda}\right)^{k}$$

  with $\lambda,\tau>0$.

Note 1 PharmML 0.4 supports the mathematical functions *gammaln* – logarithm of gamma function and *factln* – logarithm of the factorial.

Note 2 Currently only the Poisson distribution is supported by UncertML, meaning that all other PMF's have to be implemented explicitly. The Negative Binomial distribution is featured in UncertML as well but is using a different parametrisation and therefore cannot be used for our purposes.

3.7.2.2 Categorical data

The standard count data models, as presented in the previous section, are simpler to categorise and implement then the categorical ones. This is because for the former the definition of a probability mass function (PMF) and few parameters is sufficient. On the contrary the categorical models come with a vast variety of probability expressions, such as basic nominal and ordered models but also more complex once e.g. adjacent category models, continuation ratio models, latent continuous variable model and others. [Paule et al., 2012] provides an excellent overview of discrete models as used in pharmacodynamics.

Categorical model type deals with data organised in nominal (e.g. presence/absence of an event) or ordered (e.g. pain scale) categories. In the first case the probability for each category is defined. For the second case the cumulative or tail probabilities have to be defined. Furthermore, there are cases when Markovian dependency and initial/conditional probabilities have to be considered.

Typically the minimal information to be provided consists of:

- Nominal categorical data
  - Set of categories, e.g. $\{0,1\}$ or $\{1,2,3\}$
  - Category variable: e.g. Y
  - Probability for each category
    - Binomial distribution, e.g. $P(Y=1)=p$ (for $Y\in\{0,1\}$)

* Probabilities for the k (or $k-1$) categories, here for $Y \in \{1,2,3\}$,

$$P(Y=1) = a1/(a1+a2+a3)$$
$$P(Y=2) = a2/(a1+a2+a3)$$
$$P(Y=3) = 1 - P(Y=1) - P(Y=2)$$

– or alternatively PMF using predefined distributions in UncertML – available are following distributions: bernoulli, binomial, categorical.

– Markovian dependence, see Sec.3.7.2.4

– Link function from the list {*identity*, *log*, *logit*, *probit*}

- Ordered categorical data

– Set of categories, e.g. $\{0,1\}$ or $\{1,2,3\}$

– Category variable: e.g. Y

– Link function – one from the list: {*identity*, *log*, *logit*, *probit*, *loglog*, *comploglog*}.

– Probability for k or $(k-1)$ categories – using one of the link functions. The possible options are

* Exact Cumulative probability – P(Y ≤ i), log(P(Y ≤ i)), logit(P(Y ≤ i)), probit(P(Y ≤ i)), loglog(P(Y ≤ i)), comploglog(P(Y ≤ i))
* Cumulative probability – P(Y<i), log(P(Y < i)), logit(P(Y < i)), probit(P(Y < i)), loglog(P(Y < i)), comploglog(P(Y < i)),
* Exact Tail probability – P(Y ≥ i), log(P(Y ≥ i)), logit(P(Y ≥ i)), probit(P(Y ≥ i)), loglog(P(Y ≥ i)), comploglog(P(Y ≥ i))
* Tail probability – P(Y > i), log(P(Y > i)), logit(P(Y > i)), probit(P(Y > i)), loglog(P(Y > i)), comploglog(P(Y > i))

e.g.

* Cumulative probabilities, e.g.

$$P(Y \leq 1) = a1/(a1+a2+a3)$$
$$P(Y \leq 2) = (a1+a2)/(a1+a2+a3)$$
$$P(Y \leq 3) = 1 - P(Y <= 2)$$

* Tail probabilities, e.g.

$$P(Y > 1) = (a2+a3)/(a1+a2+a3)$$
$$P(Y > 2) = a3/(a1+a2+a3)$$
$$P(Y > 3) = 1 - P(Y > 2)$$

* Cumulative logit probabilities, e.g.

$$\text{logit}(P(Y \leq 1)) = \theta_1$$
$$\text{logit}(P(Y \leq 2)) = \theta_1 + \theta_2$$
$$\text{logit}(P(Y \leq 3)) = 1$$

– Markovian dependence, see Sec.3.7.2.4 for definition and examples.

Complex categorical data The proposed PharmML structure allows to encode other popular models useful in pharmacometrics [Dobson, 2002], for example

- Complementary log-log models

$$\log\{-\log[P(y=j)]\} = \beta_0 + \beta_1 x$$

- Continuation ratio model

$$\text{logit}\Big[\frac{P(y=j)}{\sum_{i=j+1}^{J} P(y=i)}\Big] = b + \sum_{j=1}^{N} \beta_j x_j$$

- Adjacent category logit model

$$\log\left[\frac{P(y=j)}{P(y=j+1)}\right] = \sum_{j=1}^{N} \beta_j x_j$$

- Complex logit functions (based on [Girard et al., 1998])

$$\log\left[\frac{P(n_j = r|n_{j-1} = q)}{1 - \sum_{j\in\{0,2\}} P(n_j = k|n_{j-1} = q)}\right] = p_{rqkq}$$

3.7.2.3 Time-to-event data

In this case the observation is time until an event occurs, which can be unique or repeated. The model is fully described by defining the the survival or hazard function. Following minimal information needs to provided to ensure lossless encoding of this model type.

- Type of observed variable – discrete/time-to-event
- One of the probability distribution defining functions
  - Hazard function, $h(t;\psi_i)$ or
  - Survival function $S(t;\psi_i)$
- Type of censoring
  - Right censoring – right censoring time
  - Interval censoring – interval length
- Maximum number of possible events.

3.7.2.4 Markovian dependence

To define Markovian dependence in any of those models above, one needs to specify:

- State variables as one of the following
  - Initial state variable, *yinit*
  - Current state variable, y
  - Previous state variable, yp
- Probability transitions or their alternative transformed forms for a discrete-time models Markov process, e.g.

$$\text{logit}(P(y <= 1|yp = 1)) = a11$$
$$\text{logit}(P(y <= 2|yp = 1)) = a11 + a12$$

  with y – current state, yp – previous state.

- Transition rates for a continuous-time Markov process, [Lavielle, 2014], e.g.

$$\rho_{1,2}(t) = \exp(a_i + b_i t)$$
$$\rho_{2,1}(t) = \exp(c_i + d_i t)$$

- Probability distribution of the initial states, $yinit1, yinit2, ...$ – 'distribution of first observations'
- Markov order.

Chapter 4

Trial design model

4.1 Introduction

In tools such as NONMEM and MONOLIX it has been common practice to encode the trial design in a data file. More specifically parts of the overall problem, e.g. the structural model and the parameters are explicitly encoded in the model file, while other, design related, parts are encoded in the data file. This implies that the software tool, when processing the data file, must associate the data items with model variables in order to recognise all characteristics of a study, such as subject-specific measurement time points and values, covariates, variability levels, etc. This has clear disadvantages for simulation purposes, because it means that when wanting to change only the design, the data file has to be changed, an error prone and time consuming work. This is clearly not an ideal situation and PharmML addresses this issue.

It is important to stress that we have based a major part of the trial design on one of the standards developed by CDISC "a global, open, multidisciplinary, non-profit organisation that has established standards to support the acquisition, exchange, submission and archive of clinical research data and metadata" [CDISC, 2013]. Over the recent years, this organisation has worked out a set of standards widely used in the medical and pharmaceutical research, both in academic and commercial centres. Using this standard gives us the reassurance that PharmML will be able to represent all trial structures that we are likely to encounter.

4.1.1 Sources of clinical data

Clinical trials are carefully structured and can vary considerably in their complexity. Typically, a trial will have one or more arms with each arm containing one or more treatment regimens and observation protocols. Individuals are then allocated to each arm from a population of subjects who have been screened for their suitability to participate in the trial. In PharmML we describe the structure and population of a trial explicitly in a dedicated section. This differs from some other approaches, but we feel it makes the clinical trial much clearer to document and easier to encode computationally.

The clinical data comes usually from different sources (and formats) and in PharmML we distinguish this information by separating it into the following three classes of data dependent on their origin[1]:

Population The attributes of the individuals in the study: the population in the population model. Each individual has a weight, an age, a gender and numerous other properties that may or may not be modelled as covariates in a given model. Importantly, the 'arm' membership of every subject/patient is part of this information. In addition, these properties may change over time.

Dosing When and how a drug or drugs are administered to the individuals in the trial.

Measurements These are the observations taken from each individual at specific times during the study. Such measurements provide the objective data used during parameter estimation and are typically the outputs calculated during a simulation.

[1] It is interesting to note that the developers of PharML had a similar insight and organised data in a similar way [NLME Consortium, 2008].

4.2 Trial Design

By separating out these classes of information you can see that the information we need to define for a clinical trial is as follows:

Structure The organisation of the trial, how the subjects are grouped into different treatment groups and what the dosing regimen is within these treatment groups.

Population As above, the properties specific to the individuals, including those that vary over time.

Individual Dosing This is related to the treatment regimens described in the trial structure, but describes the dosing history for each individual in the study.

The measurement data is then used exclusively for estimation and is encoded in the third major building block of PharmML, in the 'Modelling Steps'.

4.2.1 Structure

To define the Trial Structure we have reused, almost verbatim, the CDISC Study Design Model [CDISC consortium, 2011], which is an XML representation of a clinical trial. Figure 4.1 below shows how the CDISC trial structure is organised. It has six main components:

Epoch The epoch defines a period of time during the study which has a purpose within the study. For example a washout or a treatment window. In CDISC Epochs can describe screening or follow-up periods, which are out of the scope of PharmML. An epoch is usually defined by a time period.

Arm The arm represents a path through the study taken by a subject. An arm is composed of a study cell for each epoch in the study.

Cell The study cell describes what is carried out during an epoch in a particular arm. There is only one cell per epoch.

Segment The segment describes a set of planned observations and interventions, which may or may not involve treatment. Note that in PharmML our definition is more limited and we only describe treatments. A segment can contain one or more activities.

Activity The activity is an action that is taken in the study. Here it is typically a treatment regimen or a washout.

StudyEvent A study event describes the collection of information about a particular individual. In CDISC this can be information captured during screening or other non-treatment phases of the clinical trial. But here we restrict it to capturing observations during the treatment. In PharmML this is how we capture occasions.

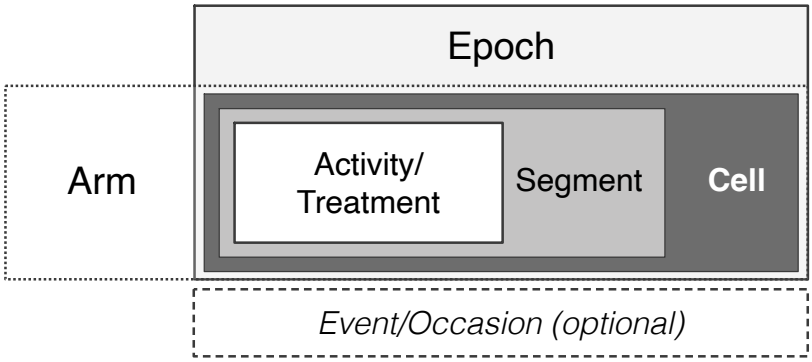


Figure 4.1: Overview of the basic concept of the Trial Structure: arm, epoch, event and a cell with segment and activity/treatment as used in the CDISC Study Design Model. See the next figure for an example.

Figure 4.2 shows one such example of a hypothetical trial consisting of two arms and three epochs: *Screen*, *Treatment* and *Follow Up*. There are accordingly six cells and segments, each consisting of one or two

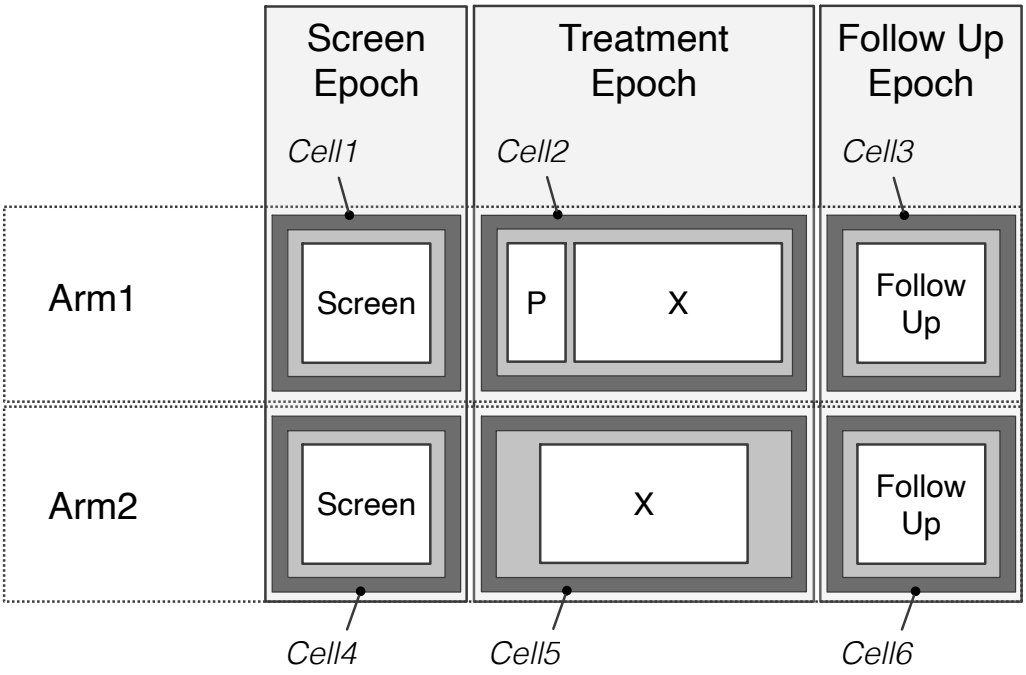


Figure 4.2: An example of a study with two arms and three epochs: Screen, Treatment and Follow Up. A segment can contain more then one Activity/Treatment as can be seen in *Cell2* with one Pre-Treatment P and one Treatment X. In this particular example no Event/Occasion is specified.

activities. *Cell2* has the most complex structure carrying two subsequent treatments, Pre-Treatment P and Treatment X. In this case no Events/Occasions are specified which is an optional element in the design.

Examples of how a trial design is encoded in PharmML can be found in the examples (see chapter 9).

Variability There is one aspect regarding the random variability worth mentioning here. The `<Population>` block carries the information about subject level variability and those variability levels above the subject, see next section for more details. In the `<Structure>` element we encode the variability which is located below the subject. This is typically known as *inter-occasion variability* but deeper levels are allowed in theory. The reader is referred to the section 3.5 where the full nested hierarchy of the random variability discussed in detail.

4.2.2 Population

This is the second major element of the trial design description where we:

- describe the individuals in the study
- describe their attributes (such as weight, gender, etc.)
- assign them to an arm of the study, but also
- indicate if variability at and/or below the subject level is to be defined, which is the case in the majority of models (if omitted then this means that we explicitly consider a setup without any random variability, which is the case for the naïve pooled data method), and
- indicate their country or centre membership to define higher levels of variability above the subject level.

We define the possible attributes of all individuals using the *IndividualTemplate* block and then map each individual to this template using a *Dataset* block.

4.2.3 Individual dosing

The two previous sections on *Structure* and *Population* described information, which is sufficient to encode e.g. a simple simulation task. Specifically, when the dosing is equal among the patients then this can be encoded in the *Structure* part of the schema with one or more dose amounts and one or more dosing times for all. However, in most cases, especially when we deal with real clinical data this is not so straighforward. Every patient will have its own specific amounts and dosing times.

For an estimation task we always need to provide experimental data for each dosing activity relevant to the particular case. With the current structure we can provide individual dosing information for every dosing activity defined in the *Structure* part (see 4.2.1).

The structure of this part is similar to that used for *Population* in that first a table template is defined with all relevant columns, i.e. *ID*, *TIME*, and *DOSE*, which is then populated with individual dosing data.

Chapter 5

PK Macros

5.1 Introduction

A long standing problem in Pharmacometrics is how to define effectively PK models, without using ODEs, in a tool-independent way. There are many tool-specific solutions, each of them using their own ways to define PK models. After a thorough analysis of a number of available approaches, the MLXTRAN PK macro system has been selected to be supported in PharmML [Lavielle and Lixoft Team, 2014]. It has the following features

- Tool-independent, but easily translatable to tool-specific libraries
- User friendly and easy to learn for users new to the field
- Flexible and providing consistent specification of compartmental PK models
- More models can be specified without ODEs, compared to nmadvan/PREDPP
- No limitations will be imposed, compared to nmadvan/PREDPP:
  - All PREDPP library models with underlying analytical solutions (ADVAN1-4, 11-12) can be specified. Templates are provided for these to help NONMEM users, see Section 5.3.
  - The PREDPP library models applying matrix exponential-based solutions (ADVAN 5, 7) are already being defined in a way very similar to the macro-based solution proposed.
  - The PREDPP library models applying numerical ODE solution (ADVAN 6, 8-9, 10, 13) can still be specified by using ODEs explicitly (or by using/mixing with macro-definitions).

Adopting the PK macros should allow seamless translation of models between MDL and PharmML and MLXTRAN. Therefore, in the PharmML implementation of the macros we tried to stay as close as possible to the original MLXTRAN specification. We had to make sure that the macros, their attributes and possible assignments are defined in a unambiguous way. In the sections 5.2.2 and 5.2.3 we explain the rules to be followed in their implementation and the translation process to and from PharmML.

5.1.1 The macro idea

Figure 5.1 visualises the principle of the PK macros solution and its use. The module within the inner MDL/PharmML box shows few example PK macros and their attributes using which a PK model can be formulated. In this case four macros, with one or more arguments, are used to encode a basic *oral 1-compartment model with linear elimination* consisting of the following items

1. depot compartment, with amount Ad,
2. central compartment, with amount Ac and volume V,
3. oral administration, 1^{st} order absorption with rate constant ka, and
4. linear elimination, with rate constant k.

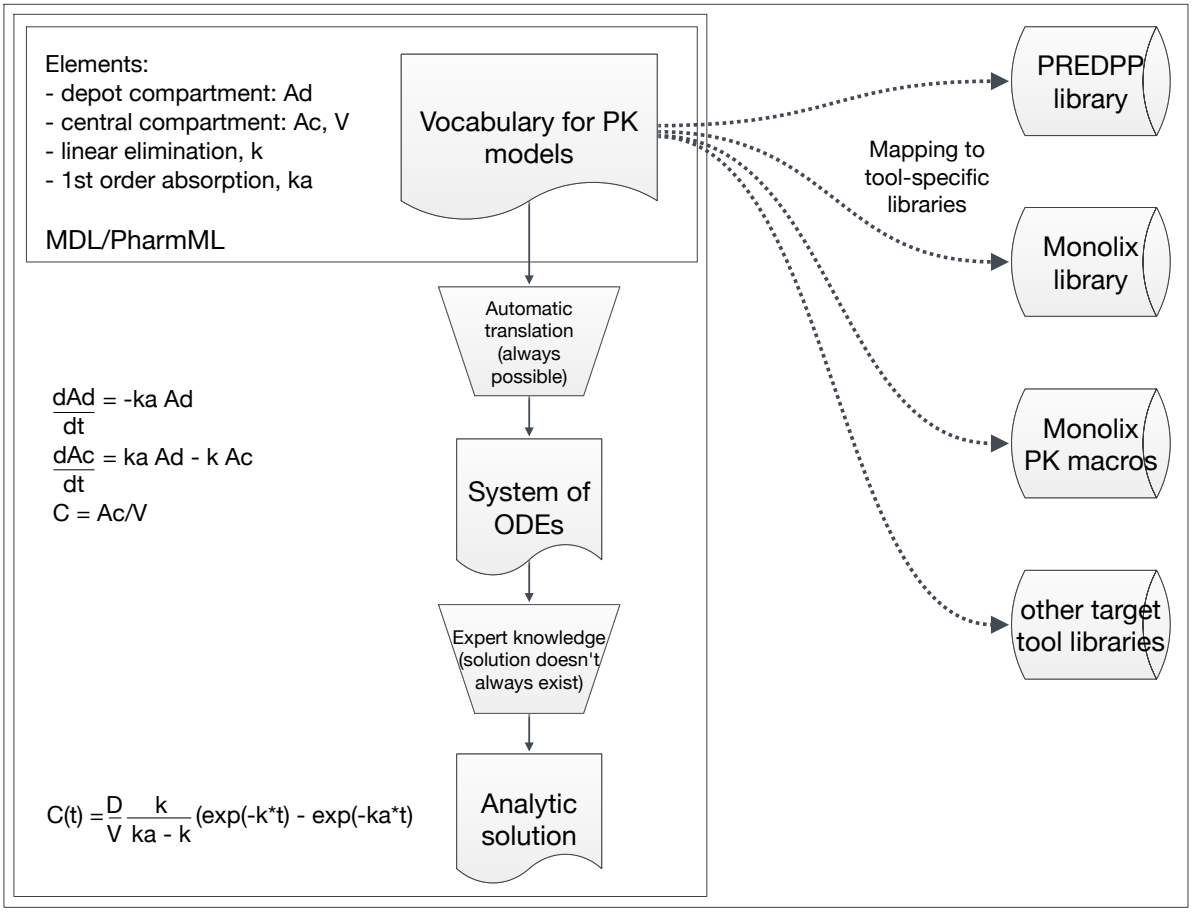


Figure 5.1: A diagram explaining how the PK macro system can be used in practise. The inner MDL/PharmML box contains the information to be stored in PharmML, i.e. the macros and arguments from the PK model vocabulary. It can then be automatically translated into a set of ODEs and for some models there exists an analytic solution. Models encoded using the controlled vocabulary can be mapped to tool specific implementations (dotted lines).

This is the *only* information that needs to be encoded in PharmML for this particular PK model. It corresponds to the following macro

```
compartment(cmt=1,concentration=Cc,volume=V)
oral(cmt=1, ka)
elimination(cmt=1, k)
```

Any information about input, i.e. dosing times and amounts will be stored in the dataset or explicitly within the `<TrialDesign>`.

The key realisation is that every set of such macro statements, if correctly defined, can be translated automatically into a unique set of ODE's and/or algebraic equations, in this case:

$$\frac{dAd}{dt} = -ka \times Ad$$
$$\frac{dAc}{dt} = ka \times Ad - k \times Ac$$
$$C = Ac/V$$

Moreover, for some models such as the above one, there exists an analytic solution. This however cannot be derived automatically for an arbitrary ODE system. It requires a powerful symbolic calculation software or expert knowledge. Once the PK model macros and their attribute set have been defined, a mapping to tool-specific libraries can be provided, see Fig.5.1. This again requires expert knowledge of each particular tool.

5.2 PK macros in PharmML

5.2.1 MLXTRAN macros

PK macros offer an equation-free encoding solution for pharmacokinetic models. The novel system allows to implement in MLXTRAN virtually any compartmental model within well-defined constraints. See [Lavielle and Lixoft Team, 2014] for a detailed description. The following table lists the available PK macros with their arguments.

Component	Macro	Arguments
Compartment	*compartment*	cmt, amount, volume, concentration
Peripheral compartment	*peripheral*	**kij**, **k_i_j**, amount, volume, concentration
Effect compartment	*effect*	cmt, **ke0**, concentration
Absorption from a depot	*depot*	type/adm, **Tlag**, **p**, target, **Tk0**, **ka**, **Ktr**/**Mtt**
Absorption for IV dose	*iv*	cmt, type/adm, **Tlag**, **p**
0-order absorption	*absorption* or *oral*	cmt, type/adm, **Tlag**, **p**, **Tk0**
1st order absorption	*absorption* or *oral*	cmt, type/adm, **Tlag**, **p**, **ka**, **Ktr**/**Mtt**
Linear elimination	*elimination*	cmt, volume, **k**, **CL**
MM elimination	*elimination*	cmt, **Km**, **Vm**
Transfer	*transfer*	from, to, **kt**

Table 5.1: PK macros and their arguments as used in MLXTRAN, based on [Lavielle and Lixoft Team, 2014]. Note that there are more components listed, 10, than macros, 9, actually defined in the MLXTRAN system. Different absorption and elimination types are listed separately for better readability. The arguments can be devided in two groups. Those who can act alone in the macro are marked with bold font, all others need to be assigned a numerical value or a symbol reference, see explanation in text.

One can distinguish two types of arguments, see Table 5.1. The arguments marked with **bold** font, can be simply references to identifiers defined somewhere else in the model file such as *ka* or *Tlag*. The remaining arguments cannot be left unassigned and have to be explicitly specified as attributes of the `<Value>` element and subsequently assigned a numerical value or symbol reference, e.g. *cmt* or *amount*. See Section 5.2.2 for the detailed explanation.

5.2.2 Encoding rules

To provide the full support for MLXTRAN PK macros, all macros and their (unordered) named arguments have to be encodable, see Table 5.1. Every macro has its own PharmML element, e.g. the basic *concentration* macro with all its arguments, encoded as `<Value>` elements, will be encoded within the `<Concentration>` element. The optional attribute `argument` is assigned a value corresponding to the argument of the macro.

As mentioned above, there are two types of arguments, and the following rules are defined to achieve a consistent macro definition and its interpretation:

- **bold** arguments (*can act alone*) can be references to symbol identifiers defined in other places in the model, such as `k`, `ka` etc.
  - cannot be assigned numerical values or expressions within the macro – all such assignments must be defined outside the `<PKmacros>` element, for example in the `<ParameterModel>` or in the `<ModellingSteps>` section
  - if an argument, e.g. 'k', is specified in a macro only, as shown below

    elimination(cmt=1, k)

    then a simple `SymbIdRef` is sufficient

```
<Elimination>
    <!-- cmt argument omitted -->
    <Value>
        <ct:SymbRef blkIdRef="pm1" symbIdRef="k"/>
    </Value>
</Elimination>
```

but it must be defined in a parameter model, here `pm1`.

– if a symbol identifier, here 'k1', is assigned to the key-word argument `k`, e.g.

elimination(cmt=1, k=k1)

then both the keyword as argument, `k`, and the `symbIdRef`, `k1` are required

```
<Elimination>
    <!-- cmt argument omitted -->
    <Value argument="k">
        <ct:SymbRef blkIdRef="pm1" symbIdRef="k1"/>
    </Value>
</Elimination>
```

- all *other* arguments (which *cannot act alone*), e.g. 'cmt' or 'amount' as in the following macro

compartment(cmt=1, amount=Ac)

have to be specified as attributes first and subsequently assigned a numerical value or a symbol reference using element `SymbRef` as in the following code snippet

```
<Compartment>
    <Value argument="cmt">
        <ct:Int>1</ct:Int>
    </Value>
    <Value argument="amount">
        <ct:SymbRef symbIdRef="Ac"/>
    </Value>
</Compartment>
```

5.2.3 Fixed argument values and exceptions

All but one macro have a fixed set of arguments, e.g. the `compartment` can take only these predefined four arguments: {`cmt, amount, volume, concentration`}. The exception is the `peripheral` macro with three fixed arguments {`amount, volume, concentration`} and a variable one, `k_ij or k_i_j`, where `i` and `j` stand for input and output compartment, respectively. These compartments numbers are set according to the model configuration.

For example, in the case of models ADVAN11 or 12, see section 5.4.2 for a complete description of the latter, we need to set the `kij` argument names accordingly to the model specification, meaning the macro

peripheral(k12, k21, amount=Ap1)

will be implemented as shown in the following snippet

```
<Peripheral>
    <Value>
        <ct:SymbRef blkIdRef="pm1" symbIdRef="k12"/>
    </Value>
    <Value>
        <ct:SymbRef blkIdRef="pm1" symbIdRef="k21"/>
    </Value>
    <!-- omitted amount argument -->
</Peripheral>
```

The `k_ij or k_i_j` attributes are a special case in that an assignment in the macro formulated as

peripheral(k12, k21 = k_test, amount=Ap1)

will be translated to PharmML as

```
<Peripheral>
    <Value>
        <ct:SymbRef blkIdRef="pm1" symbIdRef="k12"/>
    </Value>
    <Value argument="k21">
        <ct:SymbRef blkIdRef="pm1" symbIdRef="k_test"/>
    </Value>
    <!-- omitted amount argument -->
</Peripheral>
```

Note the difference between the implementation of the two first arguments. The first argument, `k12`, is implemented as before because no assignment is required. In the second case we use the `argument` attribute to inform the target tool about the meaning of the assigned parameter. Therefore, the symbol `k12` is not a parameter itself and `k_test` serves as the transfer parameter `kij` between compartment 1 and 2, which can be defined in the parameter model `pm1` and can be assigned any value in the `<ModellingSteps>` section.

5.2.4 Macros and target tools

One of the main reasons to introduce the system of macros is that they offer a flexible and efficient way of encoding PK models. The proposed macros are fully compatible with Monolix and its data format, see Appendix B in [Lixoft, 2014b].

Of course, from the perspective of the interoperability platform it is crucial that the models stored in PharmML using macros can be unambiguously translated to NONMEM and other target tools. Especially the compatibility with the predefined PREDPP routines (i.e. ADVAN 1-4 and 10-12) is essential because of their popularity among the NONMEM modellers community.

To understand the required translation rules, the knowledge of relevant NONMEM and MONOLIX features and the dataset formats these target tools use is helpful. For convenience, the most essential differences between these tools/formats from the perspective of the macros and their usage are listed here

- In PREDPP (but only when using ADVAN 1-4, 10-12 routines) the *DEPOT* compartment is always considered as the 1^{st} compartment, the central compartment as 2^{nd} and the peripheral compartments as 3^{rd}, 4^{th} etc.
- In contrast, MONOLIX doesn't assign a number to the *DEPOT* compartment at all and the numbering starts usually with the central compartment.
- The CMT column, standard in NONMEM dataset, is not used in Monolix. It uses an ADM column instead.

5.2.4.1 An illustrative example

Figure 5.2 and datasets in Table 5.2 give us some clues about two alternative interpretations for a model with at least one oral administration, dependent on the target tool. Note, that apart from a few predefined models, such as ADVAN1-4 and 10-12, one can assign an arbitrary number to the (first) depot compartment in the NONMEM dataset. This is exactly what happens in this example. While in MONOLIX we would have only two compartments with assigned number to them, all four compartments have numbers assigned in NONMEM, Figure 5.2 (right). This means that when translating a model from PharmML, a transformation of the associated dataset is required.

It is therefore possible to translate a macro and dataset, utilising the ADM column to identify administrations, used by Monolix into an PREDPP/ADVAN based NONMEM encoding system with data set using the CMT column, see Figure 5.2 and Table 5.2.

MONOLIX				
ID	TIME	AMT	**ADM**	Y
1	6	10	**1**	.
1	9	20	**2**	.
1	18	10	**1**	.
1	33	20	**2**	.
...	...	...	...	...
1	0	.	.	0
1	12	.	.	1.18
...	...	...	...	...

NONMEM				
ID	TIME	AMT	**CMT**	DV
1	6	10	**3**	.
1	9	20	**4**	.
1	18	10	**3**	.
1	33	20	**4**	.
...	...	...	...	...
1	0	.	1	0
1	12	.	1	1.18
...	...	...	...	...

Table 5.2: MONOLIX and NONMEM datasets for the model above, Figure 5.2. The translator has to make sure that the numbers in the ADM column of the former are properly converted to the CMT column in the latter dataset.

5.2.4.2 Basic translation rules

The following few basic translation rules should be treated merely as a guidance because they do not cover the whole aspect of the model/data handling within the interoperability platform but are limited to the

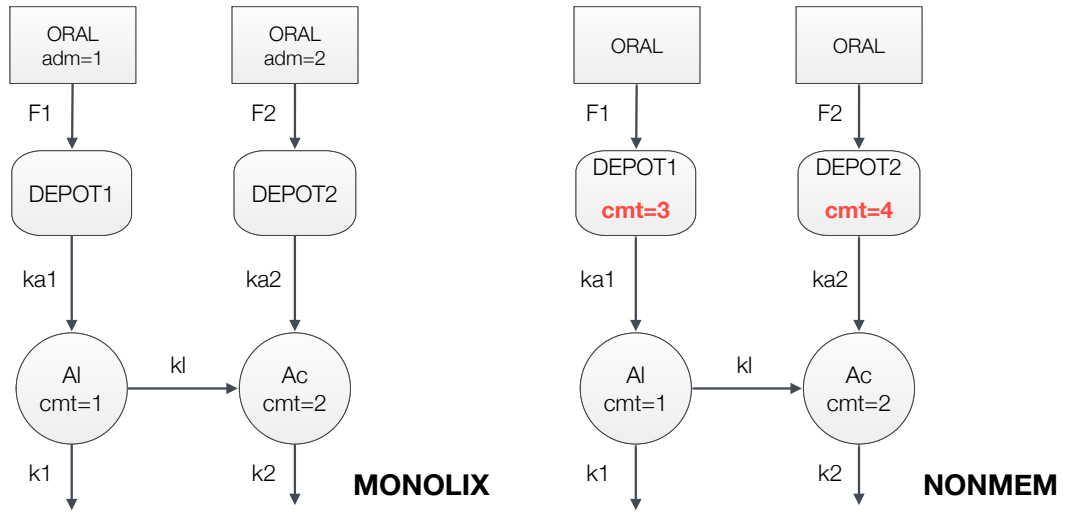


Figure 5.2: Two interpretations for a model with two oral administrations. While MONOLIX (left) doesn't assign a number to the depot compartments, NONMEM (right) does. The translation of macro coded models to target tools requires in this case changes in the dataset as well, see Table 5.2.

models encoded with macros. A more careful analysis is required to cover the structural model exchange between the key target tools and associated datasets.

A – Models corresponding to ADVAN 1-4 and 10-12 Fortunately, the translation of models encoded with macros and having their equivalents in routines ADVAN1-4 and 10-12 in PREDPP and their datasets is straightforward. Simply, one has to translate a corresponding macro into its equivalent ADVAN routine and then to rename the ADM column in the MONOLIX data set into CMT, see for more details and examples in Section 5.3 and 5.4.

B – All other models For every oral administration one new depot compartment needs to be introduced and assigned a number. The Figure 5.2 and datasets in Table 5.2 can again help to understand what is required.

In this example, `adm=1` and `adm=2` denote two oral administrations. In the NONMEM system, Figure 5.2 (right), '1' and '2' are assigned to the central and peripheral compartment respectively, as in the MONOLIX case, so that the next free numbers, '3' and '4', will be assigned to the depot compartments.

Next, the NONMEM dataset needs to be produced based on the information stored in the MONOLIX dataset in such a way that it will match the new compartment numbering scheme with the CMT column replacing the ADM column.

The oral administration, `adm=1`, aims now at the compartment 3, so that '1' in the ADM column will correspond to '3' in the CMT column. Similarly, `adm=2`, aims now at the compartment 4, so that '2' in the ADM column will be translated to '4' in the CMT column.

As additional rules, applicable for all models, one should keep in mind that (1) Y column in a MONOLIX dataset has to be renamed into DV of the NONMEM dataset and (2) the TINF column has to be renamed into RATE and its content recalculated accordingly.

5.2.4.3 Datasets mapping

We will discuss now briefly the mapping between macros and the MONOLIX dataset, i.e. the mapping between `adm/type` attributes in the PK macros and the administration type as stored in the ADM column,

- `<TargetMapping>` element with the `blkIdRef` attribute, the latter one because in PharmML multiple structural models are allowed, so we have to specify which model is holding the target compartment definition
- `<Map>` element with
  - `dataSymbol` – attribute denoting the target symbol as encoded in a dataset and

– new attribute `admNumber` – to identify the target symbol in the model.

The use of these new elements is explained in the Table 5.3.

MONOLIX datasets mapping

```
<ExternalDataSet toolName="Monolix" oid="MLXoid">
    <!-- omitted details -->
    <ColumnMapping>
        <ds:ColumnRef columnIdRef="ADM"/>
        <ds:TargetMapping blkIdRef="sm3">
            <ds:Map dataSymbol="1" admNumber="1"/>
            <ds:Map dataSymbol="2" admNumber="2"/>
            <ds:Map dataSymbol="3" admNumber="3"/>
        </ds:TargetMapping>
    </ColumnMapping>
    <!-- omitted ColumnMappings and Definition of dataset with ID,TIME,AMT,ADM -->
```

Table 5.3: Mappings of MONOLIX type datasets and PK macros.

5.2.5 Linking macros and <TrialDesign>

`<TrialDesign>` section provides an alternative to encode design and to store data records (observations, dosing and covariates) in a tool-independent manner. The `<IndividualDosing>` element is the place where relevant dosing data can be encoded inline or be referred to from an external datafile. The actual mapping is defined within the `<Activity>` tag.

A minor extension in the `<TrialDesign>` is necessary to link the administrations coded in `<Activity>` elements to the targets in the PK macros and identified using the `adm` attribute, i.e.

- new value *administrationType* is added to the `inputTarget` attribute.

Otherwise, elements defined in the last section will be reused. The Table 5.4 shows the implementations within the `<TrialDesign>` element in cases when (left) a PK model is expressed using algebraic equation with dose parameter, D, and (right) when using PK macros.

Using dose `parameter`, D, and algebraic equation for PK model, e.g. $C(t) = f(D, V, k, ...)$ (available in PharmML since version 0.2.1)

```
<Activity oid="actBolusD">
    <Bolus>
        <DoseAmount inputTarget="parameter">
            <ct:SymbRef blkIdRef="sm3" symbIdRef="D"/>
            <ct:Assign>
                <ct:Real>10</ct:Real>
            </ct:Assign>
        </DoseAmount>
        <DosingTimes>
            <!-- e.g. 10 -->
        </DosingTimes>
    </Bolus>
</Activity>
```

Using value *administrationType* for `inputTarget` and `<TargetMapping>`/`<Map>` elements

```
<Activity oid="actBolusMacro">
    <Bolus>
        <DoseAmount inputTarget="admType">
            <ds:TargetMapping blkIdRef="sm3">
                <ds:Map admNumber="2"/>
            </ds:TargetMapping>
            <ct:Assign>
                <ct:Real>10</ct:Real>
            </ct:Assign>
        </DoseAmount>
        <!-- omitted DosingTimes -->
    </Bolus>
</Activity>
```

Table 5.4: Comparison of the link between administration definitions and structural model. (left) PK model expressed using algebraic equations with dose parameter, D, and when using the PK macros (right).

5.2.6 Connection between macros and the model

The possible outputs of PK macros are amounts, e.g. Ac, and concentrations, e.g. C. A PK model defined by a system of macros will often be connected to a subsequent PD model, which expects the concentration, C, as one of the inputs. Another option is that the output of the PK macros will be mapped directly to the data in the `<ObservationModel>`.

For the *compartment* macro in question, there are two possibilities, either

- the concentration is defined in the macro

  compartment(cmt=1,concentration=C, volume=V)

  in which case no output has to be defined explicitly, or

- the alternative form of this macro is used

  compartment(cmt=1,amount=Ac, volume=V)

  and because eventually the concentration is required, a subsequent assignment for C must be provided in MLXTRAN in the *EQUATION* block, i.e. $C = Ac/V$

In PharmML, these two cases have to be treated accordingly.

- In the first case only the concentration variable, C, needs to be defined

```
<StructuralModel blkId="sm1">
    <ct:Variable symbolType="real" symbId="C"/>
    <PKmacros>
           <!-- omitted details of the macro compartment(cmt=1,concentration=C, volume=V) -->
```

- in the second case, both the amount variable, Ac, and concentration variable, C, with the $C = Ac/V$ assignment needs to be defined

```
<StructuralModel blkId="sm1">
    <ct:Variable symbolType="real" symbId="Ac"/>
    <ct:Variable symbolType="real" symbId="Cc">
        <ct:Assign>
            <math:Equation>
                <math:Binop op="divide">
                    <ct:SymbRef symbIdRef="Ac"/>
                    <ct:SymbRef blkIdRef="pm1" symbIdRef="V"/>
                </math:Binop>
            </math:Equation>
        </ct:Assign>
    </ct:Variable>

    <PKmacros>
        <Compartment>
            <!-- omitted details of the macro compartment(cmt=1,amount=Ac, volume=V) -->
```

If a model using the PK macros is defined properly, following the principles described above, both options make sure that the connection to a subsequent model or the `<ObservationModel>` can be established.

5.3 PREDPP models and PK macros

This section describes PREDPP library models as encoded in routines ADVAN1-4 and 10-12 and their PK macros implementation. As explained in Section 5.2.4 there are differences between NONMEM and MONOLIX and their datasets one has to consider when dealing with PK macros. They manifest themselves also in that way how the transfer rate constants are numbered/named. Models with more then two compartments and with 1st order input (using *DEPOT* compartment) will have different transfer rate symbols, e.g.:

- ADVAN 4

	PREDPP routines	PK macros
transfer rate constants	k23, k32	k12, k21

- ADVAN 12

	PREDPP routines	PK macros
transfer rate constants	k23, k32	k12, k21
	k24, k42	k13, k31

Models described in this section use the MLXTRAN numbering convention in order to comply with the PK macros notation.

To allow for a lossless translation between PREDPP coded models and PK macros one needs to establish well defined translation rules. To achieve this goal and provide help to other translation/converter teams one needs to analyse the predefined PREDPP routines ADVAN1-4 & 10-12 and more specifically to

1. understand what they mean
2. dissect the routines into elementary components and finally
3. find equivalent formulation for each such element in the PK macro speak.

The above steps are illustrated in Table 5.5 in an abbreviated form. In the column *Interpretation* first two steps are implemented, the last column provides the according PK macro formulation.

ADVAN routine & default TRANS1	Interpretation	MLXTRAN macro
	one compartment	
ADVAN1 compartment	– 1 compartment	`compartment(cmt=1, amount=Ac, volume=V)`
	– iv bolus administration	`iv(adm=1, cmt=1)`
	– linear elimination	`elimination(cmt=1, k)`
ADVAN2	– 1 compartment	`compartment(cmt=1, amount=Ac, volume=V)`
	– oral administration, 1st order absorption	`oral(adm=1, cmt=1, ka)`
	– linear elimination	`elimination(cmt=1, k)`
ADVAN10	– 1 compartment	`compartment(cmt=1, amount=Ac, volume=V)`
	– iv bolus administration	`iv(adm=1, cmt=1)`
	– saturable elimination	`elimination(cmt=1, Km, Vm)`
	two compartments	
ADVAN3	– 2 compartments one central (1) & one peripheral with linear transfer rates	`compartment(cmt=1, amount=Ac, volume=V)` `peripheral(k12, k21, amount=Ap)`
	– iv bolus administration (into 1)	`iv(adm=1, cmt=1)`
	– linear elimination (from 1)	`elimination(cmt=1, k)`
ADVAN4	– 2 compartments one central (1) & one peripheral with linear transfer rates	`compartment(cmt=1, amount=Ac, volume=V)` `peripheral(k12, k21, amount=Ap)`
	– oral administration, 1st order absorption (into 1)	`oral(adm=1, cmt=1, ka)`
	– linear elimination (from 1)	`elimination(cmt=1, k)`
	three compartments	
ADVAN11	– 3 compartments one central (1) & two peripheral with linear transfer rates	`compartment(cmt=1, amount=Ac, volume=V)` `peripheral(k12, k21, amount=Ap1)` `peripheral(k13, k31, amount=Ap2)`
	– iv bolus administration (into 1)	`iv(adm=1, cmt=1)`
	– linear elimination (from 1)	`elimination(cmt=1, k)`

ADVAN12	– 3 compartments
	one central (1) & two peripheral with linear transfer rates
		`compartment(cmt=1, amount=Ac, volume=V)`
		`peripheral(k12, k21, amount=Ap1)`
		`peripheral(k13, k31, amount=Ap2)`

	– oral administration, 1st order absorption (into 1)	`oral(adm=1, cmt=1, ka)`
	– linear elimination (from 1)	`elimination(cmt=1, k)`

Table 5.5: Interpretation and translation of PREDPP models using ADVAN1-4 & 10-12 routines, with default TRANS1 parameterization, into PK macros.

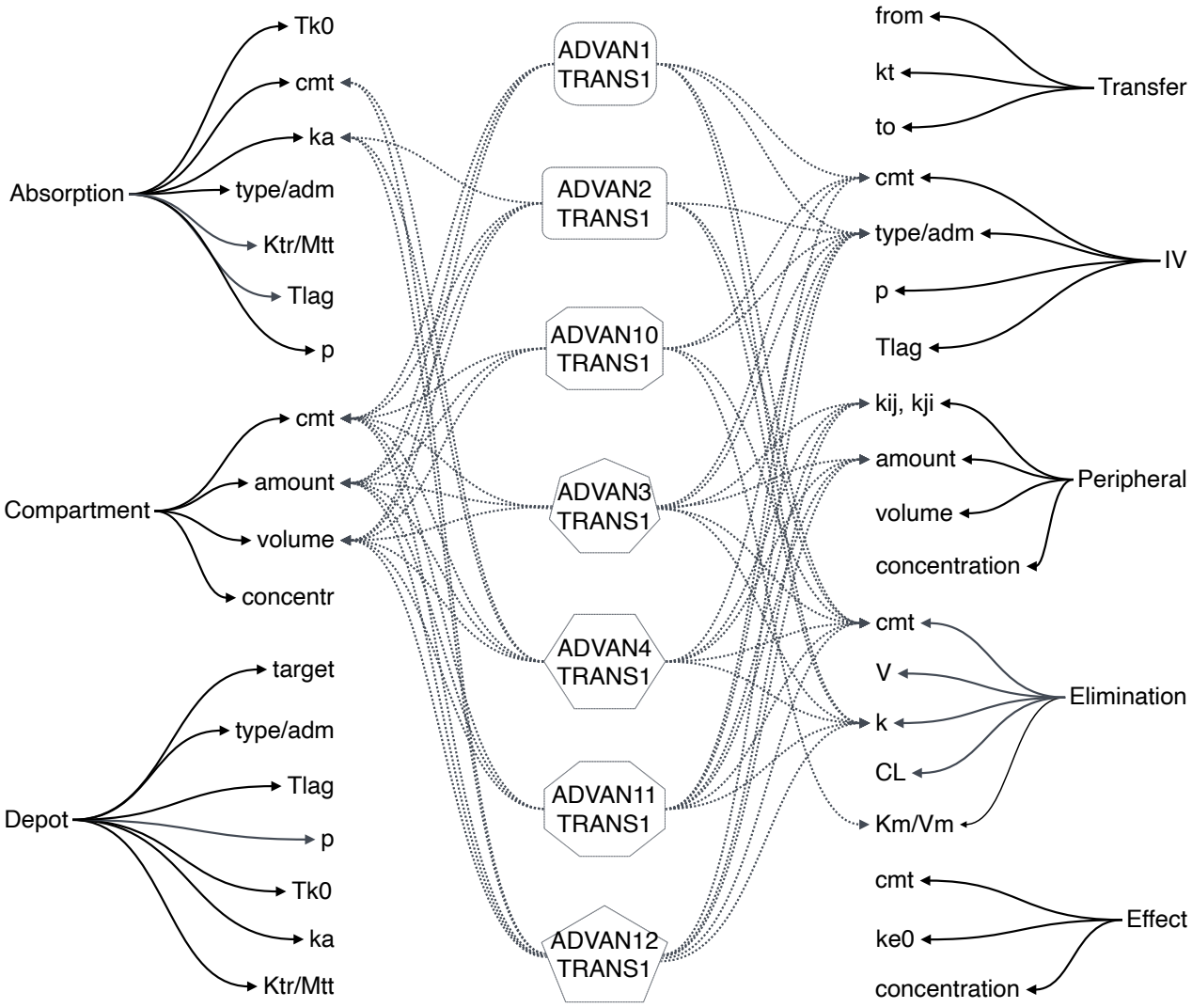


Figure 5.3: The overview of the ADVAN models, with their default parameterisation TRANS1, and their connection to PK macros based on the analysis in Table 5.5. Each ADVAN model can be uniquely mapped to macros and their attributes. For more details see next section with examples.

5.4 Examples

We will conclude this section with two examples of PREDPP coded models and show their implementation in PK macros in PharmML, which will illustrate number of aspects described in the previous sections such as interpretation of the models, their translation and the associated dataset conversion.

5.4.1 ADVAN4, TRANS1 – 2-comp 1st order input

ODE formulation:

$$\begin{aligned}\frac{dAd}{dt} &= -ka \times Ad \\ \frac{dAc}{dt} &= ka \times Ad - k_{12} \times Ac + k_{21} \times Ap - k \times Ac \\ \frac{dAp}{dt} &= k_{12} \times Ac - k_{21} \times Ap\end{aligned}$$

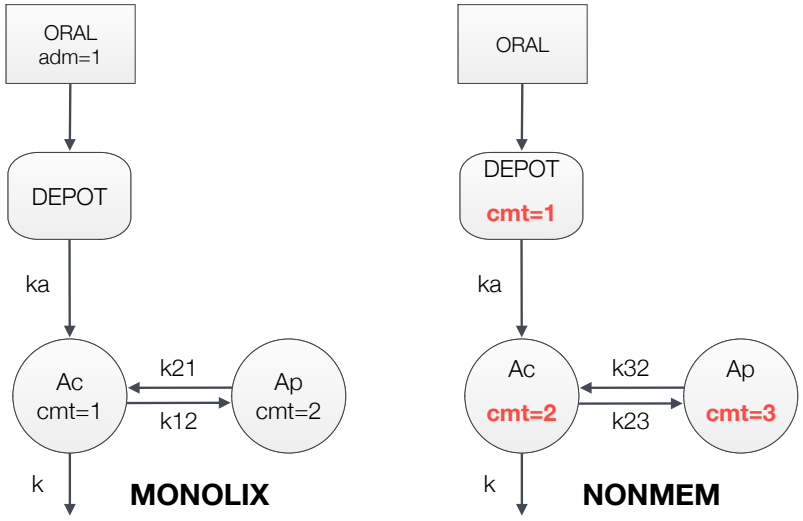


Figure 5.4: ADVAN4 model, with compartment numbering dependent on the target tool. Note, that not only the compartment numbers are different in NONMEM coded model, the rate constants names are different as well.

MONOLIX				
ID	TIME	AMT	**ADM**	Y
1	0	10	**1**	.
1	2	.	.	5
...	...	...	...	...

NONMEM				
ID	TIME	AMT	**CMT**	DV
1	0	10	**1**	.
1	2	.	2	5
...	...	...	...	...

Table 5.6: MONOLIX and NONMEM datasets for the ADVAN4 model.

PK macro
compartment(cmt=1, amount=Ac, volume=V)
peripheral(k12, k21, amount=Ap)
oral(adm=1, cmt=1, ka)
elimination(cmt=1, k)

Table 5.7: PK macros for the ADVAN4 model, as shown in Figure 5.4 (left).

PharmML code:

```
<StructuralModel blkId="sm4">
    <ct:Variable symbolType="real" symbId="Ac"/>
    <ct:Variable symbolType="real" symbId="Ap"/>
    <ct:Variable symbolType="real" symbId="Cc">
        <ct:Assign>
            <math:Equation>
                <math:Binop op="divide">
                    <ct:SymbRef symbIdRef="Ac"/>
                    <ct:SymbRef blkIdRef="pm1" symbIdRef="V"/>
                </math:Binop>
            </math:Equation>
        </ct:Assign>
    </ct:Variable>
    <PKmacros>
        <Compartment>
            <Value argument="cmt">
                <ct:Int>1</ct:Int>
            </Value>
            <Value argument="amount">
                <ct:SymbRef symbIdRef="Ac"/>
            </Value>
            <Value argument="volume">
                <ct:SymbRef blkIdRef="pm1" symbIdRef="V"/>
            </Value>
        </Compartment>
        <Peripheral>
            <Value>
                <ct:SymbRef blkIdRef="pm1" symbIdRef="k12"/>
            </Value>
            <Value>
                <ct:SymbRef blkIdRef="pm1" symbIdRef="k21"/>
            </Value>
            <Value argument="amount">
                <ct:SymbRef symbIdRef="Ap"/>
            </Value>
        </Peripheral>
        <Oral>
            <Value argument="adm">
                <ct:Int>1</ct:Int>
            </Value>
            <Value argument="cmt">
                <ct:Int>1</ct:Int>
            </Value>
            <Value>
                <ct:SymbRef blkIdRef="pm1" symbIdRef="ka"/>
            </Value>
        </Oral>
        <Elimination>
            <Value argument="cmt">
                <ct:Int>1</ct:Int>
            </Value>
            <Value>
                <ct:SymbRef blkIdRef="pm1" symbIdRef="k"/>
            </Value>
        </Elimination>
    </PKmacros>
</StructuralModel>
```

5.4.2 ADVAN12, TRANS1 – 3-comp 1st order input

ODE formulation:

$$\begin{aligned}
\frac{dAd}{dt} &= -ka \times Ad \\
\frac{dAc}{dt} &= ka \times Ad - k_{12} \times Ac + k_{21} \times Ap1 - k_{13} \times Ac \\
&\quad + k_{31} \times Ap2 - k \times Ac \\
\frac{dAp1}{dt} &= k_{12} \times Ac - k_{21} \times Ap1 \\
\frac{dAp2}{dt} &= k_{13} \times Ac - k_{31} \times Ap2
\end{aligned}$$

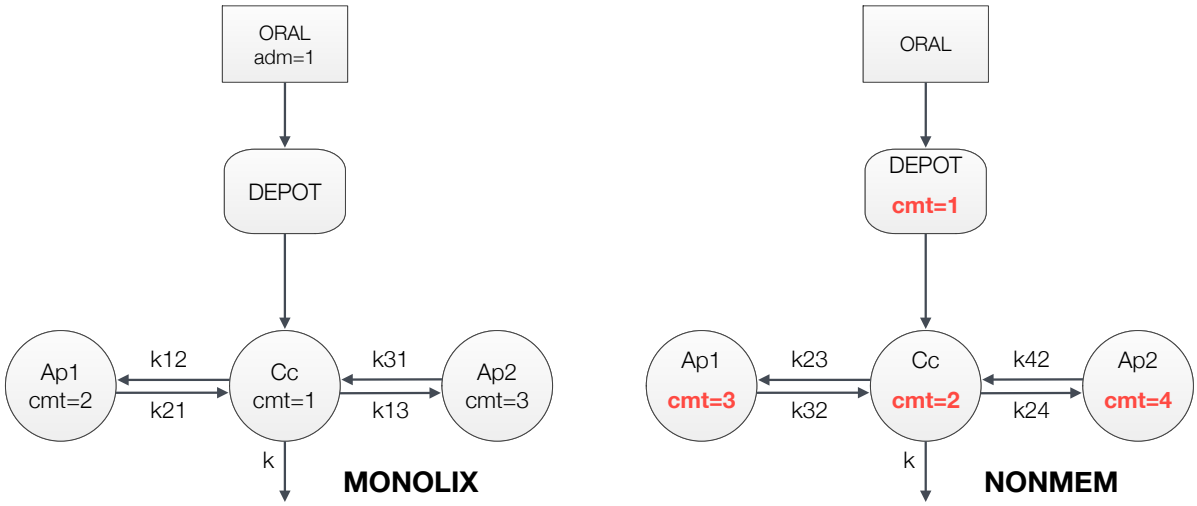


Figure 5.5: ADVAN12 model, with compartment numbering dependent on the target tool. Note, that not only the compartment numbers are different in NONMEM coded model, the rate constants names are different as well.

MONOLIX				
ID	TIME	AMT	**ADM**	Y
1	0	10	**1**	.
1	2	.	.	5
...	...	...	...	...

NONMEM				
ID	TIME	AMT	**CMT**	DV
1	0	10	**1**	.
1	2	.	2	5
...	...	...	...	...

Table 5.8: MONOLIX and NONMEM datasets for the ADVAN12 model.

PK macro
compartment(cmt=1, amount=Ac, volume=V)
peripheral(k12, k21, amount=Ap1)
peripheral(k13, k31, amount=Ap2)
oral(adm=1, cmt=1, ka)
elimination(cmt=1, k)

Table 5.9: PK macros for the ADVAN12 model, as shown in Figure 5.5 (left).

PharmML code:

```
<StructuralModel blkId="sm12">
    <ct:Variable symbolType="real" symbId="Ac"/>
    <ct:Variable symbolType="real" symbId="Ap1"/>
    <ct:Variable symbolType="real" symbId="Ap2"/>
```

```
            <ct:Variable symbolType="real" symbId="Cc">
                <ct:Assign>
                    <math:Equation>
                        <math:Binop op="divide">
                            <ct:SymbRef symbIdRef="Ac"/>
                            <ct:SymbRef blkIdRef="pm1" symbIdRef="V"/>
                        </math:Binop>
                    </math:Equation>
                </ct:Assign>
            </ct:Variable>

            <PKmacros>
                <Compartment>
                    <Value argument="cmt">
                        <ct:Int>1</ct:Int>
                    </Value>
                    <Value argument="amount">
                        <ct:SymbRef symbIdRef="Ac"/>
                    </Value>
                    <Value argument="volume">
                        <ct:SymbRef blkIdRef="pm1" symbIdRef="V"/>
                    </Value>
                </Compartment>
                <Peripheral>
                    <Value>
                        <ct:SymbRef blkIdRef="pm1" symbIdRef="k12"/>
                    </Value>
                    <Value>
                        <ct:SymbRef blkIdRef="pm1" symbIdRef="k21"/>
                    </Value>
                    <Value argument="amount">
                        <ct:SymbRef symbIdRef="Ap1"/>
                    </Value>
                </Peripheral>
                <Peripheral>
                    <Value>
                        <ct:SymbRef blkIdRef="pm1" symbIdRef="k13"/>
                    </Value>
                    <Value>
                        <ct:SymbRef blkIdRef="pm1" symbIdRef="k31"/>
                    </Value>
                    <Value argument="amount">
                        <ct:SymbRef symbIdRef="Ap2"/>
                    </Value>
                </Peripheral>
                <Oral>
                    <Value argument="adm">
                        <ct:Int>1</ct:Int>
                    </Value>
                    <Value argument="cmt">
                        <ct:Int>1</ct:Int>
                    </Value>
                    <Value>
                        <ct:SymbRef blkIdRef="pm1" symbIdRef="ka"/>
                    </Value>
                </Oral>
                <Elimination>
                    <Value argument="cmt">
                        <ct:Int>1</ct:Int>
                    </Value>
                    <Value>
                        <ct:SymbRef blkIdRef="pm1" symbIdRef="k"/>
                    </Value>
                </Elimination>
            </PKmacros>
        </StructuralModel>
```

Note A detailed description of all remaining predefined PREDPP models (ADVAN 1-3, 10, 11), alternative parameterizations other then the default TRANS1 as well aa more complex examples of macro implementation can be found in a technical report [Swat et al., 2015] on PharmML related websites `http://ddmore.eu/pharmml` or `http://pharmml.org`.

Chapter 6

Language Overview

6.1 Introduction

This chapter provides the background knowledge to understand PharmML. We recommend that you read this chapter before working through the examples in chapter 9. The chapter starts by describing how a PharmML document is organised and then go on to illustrate some of the key concepts and constructs of the language. For example, among other things, we discuss variable and parameter scoping (section 6.3.2), how to write maths (sections 6.7 and 6.8), and how to define data (section 6.6.1). The chapter concludes with a discussion of the additional resources that we expect to be used in support of PharmML, but which are outside the scope of this language specification (section 6.14).

6.2 Organisation of PharmML

PharmML is organised into three main sections (6.1): *Model Definition*, *Trial Design* and *Modelling Steps*. This reflects the natural organisation of a pharmacometric model and is the organisation implicitly found in the M&S tools used by modellers in this area. Below we detail the purpose and organisation of each section.

6.2.1 Model Definition

The *Model Definition* defines the model, typically a population model, that describes the system under investigation and any variability between individuals in the population. The modeller may wish to use it for simulation, parameter estimation or other types of analysis and exploration. The *Model Definition* in turn is composed of another set of "models" that describe specific aspects of the overall model definition. These are described below.

6.2.1.1 Variability Model

Variability is the concept that underpins a pharmacometric model and the *Variability Model* enables us to describe this. Note that it is possible to describe variability in a PharmML model without defining random variability, but by using covariates. Therefore the use of the *Variability Model* is optional. In PharmML you can use this to define individual random variability, but also a hierarchy of variability above and/or below the level of the individual (e.g. inter-occasion variability). For more details of the theory behind the random variability model, see section 3.5.

6.2.1.2 Covariate Model

The *Covariate Model* describes the covariates, both continuous and discrete, to be used in the *Parameter Model* (figure 6.1). This defines a transformation, interpolation or probability distribution for a covariate. The formal description of the covariate model can be found in section 3.6.5.

6.2.1.3 Parameter Model

The *Parameter Model* principally describes the parameters of the model definition and is typically used to describe parameters with some level of variability (typically between subject variability). The parameter is

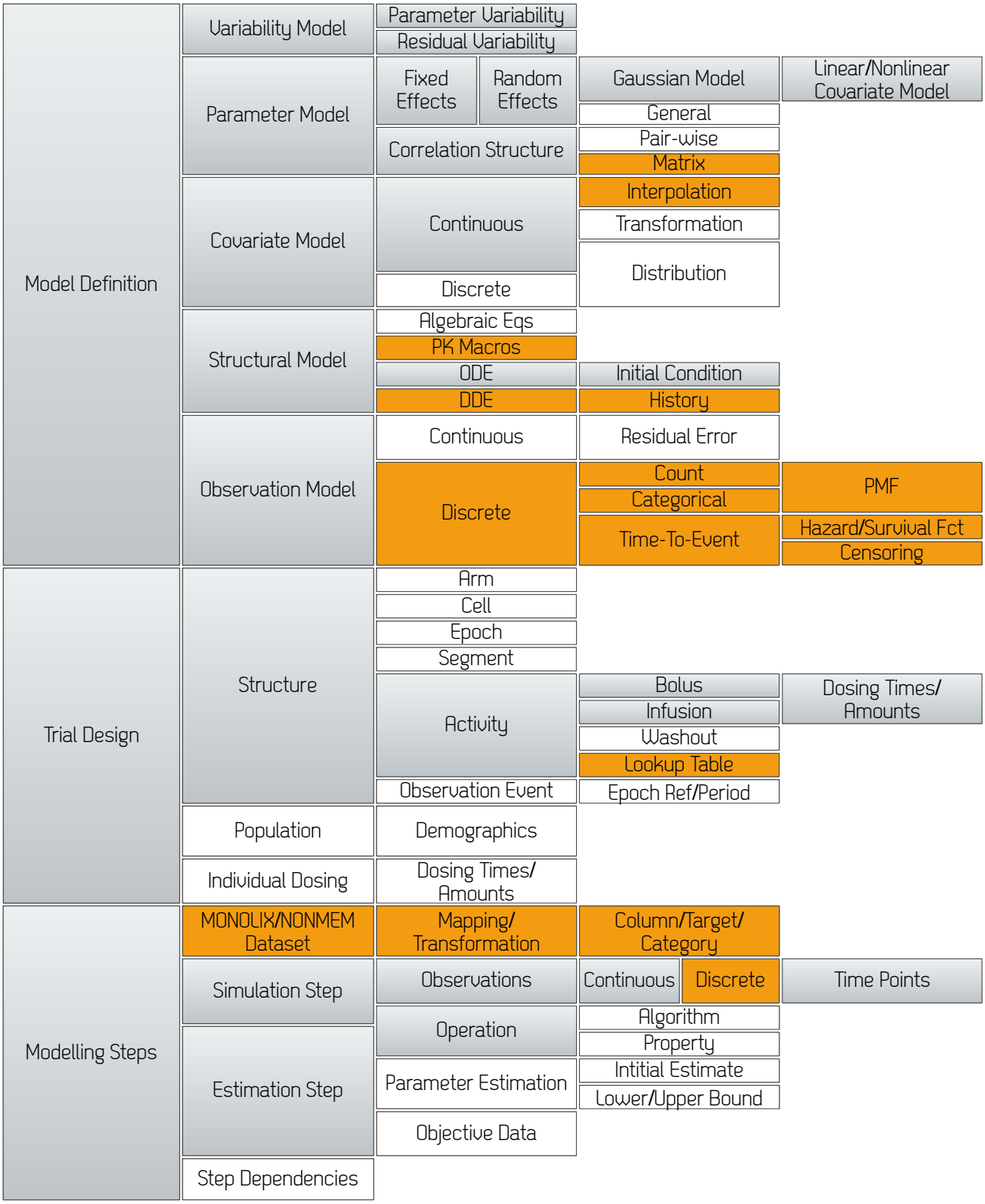


Figure 6.1: An overview of the organisation of PharmML. The orange highlighted cells illustrate the new elements introduced in this specification compared to the first public version 0.2.1. Grey indicate minor changes or extensions.

defined more formally in section 3.6, but essentially for each parameter we define a population term, one or more random effects, and its relationship to the covariates defined in the covariate model. The random

effects can be defined at different levels of variability (defined by the variability model, see section 6.2.1.1), which includes capturing their correlation structure by providing a covariance (or correlation) matrix for each level of variability.

6.2.1.4 Structural Model

At the heart of the model definition are one or more *Structural Model*s. These describe the system or systems that a modeller is interested in and they represent a particular abstraction of that system. For example a structural model may be used to describe the pharmacokinetics of a drug. We can represent PK, PD or PK-PD models as combinations of ODEs and algebraic equations. For compartmental PK models an alternative coding options are the PK macros, as described in the chapter 5.

6.2.1.5 Observation Model

In clinical trials experimental observations are made, and these observations are subject to experimental error. Different types of instrument, assay or material sampled will all have different statistical errors associated with them. In a pharmacometric model these errors are described using a residual error model (see section 3.7.1.1). In PharmML we encode the residual error model using the *Observation Model*.

The outcomes of a clinical trial that we wish to model are not always experimental measurements. A trial may aim to determine the efficacy of an analgesic using a pain score provided by the subject; or measure the frequency of seizure based on the maximum drug concentration; or establish the remission rate over a given time in a cancer trial [Bonate, 2011]. In each of these cases one needs to use a discrete statistical model to represent these outcomes. These are also defined in the Observation Model.

6.2.2 Trial Design

Clinical trials are carefully structured and can vary considerably in their complexity. Typically, a trial will be structured into one or more arms with each arm subject to one or more treatment regimens and observation protocols. Each arm is then populated with individuals from a population of subjects who have been screened for their suitability to participate in the trial. In PharmML we have two options: 1) the NONMEM/Monolix dataset to inform the model about the design, dosing and observation records, 2) `<TrialDesign>` section is used to describe the trial structure and subject population explicitly. The latter option is a novel approach making the clinical trial much clearer to document, easier to encode computationally and to use for simulations or optimal design. More information can be found in chapter 4.

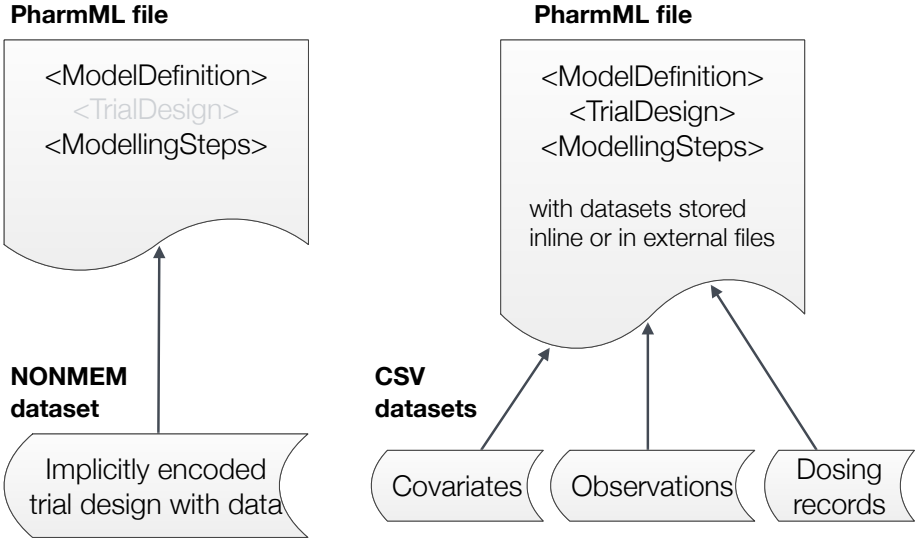


Figure 6.2: Alternative ways to work with PharmML– either the information about the underlying trial design and experimental records are sourced from a NONMEM type dataset or the trial design structure is encoded explicitly with datasets (covariates, observation and dosing records) stored inline within the PharmML model file or external CSV files, see also Figure 6.3 for more details.

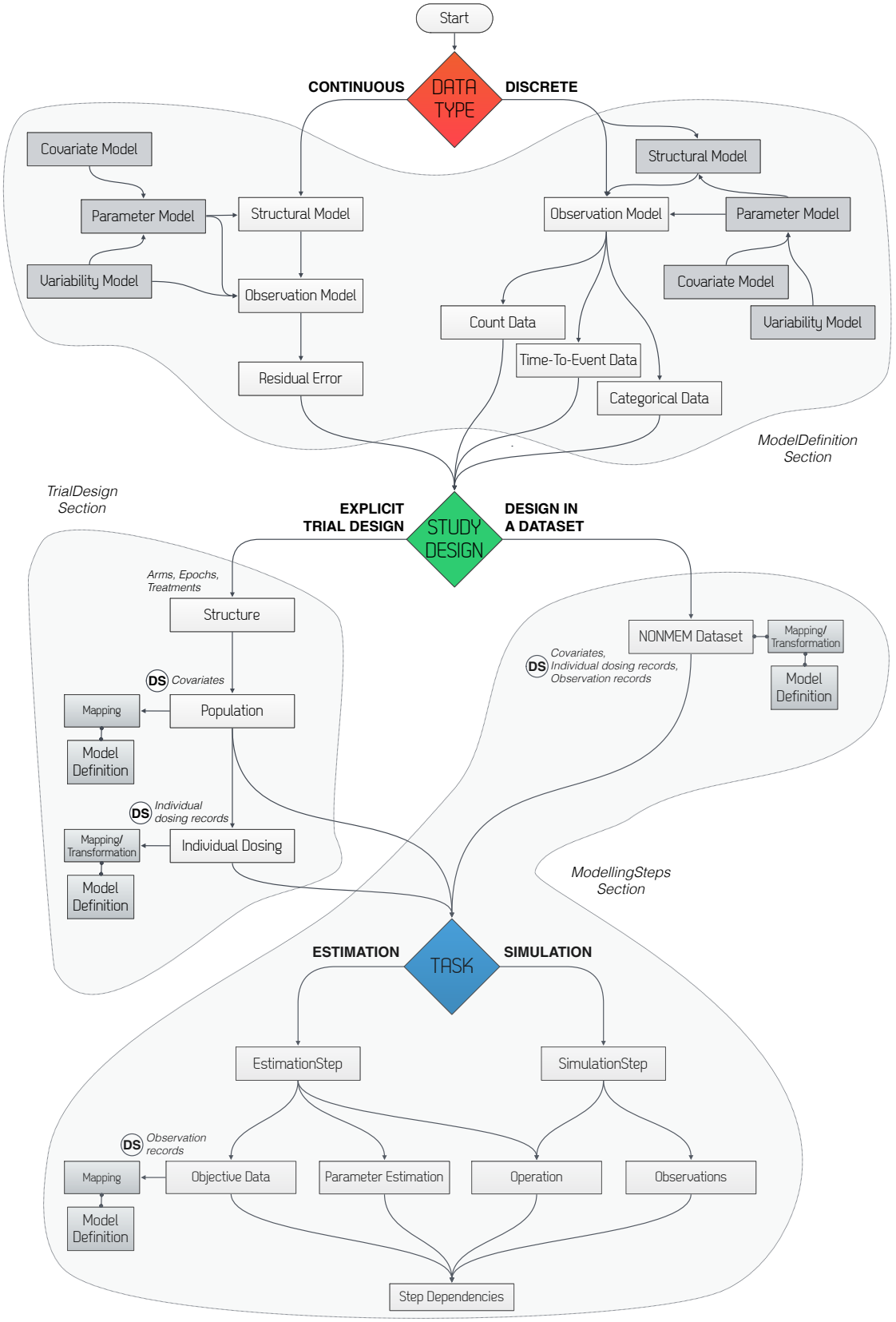


Figure 6.3: Working with PharmML– schema showing three essential decision points: the type of data (continuous and discrete), the source of the study design and data (NONMEM dataset or `<TrialDesign>`) and finally the task type (for now only simulation and estimation tasks are supported).

6.2.3 Modelling Steps

The final element when describing an M&S experiment, after defining the model and the associated trial design, is to describe how the model was used. This section of a PharmML document is akin to the Methods section of a paper. The aim is not to replicate your modelling exactly, but to provide enough information to reproduce the model[1].

6.3 Identifiers, references and namespaces

In PharmML we use the Object Identifier `oid` to identify components in the *Trial Design* and *Modelling Steps* sections of a PharmML document and the Symbol Identifier `symbId` to identify parameters and variables within the *Model Definition* section. Below we will describe the rules associated with how they are defined and referenced.

6.3.1 Object identifiers

The concept of the Object Identifier is borrowed from the CDISC XML description of a trial design [CDISC consortium, 2011]. There they use the attribute `oid` to identify and to reference components used in the design. Object identifiers have global scope, which means that all object identifiers defined in a PharmML document must be unique.

Elements that reference an object identifier by convention use the attribute `oidRef` and the id referred to must exist in the PharmML document. In addition the object referred to must be compatible with the element referencing it. The example below shows how this works:

```
<Epoch oid="e1">
  <!-- Detail omitted -->
</Epoch>
<Arm oid="a2">
  <!-- Detail omitted -->
</Arm>
<Cell oid="c2">
    <EpochRef oidRef="e1" />
    <ArmRef oidRef="a2"/>
    <SegmentRef oidRef="tb"/>
</Cell>
```

Here the element `<EpochRef>` refers to the object identifier of the Epoch, "e1", and the `<ArmRef>` element refers to the Arm object, "a2". This is correct. However, the following example is incorrect:

```
<Epoch oid="e1">
  <!-- Detail omitted -->
</Epoch>
<Arm oid="a2">
  <!-- Detail omitted -->
</Arm>
<Cell oid="c2">
    <!-- ERROR: not valid PharmML -->
    <EpochRef oidRef="a2" />
    <ArmRef oidRef="a2"/>
    <SegmentRef oidRef="tb"/>
</Cell>
```

The `<EpochRef>` points to the Arm object, which is not compatible with it. Note that this cannot be validated using SML Schema. These compatibilities are documented for each element containing an object reference (i.e. an `oidRef` attribute) in the XML Schema.

6.3.2 Blocks and symbol scoping

PharmML defines names for the parameters, variables and parts of the model that need to be uniquely identified. In PharmML we refer to these collectively as symbols. The rules we apply are relatively simple in that all symbols within a PharmML document must be unique and that all symbols in the document

[1] To replicate the execution of a model requires detailed information about not only what algorithms were used to simulate or execute a model, but also what software implementation was used and exact supporting libraries such as that of the random number generator.

are 'visible'. In other words a symbol defined in one part of a document will be available to a component elsewhere in the document. Symbols in PharmML can be organised into different scopes, which in turn are defined by blocks. We illustrate this conceptually in the example below[2]:

```
<Block blkId="blockID">
        ...
        <Symbol symbId="symbolID">
                ...
        </Symbol>
        ...
        <SymbRef symbIdRef="symbolID"/>
</Block>

<ElsewhereInXMLDocument>
        <SymbRef blkIdRef="blockID" symbIdRef="symbolID"/>
</ElsewhereInXMLDocument>
```

The block is given an identifier and is used to organise symbols. Any symbols that are defined within it are part of the block's scope (private symbols). If we refer to that symbol within the block then we do not need to specify the block name. When referred to outside the block, then the same symbol must be referred to using a combination of the block identifier (`blkId`) and symbol identifier (`symbId`). This is illustrated more completely in the example below:

```
<Symbol symbId="symb2"/> <!-- Decl 1 -->
<Symbol symbId="symb3"/> <!-- Decl 2 -->

<Block blkId="A">
        <Symbol symbId="symb2"/> <!-- Decl 3 -->
        <SymbRef symbIdRef="symb2"/> <!-- resolves to Decl 3 -->
        <SymbRef symbIdRef="symb3"/> <!-- resolves to Decl 2 -->
</Block>

<Block blkId="B">
        <Symbol symbId="symb2"/> <!-- Decl 4 -->
        <Symbol symbId="symb3"/> <!-- Decl 5 -->
        <SymbRef symbIdRef="symb2"/> <!-- resolves to Decl 4 -->
</Block>

<ElsewhereInXMLDocument>
        <SymbRef symbIdRef="symb2"/> <!-- resolves to Decl 1 -->
        <SymbRef blkIdRef="A" symbIdRef="symb2"/> <!-- resolves to Decl 3 -->
        <SymbRef blkIdRef="B" symbIdRef="symb2"/> <!-- resolves to Decl 4 -->
        <SymbRef blkIdRef="B" symbIdRef="symb3"/> <!-- resolves to Decl 5 -->
</ElsewhereInXMLDocument>
```

Here, the `<Symbol>` element defines a symbol and `<SymbRef>` refers to it. As you can see, a symbol can be defined in several places: globally (outside a block) and within blocks A and B. In each case identical `symbID`s are used, but the language can distinguish between them because of the context. This is clear when we look at the symbol references in the `<ElsewhereInXMLDocument>` element. Referring to a symbol, for example symbol "`symb2`" in block A may seem ambiguous, but the scoping rules of PharmML are clear. The reference is resolved first to the scope within the block and then to the global scope. So in block A the reference to "`symb2`" points to Decl 3, and the reference to "`symb3`" points to the global symbol, Decl 2 and not Decl 5, which is a different scope. These scoping rules are common to many programming languages. One question you may ask is what if a globally defined symbol has the same name as a block identifier? This is handled by the symbol namespace rules. Both types of identifier share the same (global) namespace and so cannot have the same name.

You will notice in the discussion above that we use the words 'define' and 'reference'. These are important concepts in PharmML. Symbols can be *defined* only once, but can be *referred* to many times. Symbols can be referred to before they are defined. PharmML is a declarative language (unlike C or Fortran), thus it is natural that the order of variable definition is not important. The listing below shows how this works using real PharmML.

```
<ct:Variable symbId="c" symbolType="real">
   <ct:Assign>
      <Equation xmlns="http://www.pharmml.org/pharmml/0.6/Maths">
         <Binop op="divide">
```

[2]Note that in the example we use the element `<Symbol>` to define a symbol and `<Block>` a block. These are not actually valid PharmML elements, but we hope to make the scoping discussion clearer by using these.

```
                <ct:SymbRef symbIdRef="b"/>
                <ct:Real>10</ct:Real>
            </Binop>
        </Equation>
    </ct:Assign>
</ct:Variable>
<ct:Variable symbId="a" symbolType="real">
    <ct:Assign>
        <Equation xmlns="http://www.pharmml.org/pharmml/0.6/Maths">
            <Binop op="plus">
                <ct:SymbRef symbIdRef="b"/>
                <ct:SymbRef symbIdRef="c"/>
            </Binop>
        </Equation>
    </ct:Assign>
</ct:Variable>
<ct:Variable symbId="b" symbolType="real">
    <ct:Assign>
        <ct:Real>1</ct:Real>
    </ct:Assign>
</ct:Variable>
```

One danger with this approach is that the language syntax does not prevent the creation of cyclic dependencies between variables. An example of this is shown below where there is dead-lock because neither variable can be initialised because the other is yet to be defined. Such cycles are forbidden in PharmML and must be checked for when validating the language. (See section 8.4.1 with the description of related validation rules, which are already or are supposed to be implemented and handled by libPharmML[3], an API under development as part of the DDMoRe project.)

```
<!-- ERROR: The declaration below creates a cycle -->
<ct:Variable symbId="d" symbolType="real">
    <ct:Assign>
        <ct:SymbRef symbIdRef="e"/>
    </ct:Assign>
</ct:Variable>
<ct:Variable symbId="e" symbolType="real">
    <ct:Assign>
        <ct:SymbRef symbIdRef="d"/>
    </ct:Assign>
</ct:Variable>
```

Related to variable definition is the initialisation of a symbol: also known as initial assignment. When a symbol is defined it is in an uninitialised state and has no value. It may be either initialised during the definition, as in the examples above, or via a subsequent initial assignment as below:

```
<!-- Symbol defined, but not initialised -->
<ct:Variable symbId="a" symbolType="real"/>
<!-- Omitted detail -->
<ct:VariableAssignment>
    <ct:SymbRef symbIdRef="a"/>
    <ct:Assign>
        <math:Equation>
            <math:Binop op="plus">
                <ct:SymbRef symbIdRef="a"/>
                <ct:Real>10</ct:Real>
            </math:Binop>
        </math:Equation>
    </ct:Assign>
</ct:VariableAssignment>
```

Either way this can only be done once. This listing illustrates the problem:

```
<!-- Symbol defined, but not initialised -->
<ct:Variable symbId="a" symbolType="real"/>
<!-- Snip -->

<!-- Incorrect -->
<!-- Duplicate initial assignments here. -->
<ct:VariableAssignment>
    <ct:SymbRef symbIdRef="a"/>
```

[3] https://sourceforge.net/projects/libpharmml.ddmore.p

```
    <ct:Assign>
        <ct:Real>0</ct:Real>
    </ct:Assign>
</ct:VariableAssignment>
<ct:VariableAssignment>
    <ct:SymbRef symbIdRef="a"/>
    <ct:Assign>
        <math:Equation>
            <math:Binop op="plus">
                <ct:SymbRef symbIdRef="a"/>
                <ct:Real>10</ct:Real>
            </math:Binop>
        </math:Equation>
    </ct:Assign>
</ct:VariableAssignment>
```

PharmML is a declarative language and the order is not important, thus symbols in PharmML can only be initialised once.

Note that in PharmML symbol references between sections only go in one direction. All sections point to the *Model Definition*, but not the reverse and the *Modelling Steps* section points to the *Trial Design* section, but again not *vice versa*. By maintaining this layered dependency structure in the design of PharmML we simplify the design of the language and ensure that the *Model Definition* section is guaranteed to be independent of the other PharmML sections.

6.3.3 Interaction between Object and Symbol identifiers

The Object and Block identifier (`oid` and `blkId` respectively) both exist in the same PharmML document and they share the same namespace. This means that they cannot share the same identifier, as is illustrated below:

```
<Symbol symbId="symb2"/>

<Block blkId="A"> <!-- ERROR -->
       <Symbol symbId="symb3"/>
</Block>

<Block blkId="B"> <!-- OK -->
       <Symbol symbId="symb4"/>
</Block>

<Oid oid="A"/> <!-- ERROR -->
<Oid oid="Z"/> <!-- OK -->
<Oid oid="symb2"/> <!-- ERROR -->
<Oid oid="symb3"/> <!-- OK -->
```

Similarly, just as a global `symbId` cannot share an identifier with a `blkId` (see section 6.3.2), neither can it share an identifier (in the above case "symb2") with an `oid`.

6.4 Type checking

Symbols in PharmML have a type. By symbol we mean something defined using a `symbId` attribute (for example a variable or parameter). Like a variable a type is simply a way we use in PharmML to map symbols to an abstraction: such as a number, a string or a table of values. As you might expect, symbols with different types are not always compatible with each other, so it is necessary when validating the correctness of a PharmML document to ensure that the types of its symbols are compatible with each other. This is known as type checking (see [Aho et al., 1986, Chapter6] and [Parr, 2010, Chapter 8] for more information). The types in PharmML are enumerated in table 8.3.

The types used in PharmML must be consistent. In general this means that all types in an expression should be identical. This is illustrated in the following example:

```
<!-- ERROR: incompatible type -->
<ct:Variable symbId="a" symbolType="int">
    <ct:Assign>
        <ct:String>A value</ct:String>
    </ct:Assign>
</ct:Variable>
```

```
<ct:Variable symbId="b" symbolType="boolean"/>
<ct:Variable symbId="c" symbolType="real">
    <ct:Assign>
        <m:Equation>
            <m:Binop op="plus"> <!-- ERROR: Cannot add a real to a Boolean -->
                <ct:Real>22</ct:Real>
                <ct:SymbRef symbIdRef="b"/>
            </m:Binop>
        </m:Equation>
    </ct:Assign>
</ct:Variable>
<ct:Variable symbId="d" symbolType="real">
    <ct:Assign>
        <ct:Int>453</ct:Int> <!-- OK -->
    </ct:Assign>
</ct:Variable>
```

Note that the variable d is of type real but was initialised with an integer value, and that this was permitted. This is an exception to the rule that all types must be the same and is a common mechanism in computer languages, called type promotion. Here the integer value can be converted to a real with no loss of information and so it is permitted. The reverse conversion is not permitted because a real value may lose information when converted to an integer.

6.5 Two ways to connect trial design and model

There are two basic options[4] to provide trial design related information in PharmML. In the first the entire information is implemented in the PharmML file, while in the second datasets with experimental data are used, see Figure 6.2 and 6.3.

`<TrialDesign>` and inline datasets	External NONMEM dataset

`<ModelDefinition>`
`...`
`</ModelDefinition>`
`<TrialDesing>`
`<Structure>`
`...`
`</Structure>`
`<Population>`
`...`
`</Population>`
`</TrialDesing>`
`<ModellingsSteps>`
`<EstimationStep oid="est1">`
`<ObjectiveDataSet>`
`...`
`</ObjectiveDataSet>`
`...`
`</EstimationStep>`
`...`
`</ModellingsSteps>`
	`<ModelDefinition>`
	`...`
	`</ModelDefinition>`
	
	`<!-- TrailDesign section is not required -->`
	
	`<ModellingsSteps>`
	`<ExternalDataSet toolName="NONMEM" oid="DSoid">`
	`...`
	`</ExternalDataSet>`
	
	`<EstimationStep oid="est1">`
	`<ExternalDataSetReference>`
	`<ct:OidRef oidRef="DSoid"/>`
	`</ExternalDataSetReference>`
	`...`
	`</EstimationStep>`
	`...`
	`</ModellingsSteps>`

Table 6.1: Comparison of model structures encoded using `<TrialDesign>` (left) and using NONMEM format dataset (right). In the latter case the `<TrialDesign>` part is redundant and the complete information about the study design is sourced from the dataset. This holds for both estimation and simulation tasks.

Table 6.1 shows the PharmML code for these two options as described in the following

- Using `<TrialDesign>`/`<ModellingSteps>` – trial design (information about design arms, dosing times, amounts etc.) is encoded explicitly in XML and according experimental data records are provided within three datasets. More specifically covariates and individual dosing records are encoded in the `<Population>` element of the `<TrialDesign>` section, while the observation records, in PharmML referred to as `<ObjectiveDataSet>`, are stored in `<ModellingSteps>`, see Table 6.1 (left).

[4] In version 0.2.1 only the `<TrailDesign>` based option was supported. This however proved to be very limiting for various reasons and support for NONMEM/Monolix datasets has been required.

- Using NONMEM/Monolix datasets – these files contain the complete information about the underlying trial design, both for estimation or simulation tasks. Using datasets has consequences for the content of the PharmML document in that the `<TrialDesign>` part is in such case redundant. Instead the dataset is referenced in the `<ExternalDataSet>` element and referenced later in e.g. estimation task, `<ExternalDataSetReference>`, see Table 6.1 (right).

6.6 Datasets

The dataset is a key concept in PharmML and is used to describe the data of the relevant trial design, i.e. the observations, covariates and dosing records. The data is tabular and the dataset has therefore been designed to represent this type of information and all data is explicitly typed.

6.6.1 Inline datasets

We start with the data implemented entirely within the PharmML file. As usual the simplest way to explain it is to look at an example, such as the code snippet below:

```
<ds:DataSet>
    <ds:Definition>
        <ds:Column columnId="id" columnType="id" valueType="string" columnNum="1"/>
        <ds:Column columnId="arm" columnType="arm" valueType="string" columnNum="2"/>
        <ds:Column columnId="reps" columnType="replicate" valueType="int" columnNum="3"/>
    </ds:Definition>
    <ds:Table>
        <ds:Row>
            <ct:String>i1</ct:String><ct:String>a1</ct:String><ct:Int>20</ct:Int>
        </ds:Row>
        <ds:Row>
            <ct:String>i2</ct:String><ct:String>a2</ct:String><ct:Int>20</ct:Int>
        </ds:Row>
        <ds:Row>
            <ct:String>i3</ct:String><ct:String>a3</ct:String><ct:Int>40</ct:Int>
        </ds:Row>
        <ds:Row>
            <ct:String>i4</ct:String><ct:String>a4</ct:String><ct:Int>40</ct:Int>
        </ds:Row>
    </ds:Table>
</ds:DataSet>
```

The dataset starts with `<Definition>` where the columns of the dataset table are defined. For each column the following attributes have to be specified

- `columnId` – the name of the column
- `columnType` – the type of the column identifies the role the stored item plays in the model, see Table 6.2 for the explanation of allowed values.
- `valueType` – the type of each column value is specified, which complies with the PharmML type system.
- `columnNum` – the column number must start at 1 and each column must be numbered in consecutive order (i.e. 1,2,3,4. . . etc.).

Next the content of the dataset is held within the `<Table>` element and this consists of one or more `<Row>` elements. Each row must contain an entry for each column defined. This approach is based on the concept of a relation in relational theory. Therefore the ordering of rows is not significant.

6.6.2 NONMEM/Monolix datasets

Inline storage of data split into three tables has disadvantages in that it requires provision of additional translation tools to convert standard datasets to this format. Editing such data sets by hand is very error prone and time consuming.

There was a lot of request for supporting external datasets, more specifically those used by NONMEM, Monolix and other tools. As shown in the following listing, the external data set can be specified using the new `<ImportData>` element, instead of `<Table>`, by providing the file path relative to the PharmML file.

columnType	Meaning
addl	number of additional doses
adm	type of administration
arm	study arm
censoring	left-censored data
covariate	covariates
cmt	target compartment
demographic	demographic type
dose	dose
duration	infusion duration
dv	dependent variable
dvid	mixed observations
epoch	study epoch
evid	dose events
id	subject identifiers
idv	independent variable
ii	inter-dose interval
limit	lower limit for interval-censored data
mdv	missing dependent variable
occasion	occasions
rate	infusion rate
reg	regression variable
replicate	in cases when subjects are assumed identical
ss	steady state
ssEndTime	steady-state administration end time
ssPeriod	steady-state administration interval
time	time
undefined	value for any undefined cases

Table 6.2: A summary of allowed values for the `columnType` attribute in PharmML.

Optional parameters such as file format and delimiter can also be provided. The external file must be a valid input file for the target tool. The template for this reads:

```
<ds:ExternalFile oid="id1">
    <ds:path>FILE_PATH</ds:path>
    <ds:format>FORMAT</ds:format>
    <ds:delimiter>DELIMITER_TYPE</ds:delimiter>
</ds:ExternalFile>
```

And listing below shows a simple example how this works. As before the `<Definition>` defines the table content and a reference to the according file and its type is provided.

```
<ds:DataSet>
    <ds:Definition>
        <ds:Column columnId="ID" columnType="id" valueType="id" columnNum="1"/>
        <ds:Column columnId="ARM" columnType="arm" valueType="id" columnNum="2"/>
        <ds:Column columnId="SEX" columnType="covariate" valueType="id" columnNum="3"/>
        <ds:Column columnId="EPOCH" columnType="epoch" valueType="id" columnNum="4"/>
    </ds:Definition>
    <ds:ExternalFile oid="id1">
        <ds:path>warfarin_conc_pca.csv</ds:path>
        <ds:format>CSV</ds:format>
        <ds:delimiter>COMMA</ds:delimiter>
    </ds:ExternalFile>
</ds:DataSet>
```

6.6.3 `<TrialDesign>` external datasets

In the `<TrialDesign>` section the user has the choice between inline or external dataset storage, see Section 6.2.2. External datasets can be specified using the `<ImportData>` element. As shown in the example below,

you must provide the file path relative to the PharmML file, file format (only CSV files allowed) and one of the allowed delimiters (`COMMA`, `SEMICOLON`, `SPACE`, `TAB`). Further validation rules are listed in xxx.

```
<ds:ExternalFile oid="id1">
    <ds:path>/datasets/myData.csv</ds:path>
    <ds:format>CSV</ds:format>
    <ds:delimiter>COMMA</ds:delimiter>
</ds:ExternalFile>
```

6.6.4 Lookup table

`<TrialDesign>` can be used to define a wide range of dosing scenarios which can be used to encode virtually any PK model. Sometimes however, the concentration data we want to couple with a PD model is available as a lookup table, i.e. PK measurement records for which the underlying model is unknown or not required.

First the data has to be implemented and this is done using the `<LookupTable>` element within the `<Activity>` of the `<TrialDesign>`. The target variable in the model needs to be defined and finally an interpolation algorithm chosen from a build-in list with the most common choices, i.e. constant, linear, nearest, spline, pchip and cubic. There are of course other scenarios when a lookup table is useful, e.g. in the minimal model where the measured insulin is provided in tabular format.

6.6.5 Dataset mapping and scaling

The provision of a dataset, implemented as inline data, as external files or as lookup tables is not sufficient to make it work. What is additionally required is a set of consistent rules and structure for the mapping between various model elements (such as administration targets, continuous/discrete observations, continuous/discrete covariates, categories in discrete models, occasions) and data records. The same holds for mapping between above mentioned model elements and various administration options as defined in `<TrialDesign>`. We will see lots of examples for such mappings in Chapter 9.

Similar holds for the dose adjustment, a very frequent element of a PK study, which is tightly coupled with column mapping. Typically such adjustment means scaling with respect to bodyweight, BW, or body surface area, BSA, or other continues covariates, such as creatine clearance, CLcr. We will discuss examples for such mappings which are defined in connection with the datasets in Chapter 9.

6.7 Defining differential equations

The easiest way to understand how one defines a differential equation in PharmML is to look at an example.

$$\frac{\mathrm{d}Ad}{\mathrm{d}t} = -ka\, Ad, \quad Ad(t = t_0) = A_0$$

The corresponding PharmML to the above equation is listed below:

```
<ct:DerivativeVariable symbId="Ad" symbolType="real">
    <ct:Assign>
        <Equation xmlns="http://www.pharmml.org/pharmml/0.6/Maths">
            <Binop op="times">
                <Uniop op="minus">
                    <ct:SymbRef blkIdRef="p1" symbIdRef="ka"/>
                </Uniop>
                <ct:SymbRef symbIdRef="Ad"/>
            </Binop>
        </Equation>
    </ct:Assign>
    <ct:IndependentVariable>
        <ct:SymbRef symbIdRef="t"/>
    </ct:IndependentVariable>
    <ct:InitialCondition>
        <ct:InitialValue>
            <ct:Assign>
                <ct:SymbRef symbIdRef="A0"/>
            </ct:Assign>
        </ct:InitialValue>
        <ct:InitialTime>
```

```
                <ct:Assign>
                    <ct:SymbRef symbIdRef="t0"/>
                </ct:Assign>
            </ct:InitialTime>
        </ct:InitialCondition>
    </ct:DerivativeVariable>
```

As you can see a derivative variable is defined using the `<DerivativeVariable>` element and the right-hand side of the equation is described by the `<Assign>` element. The independent variable is explicitly defined in this example using the `<IndependentVariable>`. If it had been omitted then the derivative would have defaulted to the independent variable set for the PharmML document as a whole. Finally its initial condition is set to a constant A_0 at t_0 in the `<InitialCondition>` element, but in fact any expressions are allowed in `<InitialValue>` and `<InitialTime>`. There are a few points to note about the definition of the derivative:

1. The symbol types on the RHS of the definition are a mixture of derivative variables and non-derivative variables and parameters.

2. The definition of the variable Ad contains a reference to itself. If this were the definition of a non-derivative type, then this would be regarded as a cyclic dependency and not be permitted, but in the definition of an ODE it is.

6.7.1 Delayed differential equations

The following new elements extend the ODE structure

- `<Delay>` element with arguments
  - y, a model variable
  - τ, discrete delay, can be a numerical value or a symbol
- `<History>` element where the past of a variable is defined for $t \leq t_0$. It comes with two child elements
  - `<HistoryValue>` stands for past/historical value of a variable y, denoted by y_0.
  - `<HistoryTime>` stands for the end time, t0, of the history definition. By default, history is defined for $t \leq t_0$.

Then the typical delay expression such as $y(t - \tau)$ would be encoded as

```
<ct:Delay>
    <ct:SymbRef symbIdRef="y"/>
    <ct:DelayVariable >
        <ct:SymbRef blkIdRef="pm1" symbIdRef="tau"/>
    </ct:DelayVariable>
</ct:Delay>
```

with τ defined in the `<ParameterModel>` *pm1*.

6.8 Mathematical expressions

Mathematical expressions are a fundamental part of a pharmacometric model and so it was important that PharmML incorporated the ability to encode these. The question we had in designing the language, however, was what is the best way to do this? Our initial approach was to reuse an existing W3C standard called MathML[5], which was designed to represent mathematical equations on web pages. Unfortunately, the full MathML standard is bigger and more complex than we need: indeed much of the standard focuses on the presentation and layout of mathematical equations rather than their underlying meaning[6]. This was also the conclusion reached for similar standards to PharmML such as SBML [Hucka et al., 2010, Section 3.4], CellML[7] and SED-ML [Waltemath et al., 2011]. Their solution to this problem was to use a subset of the standard that did what they wanted and to develop their own software to support this subset. In effect they created their own version of the MathML standard. This means that the CellML version of MathML is not

[5] `http://www.w3.org/TR/MathML3/`
[6] We should emphasise that his is not a criticism of MathML, as this was the problem it was created to solve!
[7] `http://www.cellml.org/specifications/cellml_1.1/#sec_mathematics`

compatible with the SBML version and so on, and as a consequence each standard has had to develop its own software libraries to support their own version of MathML.

Faced with the same dilemma we considered adopting yet another subset of MathML, but decided against it for a number of reasons:

1. Because MathML is designed for the presentation of maths its basic design is much more complicated than we require.

2. The design of MathML is such that it is impossible to validate whether a sensible mathematical expression has been formed using just XML Schema validation[8]. This is because it uses `<apply></apply>` elements to group operands and operators together and so a statement such as `<apply><divide/><cn>20/<cn></apply>` ($\div 20$) is syntactically valid MathML, but an incomplete mathematical expression.

3. Taking a subset of MathML requires the creation of a new XML Schema definition, new tools for validation and is effectively creating a new standard. In our view calling this MathML is misleading as each of the MathML subsets currently used are not the same and cannot be exchanged with each other, nor with W3C MathML (see discussion above).

Consequently we created our own mathematics definition, which has the following design goals:

1. Have a design that ensured that mathematical expressions were syntactically correct — allowing us to use XML Schema validating software to ensure this correctness.

2. Ensure that the maths could handle all mathematical expressions we require in PharmML.

3. Provide logical expressions for use in piecewise functions.

4. Have a simple and concise design that could be easily written by hand and also read by a developer — to facilitate testing.

Our design follows that of many programming languages, such as C [Kernighan and Ritchie, 1988], by defining unary and binary operators that take one or two operands respectively. Such operands can be literal values (e.g. numbers), variables or another operator. In languages such as C, mathematical expressions are designed to be easily read by humans, but in PharmML we don't have this restriction and we are more interested in ease of computational processing. For this reason we have adopted a prefix representation.

In a prefix representation, also called Polish notation[9], the operator is placed before its operands. We can illustrate this using the following expression, $(9 - 5) \times 2$, becomes $\times - 9\,5\,2$. This is evaluated from left to right. You first evaluate the operator which has operands that are numerical values. The result of this operator is then used as an operand of another operator and the process is repeated until all operators are evaluated. Using the expression above as an example: $-9\,5$ is evaluated first that then reduces the expression to $\times 4\,2$, until finally we are left with the result of 2. The benefits for the parser are obvious because we no longer require grouping constructs like parenthesis. As can be seen in the following listing, this prefix approach fits well with XML and allows us to express the above expression concisely[10].

```
<!-- (9 - 5) * 2 -->
<Equation xmlns="http://www.pharmml.org/pharmml/0.6/Maths"/>
    <Binop op="times">
        <Binop op="minus">
            <ct:Real>9</ct:Real>
            <ct:Real>5</ct:Real>
        </Binop>
        <ct:Int>2</ct:Int>
    </Binop>
</Equation>
```

It also allows us to use XML Schema validation to ensure correctness because we can validate that all binary operators require two operands and a unary operator one. The more complicated example below shows how to define the expression $\exp\left(-\text{logit}(i) + \beta \ln\left(\frac{W}{70}\right) + \eta\right)$ with unary and binary operators:

[8] XML Schema is an XML standard that let's you effectively define an object model in XML. The benefit of the standard is that it there many tools that can then validated automatically whether your XML document conforms to this 'object model'. We have taken advantage of this technology in PharmML and it has made development of the specification and software support much more efficient.

[9] For more information see `http://en.wikipedia.org/wiki/Polish_notation`.

[10] In fact the XML structure actually defines the abstract syntax tree of the mathematical expression, which is typically the output of a language parser.

```
<Equation xmlns="http://www.ddmore.eu/pharmml/0.6/Maths"/>
    <!-- Omitted namespace declarations -->
    <Uniop op="exp">
        <Binop op="plus">
            <Uniop op="minus">
                <Uniop op="logit">
                    <ct:SymbRef symbIdRef="i"/>
                </Uniop>
            </Uniop>
            <Binop op="plus">
                <Binop op="times">
                    <ct:SymbRef symbIdRef="beta"/>
                    <Uniop op="ln">
                        <Binop op="divide">
                            <ct:SymbRef symbIdRef="W"/>
                            <ct:Real>70</ct:Real>
                        </Binop>
                    </Uniop>
                </Binop>
                <ct:SymbRef symbIdRef="eta"/>
            </Binop>
        </Binop>
    </Uniop>
</Equation>
```

Besides mathematical expressions PharmML Maths can also define logical expressions used in conditional logic that enables us to define piecewise functions.

This uses the same postfix approach, but with alternate logical binary and unary operators defined by the `<LogicBinop>` and `<LogicUniop>` elements, respectively. The following example shows how it can be combined with mathematical expressions to describe the piecewise expression:

$$\begin{cases} -x & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}$$

Note that because logical expressions can contain strings it is possible to define such expressions using non-numerical criteria:

```
<Equation xmlns="http://www.ddmore.eu/pharmml/0.6/Maths/"/>
    <!-- Omitted namespace declarations -->
    <Piecewise>
        <Piece>
            <Uniop op="minus">
                <ct:SymbRef symbIdRef="x"/>
            </Uniop>
            <Condition>
                <LogicBinop op="lt">
                    <ct:SymbRef symbIdRef="x"/>
                    <ct:Int>0</ct:Int>
                </LogicBinop>
            </Condition>
        </Piece>
        <Piece>
            <ct:SymbRef symbIdRef="x"/>
            <Condition>
                <LogicBinop op="geq">
                    <ct:SymbRef symbIdRef="x"/>
                    <ct:Real>0</ct:Real>
                </LogicBinop>
            </Condition>
        </Piece>
    </Piecewise>
</Equation>
```

A complete list of the mathematical and operators that are available is provided in section 8.6. Here you will also find a description of each operator's semantics and the permitted types of its operands.

6.9 Vectors and Matrices

Current version of PharmML provides support for vectors and matrices, which for can be mainly used for encoding of correlation/covariance matrices and the storing of matrices in the Standardised Output (SO),

such as Fisher Information Matrix (FIM) of the estimated parameters 1.3. The schema for these new elements is based on two standards, one dealing with mathematical notation, MathML [W3C, 2010], and another one with data mining models, PMML [The Data Mining Group (DMG),], and comes with a wide range of features enabling flexible handling of vectors and matrices via a indexing schema.

The following section provide a brief overview of their structures, a more detailed description can be found in a dedicated report on PharmML website `http://pharmml.org`.

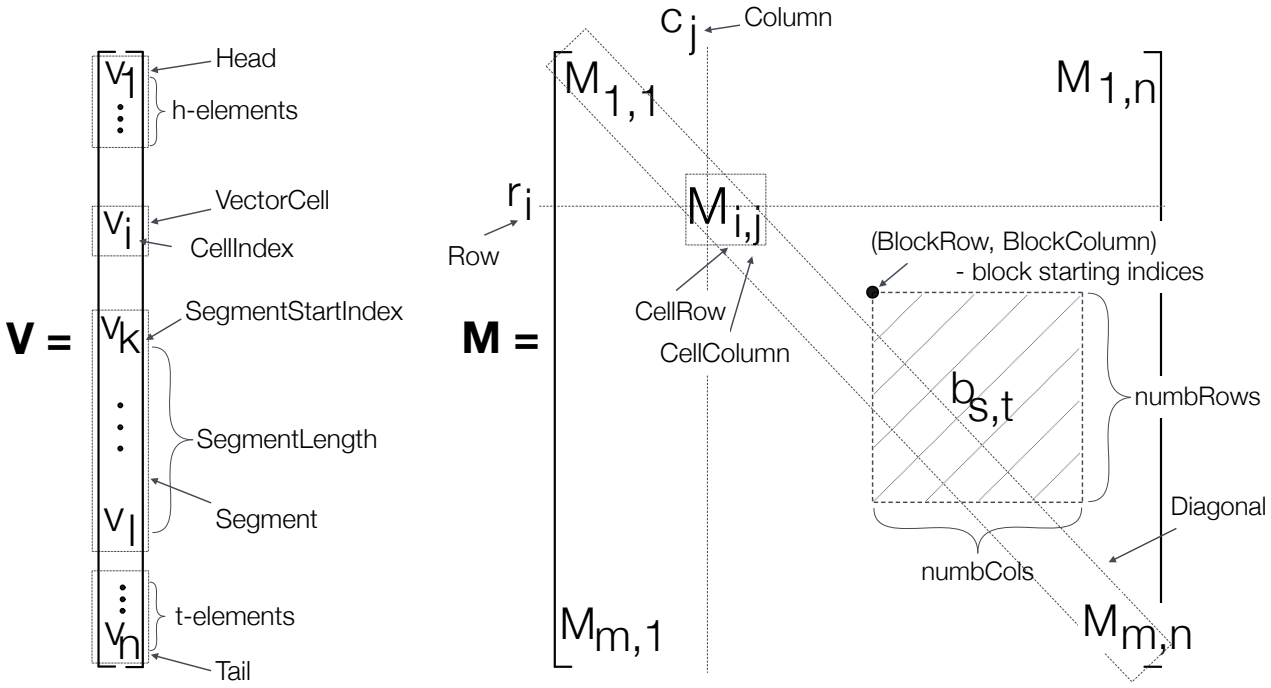


Figure 6.4: Vector and matrix – vocabulary and structure overview.

6.9.1 Vector structure with examples

The index for vector or matrix elements start with 1. Vectors are assumed to be column vectors. We provide both support for populating and reading vectors.

6.9.1.1 Populating vectors

The elements and attributes that are available to handle most common situation, schematically visualised in Figure 6.4 (left).

Table 6.3 shows two examples of how a sparse vector [0,2,0,0,5,0,0,0,0,0] can be coded in PharmML. A vector is either encoded explicitly with `<VectorElements>` using a mixture of numbers and symbol references or using the `<VectorCell>` elements (bottom right). The latter option, allowing to avoid unnecessary repetitions, requires the `default` attribute to define all remaining entires and `length` attribute to infer the vector length.

6.9.1.2 Reading vectors

Selecting elements out of a vector is very flexible and efficient. Once a vector is assigned, we need a mechanism for the readout of vector elements. This is done with the `<VectorSelector>` element. Let's consider following basic assignment operation

$$m = a + V2[5]$$

i.e. adding parameter a to the fifth element of vector $V2$. The following code shows how this is done.

```
<SimpleParameter symbId="m">
    <ct:Assign>
        <math:Equation>
            <math:Binop op="plus">
                <ct:SymbRef symbIdRef="a"/>
                <ct:VectorSelector>
                    <ct:SymbRef symbIdRef="V2"/>
                    <ct:Cell>
                        <ct:Int>5</ct:Int>
                    </ct:Cell>
                </ct:VectorSelector>
            </math:Binop>
        </math:Equation>
    </ct:Assign>
</SimpleParameter>
```

Additionally to the vocabulary we used before, we have now the `<Head>` and `<Tail>` methods, i.e. two ways to extract the first or last n elements which number can be specified explicitly. The following child elements allow flexible access to any vector element: `<SymbRef>` identifies the vector of interest (mandatory), `<Cell>` is used to pick one element, and `<Segment>` allows to select a segment of a vector.

Assume a vector V2 that has 13 elements and we want to select only those specified by the following indexes $i = \{1:3,\ 6:8,\ 10,\ 12:13\}$.

```
<SimpleParameter symbId="n">
    <ct:Assign>
        <math:Equation>
            <ct:VectorSelector>
                <ct:SymbRef symbIdRef="V2"/>
                <ct:Head><ct:Int>3</ct:Int></ct:Head>
                <ct:Segment>
                    <ct:StartIndex><ct:Int>6</ct:Int></ct:StartIndex>
                    <ct:SegmentLength><ct:Int>3</ct:Int></ct:SegmentLength>
                </ct:Segment>
                <ct:Cell><ct:Int>10</ct:Int></ct:Cell>
                <ct:Tail><ct:Int>2</ct:Int></ct:Tail>
            </ct:VectorSelector>
        </math:Equation>
    </ct:Assign>
</SimpleParameter>
```

Population of a vector using `<VectorElements>`

```
<SimpleParameter symbId="V1">
    <ct:Assign>
        <ct:Vector length="10">
            <ct:VectorElements>
                <ct:Real>0</ct:Real>
                <ct:Real>2</ct:Real>
                <ct:Real>0</ct:Real>
                <ct:Real>0</ct:Real>
                <ct:SymbRef symbIdRef="fifthElement"/>
                <ct:Real>0</ct:Real>
                <ct:Real>0</ct:Real>
                <ct:Real>0</ct:Real>
                <ct:Real>0</ct:Real>
                <ct:Real>0</ct:Real>
            </ct:VectorElements>
        </ct:Vector>
    </ct:Assign>
</SimpleParameter>
```

Populating a vector using `<VectorCell>` and attributes `default` and `length`

```
<SimpleParameter symbId="V2">
    <ct:Assign>
        <ct:Vector default="0" length="10">
            <ct:VectorCell>
                <ct:CellIndex>
                    <ct:Int>2</ct:Int>
                </ct:CellIndex>
                <ct:Real>2</ct:Real>
            </ct:VectorCell>
            <ct:VectorCell>
                <ct:CellIndex>
                    <ct:Int>5</ct:Int>
                </ct:CellIndex>
                <ct:SymbRef symbIdRef="fifthElement"/>
            </ct:VectorCell>
        </ct:Vector>
    </ct:Assign>
</SimpleParameter>
```

Table 6.3: Comparison of two encoding types of the vector [0,2,0,0,5,0,0,0,0,0]. The vector is either encoded explicitly with `<VectorElements>` using a mixture of numbers and symbol references (left) or using the `<VectorCell>` elements (right). In the latter case the use of attributes `default` and `length` is required, otherwise the dimension of the vector could not be estimated.

6.9.2 Matrix structure with examples

Similarly to the vector structure there are a few alternative options to encode a matrix. Note, that for now matrices will be mainly be used for encoding of the correlation structure between random effect, i.e. within the `<Correlation>` element. So mentioned above, the associated SO requires matrix support as well. The use of matrices will be extended to handle covariance matrices of multivariate distributions and other model elements in a future release. The index for matrix elements start with 1.

6.9.2.1 Populating matrices

PharmML provides multiple ways to populate matrices. Following matrix, Σ can be represented as a full matrix or sparse matrix.

$$\Sigma = \begin{bmatrix} 1 & \Sigma_{12} & 0 & 0 \\ 0 & 1 & \Sigma_{23} & 0 \\ \Sigma_{31} & 0 & 1 & 0 \\ 0 & 0 & 0 & \Sigma_{44} \end{bmatrix}$$

6.9.2.2 Implementation of Σ as full matrix

This is a straightforward implementation of every matrix element explicitly using the `<MatrixRow>` elements, which is used to define the matrix row-by-row.

```
<Correlation deviationMatrixType="CovMatrix">
    <ct:VariabilityReference>
        <ct:SymbRef symbIdRef="indiv"/>
    </ct:VariabilityReference>
    <Matrix matrixType="Any">
        <ct:MatrixRow>
            <ct:RowIndex><ct:Int>1</ct:Int></ct:RowIndex>
            <ct:Real>1</ct:Real>
            <ct:SymbRef symbIdRef="Sigma12"/>
            <ct:Real>0</ct:Real>
            <ct:Real>0</ct:Real>
        </ct:MatrixRow>
        <!-- omitted rows 2 & 3 -->
        <ct:MatrixRow>
            <ct:RowIndex><ct:Int>4</ct:Int></ct:RowIndex>
            <ct:Real>0</ct:Real>
            <ct:Real>0</ct:Real>
            <ct:Real>0</ct:Real>
            <ct:SymbRef symbIdRef="Sigma44"/>
        </ct:MatrixRow>
    </Matrix>
</Correlation>
```

6.9.2.3 Implementation of Σ as sparse matrix

Here the Σ matrix is implemented using `<MatrixCell>`, which is useful for matrices with a few non-zero elements. Only four values have to be stored while the rest is encoded using `diagDefault` and `offDiagDefault` attributes.

```
<Correlation deviationMatrixType="CovMatrix">
    <ct:VariabilityReference>
        <ct:SymbRef symbIdRef="indiv"/>
    </ct:VariabilityReference>
    <Matrix matrixType="Any" diagDefault="1" offDiagDefault="0">
        <ct:MatrixCell>
            <ct:CellRow><ct:Int>1</ct:Int></ct:CellRow>
            <ct:CellColumn><ct:Int>2</ct:Int></ct:CellColumn>
            <ct:SymbRef symbIdRef="Sigma12"/>
        </ct:MatrixCell>
        <!-- omitted MatrixCell's in rows 2 & 3 -->
        <ct:MatrixCell>
            <ct:CellRow><ct:Int>4</ct:Int></ct:CellRow>
            <ct:CellColumn><ct:Int>4</ct:Int></ct:CellColumn>
            <ct:SymbRef symbIdRef="Sigma44"/>
        </ct:MatrixCell>
```

```
    </Matrix>
</Correlation>
```

Note that the attributes `diagDefault` and `offDiagDefault` are set to 1 or 0, respectively, which makes the matrix encoding very efficient.

6.9.2.4 Example 1 – Covariance matrix with expressions

The elements of a matrix can contain arbitrary expressions. For example, the following covariance matrix

$$\Sigma = \begin{bmatrix} \sigma_V^2 & \rho\, \sigma_V \sigma_k \\ \rho\, \sigma_V \sigma_k & \sigma_V^2 \end{bmatrix}$$

is easily implemented as the following code shows

```
<Correlation deviationMatrixType="CovMatrix">
    <ct:VariabilityReference>
        <ct:SymbRef blkIdRef="modelVar" symbIdRef="indiv"/>
    </ct:VariabilityReference>
    <Matrix matrixType="Any">
        <ct:MatrixRow>
            <ct:RowIndex><ct:Int>1</ct:Int></ct:RowIndex>
            <math:Equation>
                <math:Binop op="power">
                    <ct:SymbRef symbIdRef="sigma_V"/>
                    <ct:Real>2</ct:Real>
                </math:Binop>
            </math:Equation>
            <math:Equation>
                <math:Binop op="times">
                    <ct:SymbRef symbIdRef="rho"/>
                    <math:Binop op="times">
                        <ct:SymbRef symbIdRef="sigma_V"/>
                        <ct:SymbRef symbIdRef="sigma_k"/>
                    </math:Binop>
                </math:Binop>
            </math:Equation>
        </ct:MatrixRow>
        <!-- omitted 2nd row -->
    </Matrix>
</Correlation>
```

6.9.2.5 Reading matrices

Once a matrix is assigned, we need a mechanism for the readout of its elements. This is done with the `<MatrixSelector>` element. Following code shows few examples of how to access single element or row from the Σ matrix defined in previous example.

```
<!-- extract one element -->
<SimpleParameter symbId="rho_tlag_ka">
    <ct:Assign>
        <math:Equation>
            <ct:MatrixSelector>
                <ct:SymbRef symbIdRef="Omega"/>
                <ct:Cell>
                    <ct:RowIndex><ct:Int>1</ct:Int></ct:RowIndex>
                    <ct:ColumnIndex><ct:Int>2</ct:Int></ct:ColumnIndex>
                </ct:Cell>
            </ct:MatrixSelector>
        </math:Equation>
    </ct:Assign>
</SimpleParameter>

<!-- extract 2nd row -->
<ct:Variable symbolType="real" symbId="SecondRow">
    <ct:Assign>
        <math:Equation>
            <ct:MatrixSelector>
                <ct:SymbRef symbIdRef="Omega"/>
                <ct:Row>
```

```
                    <ct:Int>2</ct:Int>
                </ct:Row>
            </ct:MatrixSelector>
        </math:Equation>
    </ct:Assign>
</ct:Variable>
```

6.10 Representing statistics

As can be seen in chapter 3, PharmML relies on the ability to use probability distributions to describe the variability in a pharmacometric model. Admittedly, the most commonly used distribution is the normal distribution (or transformations of it), but our intention is that PharmML will have the flexibility to describe a wide range of probability distribution types. To do this the languages uses UncertML version 3[11], which is an XML Schema based language that aims to "describe and exchange uncertainty". UncertML supports all the commonly used continuous and discrete probability distributions and so more than adequately supports the needs of PharmML. As an example we can show how the normal distribution $\mathcal{N}(0, \omega^2)$ can be defined in UncertML.

```
<NormalDistribution xmlns="http://www.uncertml.org/3.0"
    definition="http://www.uncertml.org/distributions/normal">
    <mean>
        <rVal>0</rVal>
    </mean>
    <stddev>
        <var varId="omega"/>
    </stddev>
</NormalDistribution>
```

6.11 Time

Time is required by most pharmacometric models and in PharmML it can be referred to explicitly. The symbol used for the time variable is configurable at the beginning of the document as shown below:

```
<PharmML xmlns="http://www.pharmml.org/pharmml/0.6/PharmML"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.pharmml.org/pharmml/0.6/PharmML ..."
    xmlns:math="http://www.pharmml.org/pharmml/0.6/Maths"
    xmlns:ct="http://www.pharmml.org/pharmml/0.6/CommonTypes"
    xmlns:ds="http://www.pharmml.org/pharmml/0.6/Dataset"
    xmlns:design="http://www.pharmml.org/pharmml/0.6/TrialDesign"
    writtenVersion="0.6">
    <ct:Name>IOV1 with covariates</ct:Name>
    <IndependentVariable symbId="t"/>
```

Rather than time we call the element `<IndependentVariable>` because this is more correct and because you could define a model that uses another quantity than time as the independent variable (e.g. dose in a dose-response model). Note that the independent variable always has a real type.

6.12 Element identifier

To enable other resources to refer to specific pieces of information in a PharmML document, every XML element in the document with an optional unique identifier is provided. The listing below shows how the `id` attribute is used:

```
<FixedEffect id="e10" symbId="beta_V">
    <Covariate>
        <ct:SymbRef symbIdRef="W"/>
    </Covariate>
</FixedEffect>
<FixedEffect id="e13" symbIdRef="beta_Cl">
    <Covariate>
        <ct:SymbRef symbIdRef="W"/>
```

[11] http://www.uncertml.org

```
    </Covariate>
</FixedEffect>
<RandomVariable id="e16" symbId="eta_V">
    <ct:VariabilityReference id="e17">
        <ct:SymbRef id="e18" blkIdRef="model" symbIdRef="level"/>
    </ct:VariabilityReference>
    <!-- Snip -->
</RandomVariable>
```

The `id` attribute is optional and need only be used for elements that you want to refer to. There are no rules other than that the identifier must be unique within the PharmML document. Thus to refer to a specific part of a document all you need to define is the location of the document and the identifier. So assuming that the above code snippet is found in a file called "testFile.xml" a suitable (relative) URI that refers to the random variable in the example might be `testFile.xml#e16`.

The benefit of this approach is that the `id` attribute can easily be made available programmatically and searched on with a class library generated from the XML Schema. This mechanism is also used successfully by SBML [Hucka et al., 2010, Section 6]. You will see in the sections below (sections 6.14.1 and 6.14.2) how we take advantage of this mechanism to annotate and extend a PharmML document.

6.13 Ordering modelling steps

Descriptions in a modelling step are declarative, describing what was done, what algorithms were used and what their properties were.

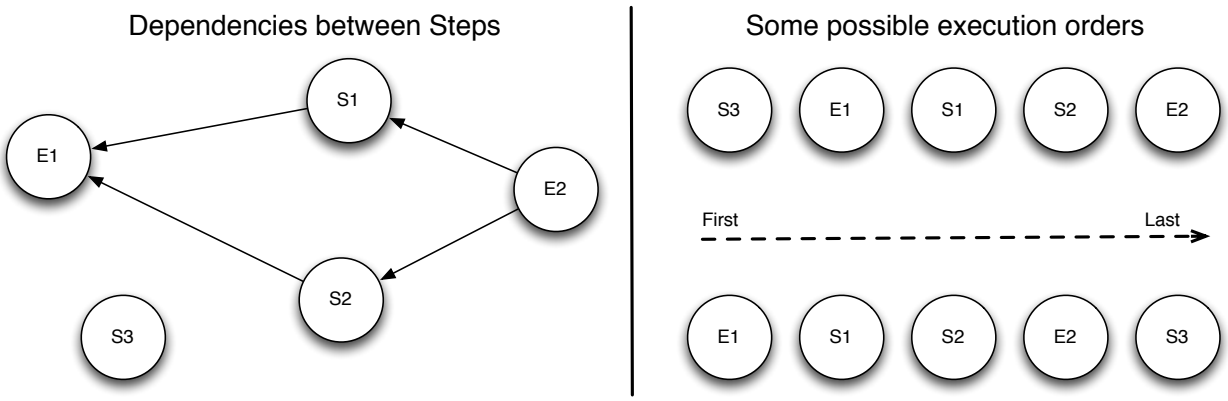


Figure 6.5: An example of the dependencies between modelling steps is shown on the left. The arrow indicates the direction of the dependency so step S1 is dependent on the successful completion of step E1. So in this example step E1 must be executed before step S1 and S2, and those steps must both execute before step E2. On the right-hand figure we show two possible execution orders that correspond to the task dependecies on the left. It is important to remember that, as in this example, there can be more than one execution order for a given set of task dependencies.

The *Modelling Steps* section has two components that describe simulation or estimation tasks. Multiple tasks can be linked together so that a given estimation task can be placed before a given simulation task or no order can be given in which case tasks can be executed in parallel or in any order. A task is ordered by defining its dependent tasks: tasks that must complete successfully before it can start (see figure 6.5). In this way the modelling steps make no assumptions about how or where its tasks are executed, but provides enough information for a workflow engine to parallelise the execution of its tasks successfully, should it wish to.

At the moment tasks cannot specify how output is generated from either an Estimation or Simulation step. A result of this restriction is that it is *not* possible to exchange information from one modelling step to another: for example to use the output of an estimation to set the parameter values in an estimation. We recognise that this is a limitation of the current version and this functionality will be provided in a future release of PharmML.

6.14 Supporting Resources

6.14.1 Metadata: annotating the PharmML document

As has been stated in Chapter 2, the purpose of the PharmML document is to provide a mathematical and structural description of a pharmacometric model, sufficient for it to be executed. Additional information, such as a description of the disease process being modelled, the exact estimation algorithm or a publication describing the model is not included in the PharmML document. Typically called metadata, this information is very important and PharmML provides support for external annotation.

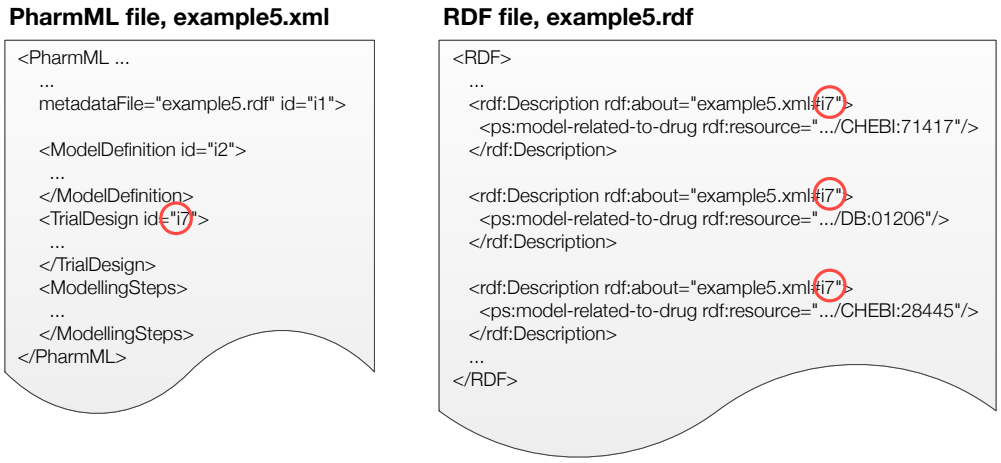


Figure 6.6: On schematic overview of the PharmML, *example5.xml* and RDF, *example5.rdf*, files relationships. Note, that we use here, for the sake of clarity, a simplified RFD statements. See the code snippet at the end of this section for the correctly formatted RDF file [RDF Working Group, 2014].

We expect such metadata comes in the form of ontological annotation and while the detail of what will be annotated is out of scope of this document it is important to stress that every PharmML element can be annotated. This is done by adding an attribute, e.g. `id="i7"`, to the `<TrialDesign>` element, see Figure 6.6.[12] The following example gives you a flavour of what an RDF annotation of a PharmML document would look like. The metadata above is encoded in RDF-XML [de Bono et al., 2011][RDF Working Group, 2014], and annotates 3 external terms; CHEBI:71417, DB01206, and CHEBI:28445.

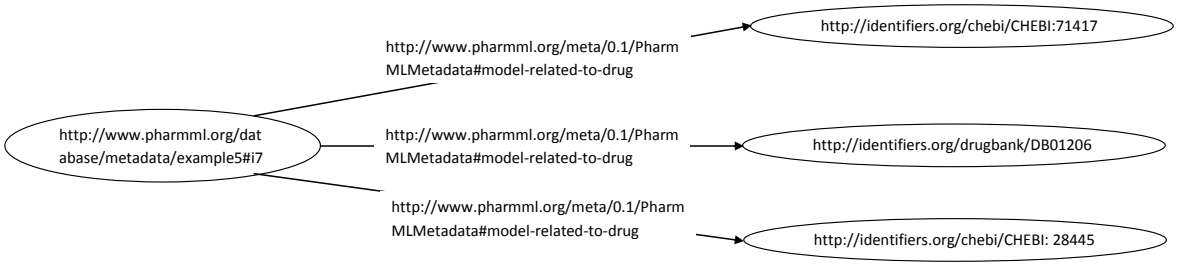


Figure 6.7: This figure illustrates how information in the RDF example is organised. Simply put the subject of the annotation is on the left and is annotated by the object on the right. The meaning of that annotation is described by the *predicate* term labelling the arrow. So reading the top subject from left to right we understand that element assigned the attribute `i7` is *model-related-to-drug* at *CHEBI:71417*.

This is a basic example, but it illustrates how the PharmML document can be annotated using metadata described in a separate RDF document

```
<rdf:RDF
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
```

[12] More information on annotating PharmML models are described in the PharmML Metadata Specification.

```
    xmlns:base="http://www.pharmml.org/database/metadata/example5#"
    xmlns:example5="http://www.pharmml.org/database/metadata/example5#"
    xmlns:ps="http://www.pharmml.org/meta/0.1/PharmMLMetadata#">

    <rdf:Description rdf:about="#i7">
        <ps:model-related-to-drug rdf:resource="http://identifiers.org/chebi/CHEBI:71417"/>
        <ps:model-related-to-drug rdf:resource="http://identifiers.org/drugbank/DB01206"/>
        <ps:model-related-to-drug rdf:resource="http://identifiers.org/chebi/CHEBI:28445"/>
    </rdf:Description>
</rdf:RDF>
```

6.14.2 Extending PharmML

As with any standard there will be circumstances when it does not represent all the information that you would like and it would be convenient to extend it. Typically there are two scenarios where this is likely to be the case. The first is when the information is genuinely not supported by PharmML. This may be because it has not been implemented yet, or it may be that there is no consensus about whether this information should be included or no agreement about the best way to represent it. The second scenario is when you want to add application specific information to a PharmML document. Perhaps because a tool wishes to use PharmML as its native storage format in which case it would also want to store information about application settings etc.

Whatever the reason PharmML can be extended. Like the metadata descriptions above (section 6.14.1) this approach relies on the element identifier (see section 6.12). The recommended approach is that you develop a separate XML document (typically in a separate file), which we will call the extension document, using any XML representation you choose. Where information in the extension document relates to the content of the PharmML document then you can refer to the relevant XML element using its identifier.

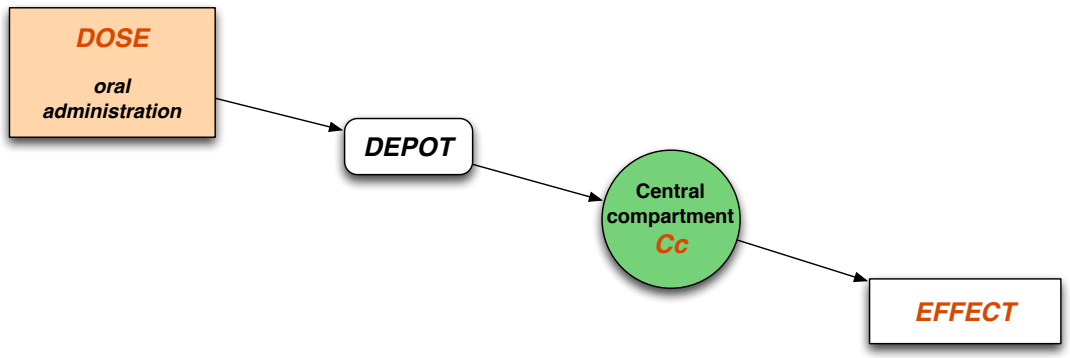


Figure 6.8: This diagram is reproduced from an example in the Monolix user manual [Lixoft, 2012] and it provides a graphical description of a structural model. In the hypothetical example in the text we illustrate how you might extend PharmML to link this graphic with the components of the model it represents.

We can illustrate with an example based on the second scenario we described above: application specific extensions. In this scenario our software tool provides a graphical interface that lets you create a pharmacometric model by drawing and connecting shapes such as the diagram in figure 6.8. When you save the diagram the application saves both the PharmML that encodes the model and an extension document that describes the graphical layout and maps the graphical elements to the relevant parts of the model. The extension document could look like this:

```
<Diagram resource="file:///anotherPharmMLFile.xml">
    <Rectangle id="1">
        <Name>Dose</Name>
        <Bounds x="10" y="10" w="50" h="30"/>
        <!-- Omitted other information such as colour and other text-->
        <Ref idRef="e35"/>
    </Rectangle>
    <!-- Omitted -->
    <Circle id="3">
        <Name>Central</Name>
        <Bounds x="10" y="200" w="45" h="45"/>
        <!-- Omitted other information such as colour and other text-->
```

```
        <Ref idRef="e46"/>
    </Circle>
    <!-- Omitted other shape definitions-->
    <Link src="1" tgt="2"/>
    <Link src="2" tgt="3"/>
    <Link src="3" tgt="4"/>
</Diagram>
```

The XML describes the shapes of the nodes, their location and the connections between them. What allows it to extend the PharmML document? First the `resource` attribute provides a URL describing the location of the PharmML document being extended. Next the `<Ref>` element defines a reference that points to an element in the PharmML document. From this information the application is able to read both the PharmML and the extension documents and then relate the diagram to the relevant part of the model.

Note that while this is a hypothetical example for PharmML, this type of solution has been implemented by a graphical Systems Biology editor called CellDesigner[13]. It uses SBML as its native application format: encoding the model in SBML and using SBML's extension facilities to store graphical information and other application specific properties.

One final clarification. The extension mechanism does not require an application to use the same XML elements as described in the example above. The XML content of the extension document is entirely the concern of the application. In order to extend a PharmML document all it must do is:

1. Specify how to find the PharmML document. Using a URI is a good way to do this.
2. Use the element identifier to refer to the content of the PharmML document.

6.14.3 Organising PharmML resources

We expect that PharmML will in normal usage consist of more than one file. From the discussion about annotating (section 6.14.1) and extending (section 6.14.2) PharmML it is clear that both of these cases require the creation of resources that are closely related to a PharmML document. It is therefore desirable that they are kept together and exchanged and used together.

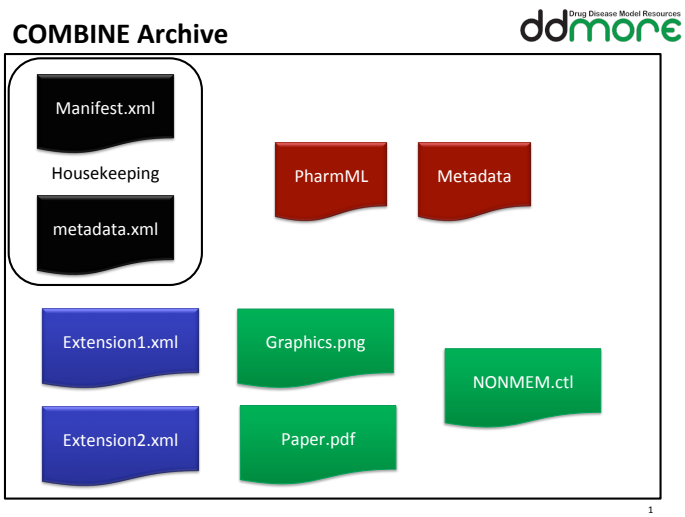


Figure 6.9: An overview of how a COMBINE archive may be used to hold XML documents and files associated with a model encoded in PharmML. In this example the archive holds the model and its metadata (in red) and two application specific extension documents (in blue). This also shows another advantage of using the archive: you can store other useful information, such as the original model file, relevant papers or images (in green). Finally, the archive contains a metadata file (in black) that helps an application reading the archive make sense of what it contains.

An archive file can provide this functionality. It acts as a container, and since it is a file can be easily exchanged and stored. We recommend that you use an archive based on the emerging COMBINE Archive

[13] `http://www.celldesigner.org`

standard[14]. It is based on a zip archive and holds additional information that allows you to identify the modelling resources in the file. The exact details of how it works is outside the scope of this document, but the components of the archive and how it can be used to hold PharmML related resources is illustrated in figure 6.9

6.14.4 Software support for PharmML

PharmML is a complex language. It is designed using an industry standard, XML Schema[15], which gives us the ability to use widely available software packages to verify that the XML file is correctly written and that elements are put in the right place (syntax checking). However, PharmML describes a pharmacometric model and so there is a lot of information that is very specific to this domain and cannot be validated by standard tools. That's why we need PharmML specific software tools and libraries. Without such software the burden of validation falls on the modelling tools reading and writing PharmML. Given the complexity of PharmML's validation rules it is unlikely that such validation would be complete and implemented consistently, which makes our goal of exchange between modelling tools less likely to succeed.

Therefore in parallel to the development of this specification a software library called libPharmML is being developed to support it. This will allow you do the following:

1. Create a new PharmML document.
2. Read an existing PharmML document from a file (or other resource).
3. Write a PharmML document to a file (or resource).
4. Validate that a PharmML document complies with the XML Schema definition and the rules set out in this specification.

The library is implemented in Java. There are no plans to implement an equivalent version in another programming language, such as C++, but this could be done if there was sufficient demand for it and sufficient developer resources were available to implement it.

[14] `http://co.mbine.org/documents/archive`
[15] `http://www.w3.org/XML/Schema`

Chapter 7

The XML Schema Definition

7.1 How is PharmML Defined?

The normative definition of PharmML consists of two parts. First is the XML Schema definition. This describes the syntax of the XML document and defines some semantic rules such as constraints on permitted attribute values (enumerations) and the uniqueness of some identifiers. In order to be valid PharmML, an XML document must conform to this schema definition. The XML Schema definition files are available with this specification and should be regarded as the definitive authority. A documented form of the schema definition can be downloaded from the PharmML related websites `http://ddmore.eu/pharmml` and `http://pharmml.org`.

Most of the semantic rules in PharmML are *not* captured by the XML Schema definition, so the second part of the normative definition is provided by the rules in chapter 8. The rules are enumerated to allow validating tools to conveniently refer to them and also to make them clear. The rules specified here are definitive and take precedence over sources such as software implementations.

7.2 Design Guidelines

In designing the XML Schema definition of PharmML we adopted a number of design and naming conventions. These were based on a number of best practise recommendations[1,2] and we have tried to be consistent in their application. Unless specifically documented a deviation from these guidelines is an error on our part.

7.2.1 XML Schema Compliance

PharmML is defined using version 1.0 of the XML Schema[3].

7.2.2 Naming Conventions

Obviously the XML Schema has rules about how names can be defined, but on top of these rules we have adopted the following conventions:

- Names should, wherever possible be, be descriptive of the named component's purpose, and acronyms should be avoided.
- Names should be in English with British English spellings.
- Names should not be excessively long. Especially names used very frequently as they may clutter the XML and unnecessarily bloat the size of the resulting XML documents.
- Element names should be capitalised and use camel case to delineate words.
- Attribute names should start with a lowercase character and use camel case to delineate words.
- Enumerations should follow the convention used for attributes.

[1] `http://alturl.com/jdmkn`
[2] `http://www.xml.com/pub/a/2002/11/20/schemas.html`
[3] `http://www.w3.org/TR/xmlschema-1/`

7.2.3 Design Pattern

We adopted the Venetian Blind design pattern[4] for PharmML. In this pattern all non-global XML Elements are defined using a complex type. Complex types were inherited using extension rather than restriction. Typically we tried to reduce the number of global XML Elements to make identification of the top level element easier (in cases where there was a single preferred top-level element).

The benefit of this approach is that it maximises reuse and extension of the schema by allowing other schemas to reuse or extend the complex types. This is at the expense of cluttering the design with a surfeit of complex type definitions.

7.2.4 Namespaces

The domain name `pharmml.org` has been reserved for use by PharmML. Consequently we use this domain name in all the namespaces associated with PharmML.

There are a number of possible strategies for defining namespaces[5] and no definitive best practise. Following the practice used in SBML or PMML, and taking into account that we need three meaningful namespaces for PharmML, SO and metadata, we use the following form:

- for PharmML – `http://www.pharmml.org/pharmml/x.y/Resource`
- for SO – `http://www.pharmml.org/so/x.y/Resource`
- for metadata – `http://www.pharmml.org/meta/x.y/Resource`

with x for major version and y for minor version. *Resource* provides a short, but descriptive name of the purpose of the schema. See also Table 7.1.

7.2.5 Elements

Elements should always be qualified by a namespace. This corresponds to the XML Schema declaration: `elementFormDefault="qualified"`.

7.2.6 Attributes

Default attributes effectively change the structure of the DOM from that described by the XML itself. This can cause problems with validation and result in difficult to track errors. For these reasons the use of default attribute values is forbidden.

Attributes should never be qualified by a namespace[6]. This corresponds to the XML Schema declaration `attributeFormDefault="unqualified"`.

7.2.7 Elements vs. Attributes

It is a common dilema in design XML documents: when do I use an element and when an attribute? In this schema design we have tried to follow the advice provided in an IBM technical document[7]. The advice can be summarised as follows:

Principle of core content Briefly data or core content should be held in elements and metadata should be in attributes.

Principle of structured information If the information needs to be structured then it should be represented by an element. If it is atomic then us an attribute.

Principle of readability If the information is intended to be read and understood by a person then use elements. If it is intended to be used by a machine then use attributes.

Principle of element/attribute binding If the information to be represented can be modified by another attribute then use an element to represent it.

[4] `http://www.oracle.com/technetwork/java/design-patterns-142138.html`
[5] `http://www.ibm.com/developerworks/library/x-namcar/index.html`
[6] `http://www.xml.com/pub/a/2002/11/20/schemas.html?page=3`
[7] `http://http://www.ibm.com/developerworks/xml/library/x-eleatt/index.html`

File Name	Namespace URI	Description
	PharmML	
pharmml.xsd	http://www.pharmml.org/pharmml/0.6/PharmML	The overall PharmML definition including all the other components.
modelDefinition.xsd	http://www.pharmml.org/pharmml/0.6/ModelDefinition	Defines the model definition section.
trialDesign.xsd	http://www.pharmml.org/pharmml/0.6/TrialDesign	Defines the trial design section.
modellingSteps.xsd	http://www.pharmml.org/pharmml/0.6/ModellingSteps	Defines the modelling steps section.
commonTypes.xsd	http://www.pharmml.org/pharmml/0.6/CommonTypes	Defines the types and elements common to the all schema definitions.
dataset.xsd	http://www.pharmml.org/pharmml/0.6/Dataset	Defines the dataset and structures used in the trial design and modelling steps to represent tabular data.
maths.xsd	http://www.pharmml.org/pharmml/0.6/Maths	Defines the representation of mathematical expressions and statements.
UncertML30.xsd	http://www.uncertml.org/3.0	Defines the probability distributions.
	SO	
standardisedOutput.xsd	http://www.pharmml.org/so/0.1/StandardisedOutput	The definition of the Standardised Output format.

Table 7.1: Namespace URI overview for PharmML (with UncertML) and SO.

These rules are to some extent a matter of judgement and in some cases there are marginal cases. The names of parameters, variables and other symbols used in PharmML are a case in point. The names for such symbols have meaning for a modeller, but they are also used computationally. In addition, variable names can have to forms, a computer friendly form such as `omega_V` and a mathematical form such as ω_V. Currently PharmML only handles the latter form, and although this is a computational name it is also commonly used by modellers. Our solution has been to treat the symbol name as an element that can in the future be extended to handle a mathematical form of the variable, but with the computation name given as an attribute. For example:

```
<Variable symbID="pV" symbolType="scalar" >
  <ct:Symbol>pop_V</Symbol>
</Variable>
```

In the above case the `<ct:Symbol>` element is optional so it is only provided if the displayed name of the symbol is required.

7.2.8 Keys and Key References

We use the XML Schema key and keyref mechanisms only. ID and IDREF types should not be used[8].

7.2.9 Versioning Strategy

PharmML will change over time and indeed we have described the versioning strategy for PharmML earlier in this document (section 1.8 on page 12). While this makes clear how we anticipate the specification document to change, what we also need to accommodate are changes to the XML Schema definition that such change implies.

Unfortunately there is no definitive solution for versioning XML Schema definitions, but we are following the conventions described elsewhere[9]. In particular we will do the following:

1. Use the `version` attribute in the XML Schema definition to define the current version of the schema.
2. Add a version attribute in the top-most elements of the instance document indicating what version they were compliant with, when they were last updated.
3. Rely on the PharmML validator to ensure that the version of the instance document and XML Schema definition are compatible.

[8] http://www.xml.com/pub/a/2002/11/20/schemas.html?page=3#identity_constraints
[9] http://www.xfront.com/Versioning.pdf

4. The namespace URL will only change if there is a significant change in the symbols defined in the namespace or if the meaning of a significant proportion has been redefined.

7.3 XML Schema Organisation

PharmML is a large language and the XML Schema definition is correspondingly large too. The language is naturally organised into three sections (see chapter 6): Model Definition, Trial Design, and Modelling Steps, which provides a convenient way to modularise the XML Schema definition. In addition, we have two components, which are also naturally independent of the core PharmML specification: the definition of both mathematical expressions and probability distributions. Using these natural divisions, we split the PharmML XML Schema definition into the a number of xsd files, see Table 7.1.

Note that a PharmML document must be compliant with the pharmml.xsd definition, which includes all the other components. The other schema definitions should not be used independently to validate a PharmML document.

Chapter 8

Validation of PharmML

8.1 Introduction

In this section we provide detailed rules about what constitutes a valid PharmML document. Where possible we have tried to keep each rule definition discrete and also we have provided a unique identifier for such rules. We recommend that developers implementing support for PharmML validation report such rule identifiers in their error messages. Users can then cross-reference such errors with this specification if they require more detailed information.

The rules are organised so that we cover the basic language features and constructs first and then go into specific rules for each of the sections of a PharmML document: Model Definition, Trial Design and Modelling Steps.

8.2 Namespaces and Scopes

8.2.1 Defining Symbols and Objects

The namespaces and scopes used in PharmML are shown in figure 8.1. By namespace we mean a dictionary of names, in which each name must be unique within its given scope. As you can see from the figure there are two namespaces, one which defines the symbols used to describe the model (for more background information read section 6.3.2) and the other (namespace Element) is used to allow the PharmML document to be cross references externally (see section 6.12). The symbols can be classified as follows:

Independent Variable A special variable that defines the independent variable used throughout the model.

Function Definition A function that can be reused throughout the PharmML document.

Function Argument The parameters of the function. Their scope extends into the body of the function. For example: $f(x) = x + 1$.

Object An identifier used to uniquely identify conceptual objects within the PharmML document.

Block An identifier that defines a model within the Model Definition section of PharmML. This provides a scope for symbols defined within the block and gives the model definition a degree of modularity.

Variability Level A symbol that defines a level of random variability.

Covariate A symbol that defines variability associated with an individual. It can be continuous or categorical. In the latter case categories are scoped by the covariate symbol.

Category A category of a categorical covariate.

Simple Parameter A parameter that cannot be assigned a random variable.

Individual Parameter A parameter that can be assigned a random variable.

Random Variable A special parameter than is described by a probability distribution.

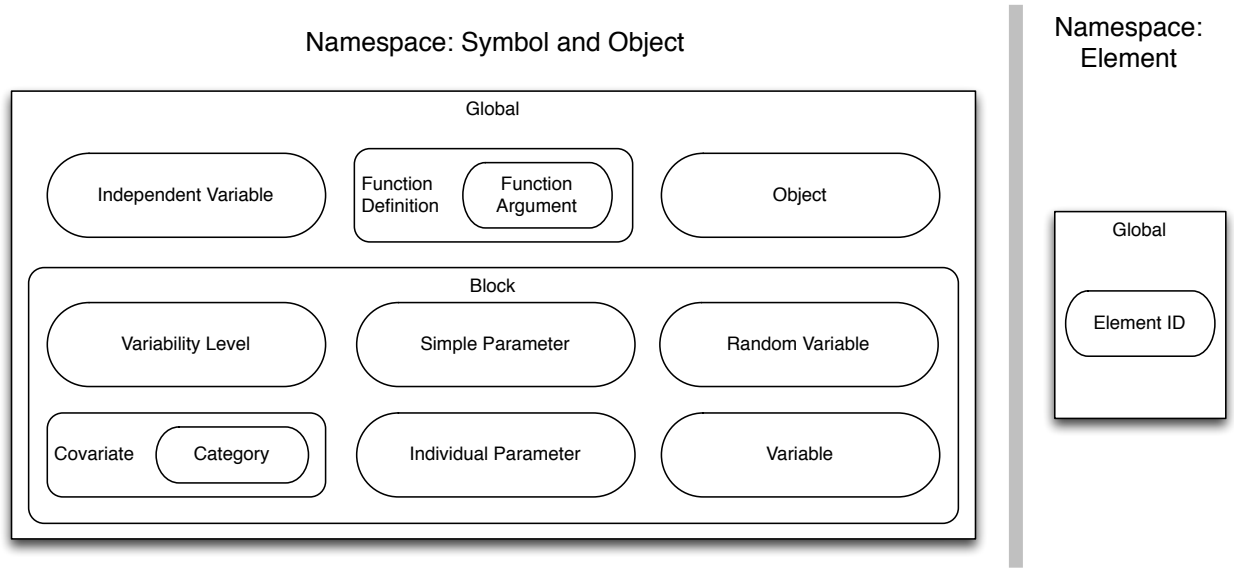


Figure 8.1: An overview of the scopes and namespaces used in PharmML. The class of symbols within the scope are shown as ovals, symbols that also define a scope are rounded rectangles and the global scope is shown as rectangles. So for example, the function argument is a class of symbol that is scoped by the function definition which in turn belongs to the global scope.

Variable A variable in the model. This is distinguished from a parameter in that it can change over time, while a parameter cannot.

Element ID An identifier used by external resources to identify a specific element within the PharmML document.

Using these concepts we can apply the following rule:

S1 *Symbols must be unique with their scopes.* Duplicate symbols are not permitted within a given namespace.

8.2.2 Symbol Resolution

Symbols must be resolved using the scoping rules. This is described in detail in section 6.3.2. Symbols are typically referred to using the `<SymbRef>` element and objects by the `<OidRef>` element or XML elements of type OidRefType. Resolution rules are:

S2 *References to symbols and objects must resolved.* Dangling references are not permitted.

S3 *The resolved symbol must be compatible with the PharmML component referencing it.* This means that an `<ArmRef>` which should match an arm definition should not point to an Epoch definition. Compatibility is defined in the table 8.1.

S4 *A `<SymbRef>` element must only reference symbols that are compatible with its parent element.* Compatibility is defined by table 8.1.

S5 *A `<OidRef>` element or element using the type `OidRefType` must only reference objects that are compatible with its parent element.* Compatibility is defined by table 8.2.

Reference Parent	Target
VariableAssignment	SimpleParameter, CovariateModel/Covariate, RandomVariable IndividualParameter, Variable, DerivativeVariable
ParentLevel	VariabilityModel/Level (The correct target is also affected by rule M6.)

PopulationParameter	SimpleParameter
LinearCovariate/Covariate	CovariateModel/Covariate
FixedEffect	SimpleParameter
GeneralCovariate	SimpleParameter, CovariateModel/Covariate
GaussianModel/RandomEffects	RandomVariable
IndividualParameter/Assign	SimpleParameter, CovariateModel/Covariate, RandomVariable, IndividualParameter
VariabilityReference	VariabilityModel/Level
SimpleParameter	SimpleParameter
RandomVariable1	RandomVariable
RandomVariable2	RandomVariable
CorrelationCoefficient	SimpleParameter
Covariance	SimpleParameter
Variable	SimpleParameter, CovariateModel/Covariate, RandomVariable, IndividualParameter, Variable, DerivativeVariable
DerivativeVariable/Assign	SimpleParameter, CovariateModel/Covariate, RandomVariable, IndividualParameter, Variable, DerivativeVariable
DerivativeVariable/IndependentVariable	Variable
InitialCondition	SimpleParameter, CovariateModel/Covariate, RandomVariable, IndividualParameter,Variable,DerivativeVariable
ObservationModel/General	SimpleParameter, CovariateModel/Covariate, RandomVariable, IndividualParameter
ObservationModel/Standard/Output	Variable, DerivativeVariable
ErrorModel	FunctionDefinition, SimpleParameter, IndividualParameter, RandomVariable
RandomError	RandomVariable
DoseAmount	Variable, DerivativeVariable[1]
SteadyState/EndTime	Variable, DerivativeVariable
SteadyState/Interval	Variable, DerivativeVariable
DosingTimes	Variable, DerivativeVariable
Duration	Variable, DerivativeVariable
Rate	Variable, DerivativeVariable
CovariateMapping	Covariate
SimulationStep/Observations/Continuous	Variable/DerivativeVariable
ParameterEstimation	SimpleParameter, IndividualParameter, Covariate
InitialEstimate	Variable, DerivativeVariable, FunctionDefinition, SimpleParameter, IndividualParameter, RandomVariable
LowerBound	Variable, DerivativeVariable, FunctionDefinition, SimpleParameter, IndividualParameter, RandomVariable
UpperBound	Variable, DerivativeVariable, FunctionDefinition, SimpleParameter, IndividualParameter, RandomVariable

Table 8.1: This table describes the compatibility of symbol references defined using `<SymbRef>`. The comparability is with the parent elements that use the `<SymbRef>` to refer to other symbols within the PharmML document. In the table the Reference Parent column describes the element which is the immediate parent of the `<SymbRef>` element. The target column specifies the set of elements that can be the target of this reference. Where the parent element is required to identify the correct element a 'path' is indicated using the '/' symbol.

[1]The choice of valid target is governed by rules D11-13.

Reference Parent	Target
EpochRef	Epoch
ArmRef	Arm
SegmentRef	Segment
ActivityRef	Activity
DemographicMapping	Demographic
Step	SimulationStep, EstimationStep
Dependents	SimulationStep, EstimationStep

Table 8.2: This table describes the compatibility of object references defined using `<OidRef>` or from elements of type `OidRefType`. The comparability is with the parent elements that use the above elements to refer to objects within the PharmML document. In the table the Reference Parent column describes the element which is the immediate parent of the reference element. The target column specifies the set of elements that can be the target of the reference. Where the parent element is required to identify the correct element a 'path' is indicated using the '/' symbol.

8.3 Type System

8.3.1 Types

PharmML has the types in the table 8.3. Some types can be automatically converted (promoted) to another type. The rules are described below, with detailed information provided in the specified tables.

S6 *PharmML has a type system and all symbols and elements, if they have a type must conform to it.* The types are specified in table 8.3.

S7 *Symbol classes have a type.* The types are specified in table 8.4.

S8 *Elements, that are explicitly typed must be associated with quantities of same type* Quantities associated elements in a PharmML document, must be of the same type. The type of the relevant elements are described in table 8.5.

S9 *Literal values have a type.* The types of literal values are specified in table 8.6.

8.4 Common Constructs

8.4.1 Assignment

An assignment operation evaluates an expression, that may be a literal value, a reference to a symbols or a mathematical equation. It then associates that expression with a symbol, such as variable, parameter or covariate, or with an element in the XML document. An assignment is indicated by the `<Assign>` element. The following rules apply:

S10 *No circular assignment for non-derivative symbols.* A circular assignment occurs if a symbol is initialised with an expression that when traced through the definition of each symbol in the expression ends back where it started. This generally prohibited, but permitted if the symbol being initialised is of derivative type. See section 6.3.2 for a more detailed description.

S11 *A symbol can be assigned only once.* See section 6.3.2.

S12 *Both sides of an assignment must have the same type.* This means that the expression (the right-hand side of the assignment) must evaluate to have a type that is identical to that of the symbol or element it is to be associated with (the left-hand side).

Name	Promotion	Definition
real	real	Values of this type should conform to the double type defined by XML Schema (see `http://www.w3.org/TR/xmlschema-2/#double`).
int	real	Values of this type should conform to the integer type defined by XML Schema (see `http://www.w3.org/TR/xmlschema-2/#integer`).
array	array	A one-dimensional array of `real` values.
string	string	The definition of string conforms to the XML Schema definition (see `http://www.w3.org/TR/xmlschema-2/#string`).
boolean	boolean	A two-valued logic value (True or False). In PharmML we comply with the XML Schema definition (see `http://www.w3.org/TR/xmlschema-2/#boolean`).
id	id	An identifier string, defined as equivalent to a non-colonised named in XMl Schema (see `http://www.w3.org/TR/xmlschema-2/#NCName`).
void	void	A non-type. For consistency in defining language rules it is useful to give some symbols a type that do not have one in any meaningful sense. In such cases we use this type.

Table 8.3: Symbols can be created using these types. The types that can be used with each symbol class can vary. In other cases the type is implicit. This information is defined in the table below. Note that if the type is "explicit" then this means that the range of possible types are specified in the XML document and that the possible types are encoded in the XML Schema definition.

Symbol class	Type
Independent Variable	real
Function Definition	string, real, boolean, id, int
Function Argument	string, real, boolean, id, int
Object	void
Block	void
Variability Level	void
Covariate	continuous: real categorical: id
Simple Parameter	real
Individual Parameter	real
Random Variable	real
Variable	string, real, boolean, id, int
Element ID	void

Table 8.4: Each symbol class has one or more types that it can be assigned to.

8.4.2 Mapping to a Dataset

Elements map the symbol or model to a column in the `<DataSet>` using the `<ColumnRef>` element. This gives us the following rules:

S13 *A column reference must always resolve to a column in its associated dataset.* The associated dataset is clear from the content of reference in the XML Schema structure. To resolve correctly the value `columnIdRef` attribute must be identical to that of the `columnId` attribute in Column definition of dataset.

S14 *A mapping between a symbol or object and a column in a dataset must be type consistent.* By this we mean that the type of the object or element (defined in the table 8.7) must be the same as the type specified in the column definition of the dataset.

S15 *Mapping of derivative variables* If dosing target is `derivativeVariable` then the dosing variable must

Element	Type
PopulationParameter	real
FixedEffect	real
GeneralCovariate	real
GaussianModel/RandomEffects	real
RandomVariable1	real
RandomVariable2	real
CorrelationCoefficient	real
Covariance	real
DerivativeVariable/IndependentVariable	real
InitialTime	real
InitialValue	real
ObservationModel/General	real
ObservationModel/Standard/Output	real
ErrorModel	real
RandomError	real
DoseAmount	real
SteadyState/EndTime	real
SteadyState/Interval	real
DosingTimes	real
Duration	real
Rate	real
SimulationStep/Observations/Continuous	real
SimulationStep/Observations/Discrete	real
Property	real, int, string, boolean or array

Table 8.5: As well as symbols defined by the language quantities can be represented by constructs or concepts in the XML document. In many cases such quantities are assigned by an `<Assign>` element. To ensure type consistency we must understand the type of the quantity on its left-hand side.

Literal	Type	Example
Real	real	`<Real>22.3</Real>`
Int	integer	`<Int>22</Int>`
String	string	`<String>Hel lo</String>`
ID	id	`<Id>hel10</Id>`
True	boolean	`<True/>`
False	boolean	`<False/>`

Table 8.6: In common with other computational languages PharmML provides a mechanism to define literal values. In all cases these literals has a type.

be a derivative variable.

S16 *Mapping of non-derivative variables* If dosing target is `variable` or `parameter` then the dosing variable must be a non-derivative variable.

S17 *Mapping of PK macros* If dosing target is `admType` then the dosing is aimed at a PK macro *depot*, *absorption*, *oral* or *iv*.

S18 *Mapping of column with `columnType="id"`* Dataset column with `columnType="id"` must be mapped to the variability level representing the subject if more then the reference level are defined in a variability

model.

S19 *Multiple DV mapping.* The use of `<CategoryMapping>` element is allowed within `<Piecewise>` only in the context of multiple DV mapping as implemented using `<MultipleDVMapping>`.

Mapping Element	Column Type	Value type	Target of Mapping
ColumnMapping	idv	real	IndependentVariable
	id	id/string	VariabilityModel/Level
	covariate	id/string	ModelDefinition/Covariate (categorical)
	covariate	real	ModelDefinition/Covariate (continues)
	covariate	int	ModelDefinition/VariabilityModel
	dose	real	ModelDefinition/DerivativeVariable, Variable
	dv	real	ModelDefinition/ObservationModel
DemographicMapping		scalar	Demographic
IndividualDosing/DoseAmount		real	DosingRegimen/*/DoseAmount
IndividualDosing/DosingTime		real	DosingRegimen/*/DoseAmount
IndividualDosing/Rate		real	Infusion/Rate
IndividualDosing/Duration		real	Infusion/Duration
IndividualDosing/SSEndTime		real	SteadyState/EndTime
IndividualDosing/SSPeriod		real	SteadyState/Interval

Table 8.7: There are a number of mapping constructs in PharmML that assign the values in the column of a dataset to symbols in the model or objects, for example to instantiate the trial design. In some cases the symbol or object mapped to is implied by the mapping element, in other cases this is explicitly defined with a `<SymbRef>` or `<OidRef>` element.

8.4.3 Array Literal Types

Symbols of array type cannot be defined in PharmML, but there are cases where it is useful to define an array of values, for example when defining a set of dosing times, or a matrix of values. PharmML provides ways to do this. The `<Sequence>` element specifies a uniform sequence of scalar values and the `<Vector>` defines an indexed list of scalar values or expressions. The `<Matrix>` element specifies a 2-dimentional indexed structure of scalar values or expressions. Their usage is governed by the following rules.

C1 *Sequence element validation rules.*

1. Step size cannot be 0.
2. Steps greater than 0 implies that Begin must be greater than or equal to End.
3. Steps les than 0 implies that Begin must be less than or equal to End.
4. Repetitions must be greater than or equal to zero.
5. The type of the sequence must be consistent. Types may be promoted to maintain type consistency. For example if the value of the step size is real type and the begin and end elements have integer values then all will be promoted to a real type and the construct will generate sequence of real numbers.

C2 *Vector validation rules.*

1. It must contain values that are type consistent. Types may be promoted to main type consistency, in which case all values in the vector will be of the promoted type.
2. The order of elements in the vector are significant. Values can be repeated and the values are not sorted in any way.
3. Length of vector cannot be smaller the number of explicitly defined elements.

4. The `default` and `length` attributes must be defined if not all elements are specified.

C3 *Matrix validation rules.*

1. No overlapping elements or indexes are allowed.
2. The `default` and `length` attributes must be defined if not all matrix elements are specified.

8.5 Dataset

The dataset defines a table of data. It is described in some detail in section 6.6.

DS1 *Only one column with `columnType="id"` attribute is allowed.*

DS1 *Columns in a dataset file.* All columns must be present in the file and must have a unique `columnNum` attribute value.

DS2 *Each cell must contain a value that is type compatible with the column definition.*

DS3 *Each row must define a cell for each column.*

DS4 *The dataset file must be present.*

8.5.1 Trial design related datasets

Additionally when trial design in defined explicitly in PharmML with external dataset referenced in the `<Population>`, `<IndividualDosing>` and `<ObjectiveDataSet>` elements following rules apply

DS5 *Only CSV files are allowed.* The dataset must be in character-separated value (CSV) format.

DS6 *Allowed set of delimiters.* Delimiters allowed are COMMA, SEMICOLON, SPACE and TAB and cannot be escaped with in the dataset.

DS7 *Character set encoding.* The character set needs to be encoded in UTF-8.

DS8 *Use of strings.* String values must be encoded within quotes.

DS9 *File headings.* Headings are not allowed in the data file.

DS10 *Commenting unused lines.* Lines can be commented using #.

8.6 Maths

As described in more detail in section 6.8 the definition of mathematical expressions in PharmML relies on a combination of literal values, symbol references, and binary and unary operators. The operands of the operators needs more detailed definition.

8.6.1 Numerical Operators

T1 *The operands of the binary numerical operators have specified semantics.* The semantics are defined in table 8.8.

T2 *The operands of the unary numerical operators have specified semantics.* The semantics are defined in table 8.9.

T3 *All numerical operators take one or more operands of type real and return a result of type real.*

Operator	Definition	Operand 1	Operand 2
plus	Addition: $a + b$	a	b
minus	Subtraction: $a - b$	a	b
times	Multiplication: $a \times b$	a	b
divide	Division: a/b	a	b
power	Power: x^y	base (x)	exponent (y)
root	Root: $\sqrt[y]{x}$	radicand (x)	degree (y)
logx	Logarithm: $\log_y(x)$	power (x)	base (y)
min	Minimum: $min(a, b)$	a	b
max	Maximum: $max(a, b)$	a	b
rem	Remainder: $rem(a, b)$	a	b
atan2	$atan2(a, b)$	a	b

Table 8.8: Numerical binary operator semantics

8.6.2 Logical Operator

T4 *The operands of the binary logical operators have specified semantics.* The semantics are defined in table 8.10.

T5 *The operands of the unary logical operators have specified semantics.* The semantics are defined in table 8.11.

T6 *The logical binary operators take operands of a specified type.* The types are defined in table 8.10.

T7 *The logical unary operators take operands of a specified type.* The types are defined in table 8.11.

T8 *All the logical operators return a result of type boolean.*

8.6.2.1 Logical Unary Operators

8.6.3 Constants

T9 *All mathematical constants have type real.*

T10 *The constants have specified semantics.* The semantics are defined in table 8.12.

8.7 ModelDefinition

The following rules relate to the `<ModelDefinition>` element and its children.

M1 *Only one uninitialised category permitted in a categorical covariates.* If one category is assigned a value in the ModelDefinition then all but at most one category must be assigned a value.

M2 *The sum of the discrete probabilities of categorical covariates must equal 1.* If all categories are assigned a probability then they must sum to 1. If a category is unassigned then it has a probability of 1-sum of all the remaining probabilities.

M4 *All Variability Levels must be used.* All variability levels in the model definition must be used by at least one random effect in the model.

M5 *Random effect correlations must be unique.* A correlation between identical random effects at the same variability level of the model is forbidden.

M6 *The variability levels within a particular type must be defined as a list.* Each variability level can only have a maximum of one child level and one parent level. Variability levels that belong to models with different types (specified by the `type` attribute) cannot be linked.

Operator	Definition
exp	Exponential function e^x
log	Natural logarithm: $\log(x)$
minus	Negation: $-x$
sqrt	Square root : $\sqrt{x}$
factorial	Factorial: $x!$
sign	Sign: $sign(x)$
Heaviside	Heaviside function: $H(x)$
sin	Sine function: $\sin(x)$
cos	Cosine function: $\cos(x)$
tan	Tangent function: $\tan(x)$
sec	Secant function: $\sec(x)$
csc	Cosecant function: $\csc(x)$
cot	Cotangent function: $\cot(x)$
sinh	Hyperbolic sine function: $\sinh(x)$
cosh	Hyperbolic cosine function: $\cosh(x)$
tanh	Hyperbolic tangent function: $\tanh(x)$
sech	Hyperbolic secant function: $sech(x)$
csch	Hyperbolic cosecant function: $csch(x)$
coth	Hyperbolic cotangent function: $\coth(x)$
arcsin	Inverse Sine function: $\arcsin(x)$
arccos	Inverse Cosine function: $\arccos(x)$
arctan	Inverse Tangent function: $\arctan(x)$
arcsec	Inverse Secant function: $arcsec(x)$
arccsc	Inverse Cosecant function: $arccsc(x)$
arccot	Inverse Cotangent function: $arccot(x)$
arcsinh	Inverse Hyperbolic sine function: $arcsinh(x)$
arccosh	Inverse Hyperbolic cosine function: $arccosh(x)$
arctanh	Inverse Hyperbolic tangent function: $arctanh(x)$
arcsech	Inverse Hyperbolic secant function: $arcsech(x)$
arccsch	Inverse Hyperbolic cosecant function: $arccsch(x)$
arccoth	Inverse Hyperbolic cotangent function: $arccoth(x)$
floor	Floor: rounds down (towards $-\infty$) to the nearest integer.
ceiling	Ceiling: rounds up (towards $+\infty$) to the nearest integer.
abs	Absolute value: $\|x\|$
logistic	Logistic function: $f(x) = \frac{1}{1+e^{-x}}$
logit	Inverse of the logistic function: $logit(x)$
probit	Probit function: $probit(x)$

Table 8.9: Numerical unary operator semantics

M6 *Distribution of random effects.* When using `<GaussianModel>`, the according random effect must be normally distributed.

M7 *Specification of variances of random effects must the unique.* The correlation structure of random effects can be defined by a variance-covariance matrix within the `<Correlation>` element along with the definition of individual parameters and their according random effects. The variances must then be identical of the corresponding random effects.

M8 *Referencing variability model.* References within the `<VariabilityReference>` must point to an existing `<VariabilityModel>`.

Operator	Definition	Operand 1	Operand 2
lt	$<$	real	real
leq	$\leq$	real	real
gt	$>$	real	real
geq	$\geq$	real	real
eq	$=$	real	real
	$=$	boolean	boolean
neq	$\neq$	real	real
	$\neq$	boolean	boolean
and	Boolean AND	boolean	boolean
or	Boolean OR	boolean	boolean
xor	Boolean XOR	boolean	boolean

Table 8.10: Logical binary operator semantics and expected types.

Operator	Definition	Operand
isDefined	Test if a value is Not NULL	any scalar type
not	Boolean NOT	boolean

Table 8.11: Logical unary operator semantics

Constant	Definition
notanumber	Corresponds to the IEEE NaN[2].
pi	Pi: π
exponentiale	Eulers number: e
infinity	Infinity: ∞

Table 8.12: Numerical constant semantics

8.8 Trial Design

D1 *Demographic values of must all have the same type.* The values of a demographic, specified by the `<Demographic>` element, must all have the same type.

D2 *Dosing cannot start before the model is initialised.* This means that all dosing times must start at or after the initial time of the model.

D3 *IV dependent attributes cannot begin before the model is initialised.* For example a time dependent covariate cannot occur before the initial time of the model.

D4 *Single dose amount and multiple dosing times permitted.* In a dosing regiment, if a single dose amount is specified with multiple doses then this indicates that the same dose amount should be administered at each dosing time.

D5 *Multiple dose amount and multiple dosing times permitted.* In a dosing regiment, if a multiple doses amount is specified with multiple doses then each dose amount is administered at the corresponding dosing time. The order of the amounts and times is significant so the first amount is administered at the first time and so on. Note that the vectors of amount and time must be *exactly the same length.*

D6 *More than one dosing time implies multiple dosing.*

D7 *An Epoch period must progress in time.* The end of an epoch must be equal to or after its beginning. It must also have both a start and end time.

D8 *A trial structure must be complete.* A `<Structure>` element must contain at least one each of Epoch, Arm, Cell, Segment and Activity.

D9 *An Observation Group period must progress in time.* The end of an observation group must be after its beginning. It must also have a duration greater than zero. It must also have both a start and end time.

D10 *Dosing times relative to epoch.* Dosing times are relative to the start time of the epoch to which they are assigned.

8.9 Modelling Step

L1 *No uninitialised symbols.* All symbols such as variables, parameters, and covariates must be initialised. By initialised we mean that they must have an initial assignment (including an initial condition) or an initial estimate defined.

L2 *Times used in a modelling step cannot occur before the initial time of the model.*

Chapter 9

Worked Examples

During the development process of an exchange format such as PharmML it is very useful to test, validate and explain it using real-life examples. After providing a short (mathematical) description of some basic pharmacometric models the corresponding XML representation in PharmML will be provided and commented. Each example is designed to illustrate different aspects of PharmML and what it can do and — perhaps equally importantly — what it cannot do. For clarity and to save space we will only show key excerpts from the examples, but the complete examples are available and will be distributed with this document and can be downloaded from PharmML related websites, `http://ddmore.eu/pharmml` and `http://pharmml.org` .

9.1 Model structure is task and approach dependent

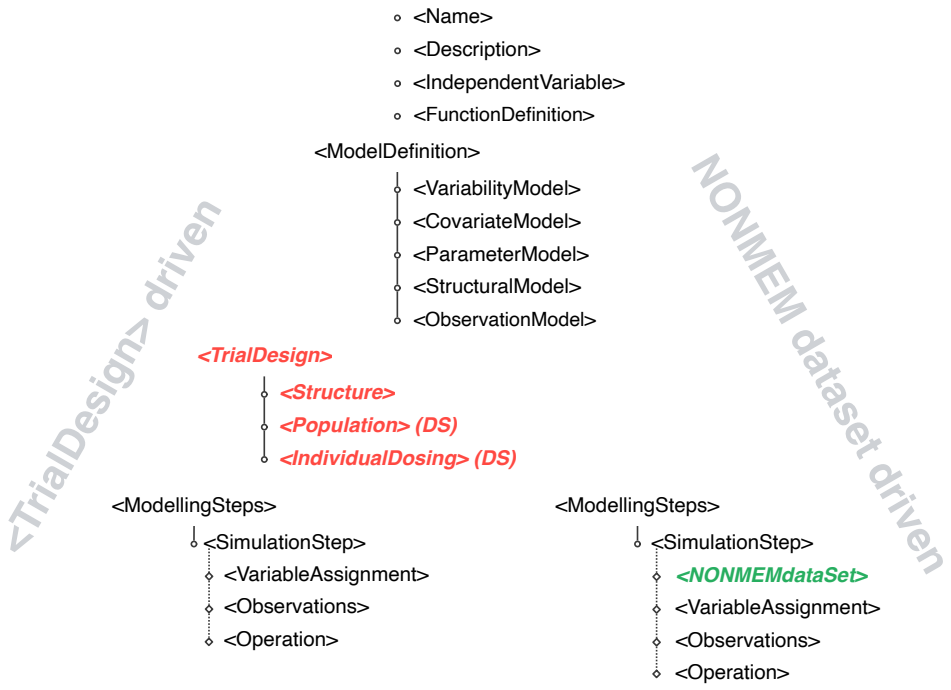


Figure 9.1: PharmML building blocks used for a **SIMULATION** task implemented using the `<TrialDesign>` (left) or NONMEM dataset (right) driven option. (DS) indicates that tabular data structure is used.

Figures 9.2 and 9.1 show an comparison in the structure to be implemented for typical simulation and estimation tasks. Colours underline elements where the structure of PharmML for these tasks differs dependent whether the `<TrialDesing>` or NONMEM datasets are used to inform the model about the underlying study design and experimental data.

Figure 9.2: PharmML building blocks used for an **ESTIMATION** task implemented using the `<TrialDesign>` (left) or NONMEM dataset (right) driven option. (DS) indicates that tabular data structure is used.

The following sections will provide two versions of a number of simulation and estimation examples, each with the complete description of the according model, trial design and task implementation.

9.2 Example 1: Simulation, PK + PD response

The following example[1] is based on the CTS1 use case [Lavielle and Grevel, 2011]. Both PK (the drug concentration) and PD (the drug effect) are simulated. A one compartment PK model is linked to an indirect response PD model, see Figure 9.3 for a typical simulation result for one patient.

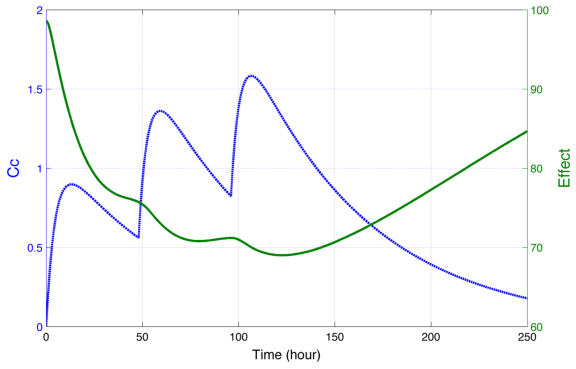


Figure 9.3: Simulated combined model as defined in the example with PK (blue) and PD (green) time courses for one subject. Here three doses were administered every $48h$.

[1]The example is encoded in two versions, `example1.xml` and `example1_NONMEM.xml`, with explicit encoded trial design and design sourced from a NONMEM datafile, respectively.

9.2.1 Model Definition

9.2.1.1 Structural model

This is an oral one compartment model and an indirect response model with parameters ka, V, CL, $Imax$, $IC50$, Rin and $kout$.

$$\begin{aligned} k &= \frac{CL}{V} \\ \frac{dAd}{dt} &= -ka \times Ad \\ \frac{dAc}{dt} &= ka \times Ad - k \times Ac \\ \frac{dE}{dt} &= Rin \times \left(1 - \frac{Imax \times Cc}{Cc + IC50}\right) - kout \times E \\ Cc &= \frac{Ac}{V} \end{aligned} \tag{9.1}$$

initial conditions:

$$\begin{aligned} E(t=0) &= \frac{Rin}{kout} \\ Ad(t=0) &= 0 \\ Ac(t=0) &= 0 \end{aligned} \tag{9.2}$$

9.2.1.2 Covariate model

The only covariate is Weight, W, and it is a continuous covariate:

$$W \sim \mathcal{N}(pop_W, \omega_W) \tag{9.3}$$

The following transformation is applied:

$$\log(W/70) \tag{9.4}$$

and the initial values are:

$$pop_W = 70.07, \quad \omega_W = 14.09$$

9.2.1.3 Parameter model

PK parameters The model uses the following individual parameters:

ka absorption rate constant

V volume of distribution

CL clearance of elimination

All follow a log-normal distribution:

$$\log(ka_i) = \log(pop_{ka}) + \eta_{ka,i} \tag{9.5}$$

$$\log(V_i) = \log(pop_V) + \beta_{1,V} \log(W_i/70) + \eta_{V,i} \tag{9.6}$$

$$\log(CL_i) = \log(pop_{CL}) + \beta_{1,CL} \log(W_i/70) + \eta_{CL,i}$$

where

$$\eta_{ka,i} \sim N(0, \omega_{ka}), \quad \eta_{V,i} \sim N(0, \omega_V), \quad \eta_{CL,i} \sim N(0, \omega_{CL})$$

with initial values:

$$pop_{ka} = 1, \quad \omega_{ka} = 0.6 \qquad pop_V = 8, \quad \omega_V = 0.2$$
$$pop_{CL} = 0.13, \quad \omega_{CL} = 0.2 \qquad \beta_{1,V} = 1, \quad \beta_{1,CL} = 0.75$$
$$\rho_{V,CL} = 0.7^2$$

PD parameters The model uses the following individual parameters:

Imax maximal antagonistic response

IC50 concentration giving half the maximal response

Rin input (synthesis) rate

kout output (elimination) rate

All follow a log-normal distribution, except *Imax*, which follows a logit-normal distribution.

$$\begin{aligned} logit(Imax_i) &= logit(pop_{Imax}) + \eta_{Imax,i} \\ \log(IC50_i) &= \log(pop_{IC50}) + \eta_{IC50,i} \\ \log(Rin_i) &= \log(pop_{Rin}) + \eta_{Rin,i} \\ \log(kout_i) &= \log(pop_{kout}) + \eta_{kout,i} \end{aligned}$$

where

$$\eta_{Imax,i} \sim N(0, \omega_{Imax}), \quad \eta_{IC50,i} \sim N(0, \omega_{IC50}),$$
$$\eta_{Rin,i} \sim N(0, \omega_{Rin}), \quad \eta_{kout,i} \sim N(0, \omega_{kout})$$

with initial values:

$$pop_{Imax} = 0.9, \quad \omega_{Imax} = 2 \qquad pop_{IC50} = 0.4, \quad \omega_{IC50} = 0.4$$
$$pop_{Rin} = 5, \quad \omega_{Rin} = 0.05 \qquad pop_{kout} = 0.05, \quad \omega_{kout} = 0.05$$

Variance-covariance matrix The full variance-covariance matrix for the random effects is:

$$\Omega = \begin{pmatrix} \omega_{ka}^2 & 0 & 0 & 0 & 0 & 0 & 0 \\ & \omega_V^2 & \omega_{V,CL} & 0 & 0 & 0 & 0 \\ & & \omega_{CL}^2 & 0 & 0 & 0 & 0 \\ & & & \omega_{Imax}^2 & 0 & 0 & 0 \\ & & & & \omega_{IC50}^2 & 0 & 0 \\ & & & & & \omega_{Rin}^2 & 0 \\ & & & & & & \omega_{kout}^2 \end{pmatrix} \tag{9.7}$$

where

$$\omega_{V,CL} = \omega_V \, \omega_{CL} \, \rho_{V,CL}$$

9.2.1.4 Observation model

We apply a residual error model to the output variables *Cc* and *E* from the PK and PD models respectively.

Output Variable	***Cc***	***E***
Observation Name	Concentration	PCA
Units	mg/l	%
Type	Continuous	Continuous
Model	Combined	Constant
Parameters	$a = 0.5, \quad b = 0.1$	$a = 4$

[2]Correration coefficient between $\eta_{V,i}$ and $\eta_{CL,i}$

9.2.2 Trial Design

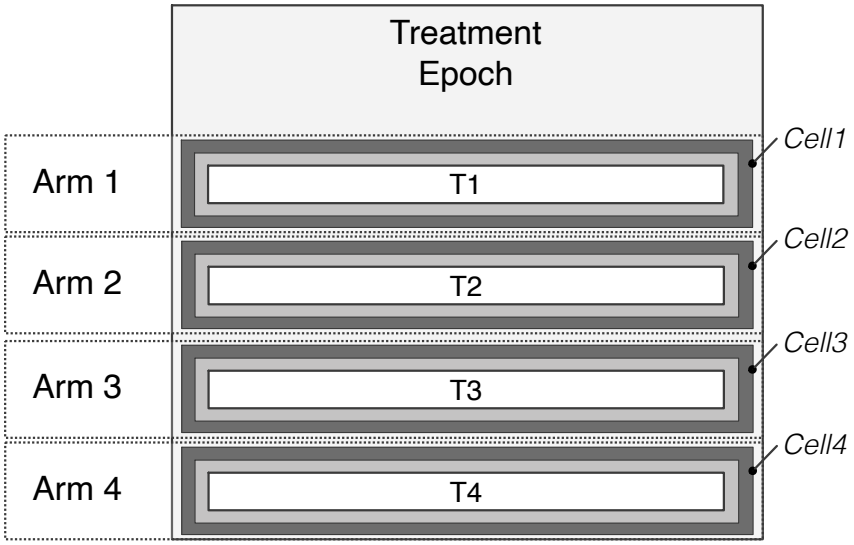


Figure 9.4: Design overview: this study consists of four arms and one epoch. The differences between arms lie in the number of subjects, dose amount and times. See table below for the details.

Figure 9.4 shows the *Structure* of this example consisting of four arms and one epoch, meaning there are four treatment types for which only one time period needs to be specified.

The dosing regimen for the trial is given for each arm below. Note that all dosing is bolus dosing (discrete administration at specific times) and all doses are administered to the same compartment.

Arm	**1**	**2**	**3**	**4**
Number of subjects	20	20	40	40
Dose target	*Ad*	*Ad*	*Ad*	*Ad*
Dosing Amount	0.25	0.5	0.5	1
Dose Units	mg/kg	mg/kg	mg/kg	mg/kg
Dose per kg	yes	yes	yes	yes
Dosing times (h)	0:24:192	0:48:192	0:24:192	0:48:192

9.2.3 Modelling Steps

Time of measurement for PK and PD happens according to different schedules and these observation time points are produced by the simulation. The output variables to be generated by the simulation and their associated time points are shown below:

Output Variable	***Cc***	***E***
Observation times	$[0.5{,}4 : 4 : 48, 52 : 24 : 192, 192 : 4 : 250]$	$0 : 24 : 288$

9.2.4 PharmML Document Structure

An overview of the model with the key sections collapsed as shown in this listing

```
<PharmML xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns="http://www.pharmml.org/pharmml/0.6/PharmML"
    xsi:schemaLocation="http://www.pharmml.org/pharmml/0.6/PharmML http://www.pharmml.org/pharmml/0.6/PharmML"
    xmlns:math="http://www.pharmml.org/pharmml/0.6/Maths"
    xmlns:ct="http://www.pharmml.org/pharmml/0.6/CommonTypes"
    xmlns:ds="http://www.pharmml.org/pharmml/0.6/Dataset"
    xmlns:mdef="http://www.pharmml.org/pharmml/0.6/ModelDefinition"
    xmlns:mstep="http://www.pharmml.org/pharmml/0.6/ModellingSteps"
    xmlns:mml="http://www.pharmml.org/pharmml/0.6/PharmML"
```

```
    implementedBy="MJS" writtenVersion="0.5.1"
    metadataFile="example1.rdf" id="i1">
    <ct:Name>Example 1 - simulation continuous PK/PD</ct:Name>
    <IndependentVariable symbId="t"/>
    <FunctionDefinition xmlns="http://www.pharmml.org/pharmml/0.6/CommonTypes"
        symbId="constantErrorModel" symbolType="real">
        <!-- omitted details -->
    </FunctionDefinition>
    <FunctionDefinition xmlns="http://www.pharmml.org/pharmml/0.6/CommonTypes"
        symbId="combinedErrorModel" symbolType="real">
        <!-- omitted details -->
    </FunctionDefinition>
    <ModelDefinition xmlns="http://www.pharmml.org/pharmml/0.6/ModelDefinition">
        <!-- omitted details -->
    </ModelDefinition>
    <TrialDesign xmlns="http://www.pharmml.org/pharmml/0.6/TrialDesign">
        <!-- omitted details -->
    </TrialDesign>
    <ModellingSteps xmlns="http://www.pharmml.org/pharmml/0.6/ModellingSteps">
        <!-- omitted details -->
    </ModellingSteps>
</PharmML>
```

illustrates the main sections of the model as described in Section 6.2. The key points to note are the use of the `<IndependentVariable>` element to set the time variable to t (c.f. section 6.11), and how the element `<Name>` defines the name of the model. In addition the top-level `<PharmML>` element contains a number of attributes prefixed `xmlns` (`xmlns` stands for XML namespace.). These are required by the XML Schema standard and can be ignored as we go through the examples[3].

Few attributes are use, such as

- `implementedBy` – to identify the person who implemented the model
- `writtenVersion` – to identify the PharmML version
- `metadataFile` – specifies the optional RDF file associated with the model introduced in Section 6.14.1
- `id` – metadata element, described in Section 6.12.

The following listing introduces the `<FunctionDefinition>` element. It defines a function that returns a real type as shown in the following listing

```
<FunctionDefinition xmlns="http://www.pharmml.org/pharmml/0.6/CommonTypes"
    symbId="combinedErrorModel" symbolType="real">
    <FunctionArgument symbId="a" symbolType="real"/>
    <FunctionArgument symbId="b" symbolType="real"/>
    <FunctionArgument symbId="f" symbolType="real"/>
    <Definition>
        <Equation xmlns="http://www.pharmml.org/pharmml/0.6/Maths">
            <Binop op="plus">
                <ct:SymbRef symbIdRef="a"/>
                <Binop op="times">
                    <ct:SymbRef symbIdRef="b"/>
                    <ct:SymbRef symbIdRef="f"/>
                </Binop>
            </Binop>
        </Equation>
    </Definition>
</FunctionDefinition>
```

The function takes three arguments of scalar type and defines the function:

$$combinedError(a, b, f) = a + bf$$

This function is used to encode the combined error model function used later in the observation model (see Section 9.2.5.6).

[3]To learn more about the technical aspects of XML and the XML Schema standard used to define PharmML then the recommend place to start is: `http://www.w3.org/TR/xmlschema-0/`.

9.2.5 Model Definition

As you can see in the following listing

```
<ModelDefinition xmlns="http://www.pharmml.org/pharmml/0.6/ModelDefinition">
    <VariabilityModel blkId="vm1" type="parameterVariability">
        <!-- omitted details -->
    </VariabilityModel>
    <VariabilityModel blkId="vm2" type="residualError">
        <!-- omitted details -->
    </VariabilityModel>
    <CovariateModel blkId="cm1">
        <!-- omitted details -->
    </CovariateModel>
    <ParameterModel blkId="p1">
        <!-- omitted details -->
    </ParameterModel>
    <StructuralModel blkId="sm1">
        <!-- omitted details -->
    </StructuralModel>
    <ObservationModel blkId="om1">
        <!-- omitted details -->
    </ObservationModel>
    <ObservationModel blkId="om2">
        <!-- omitted details -->
    </ObservationModel>
</ModelDefinition>
```

the model definition section defines the main components that you will see in the subsequent examples. All of these elements are optional and all follow the scoping rules described in section 6.3.2.

9.2.5.1 Variability Model

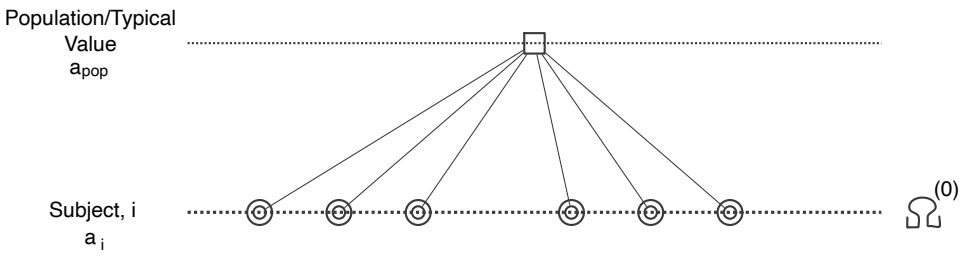


Figure 9.5: There is only one level of variability in this example – inter-individual variability.

From the following listing one can also see that there are two variability models defined using the element `<VariabilityModel>`. This is how we explicitly define the level of variability in the model (c.f. section 3.5). In this case there is one level of variability, the inter-individual, Figure 9.5 and the residual error model variability, see the following listing

```
<VariabilityModel blkId="vm1" type="parameterVariability">
    <Level referenceLevel="true" symbId="indiv">
        <ct:Name>Individual Variability</ct:Name>
    </Level>
</VariabilityModel>
<VariabilityModel blkId="vm2" type="residualError">
    <Level symbId="residual">
        <ct:Name>Residual Error</ct:Name>
    </Level>
</VariabilityModel>
```

for the details of the implementation. Please, note that we use two different variability attribute types, `parameterVariability` and `residualError`. The former stands for random variability associated with the parameters, as described in section 3.5. For this example we are defining one level of variability in the parameter model and this corresponds to variability between subjects. The latter stands for residual error variability.

The attribute `referenceLevel="true"` identifies the `indiv` level as the reference level. This attribute is not required if there is only one level and the proper mapping from the dataset or `<Population>` in the `<TrialDesign>` section is defined. However it is required if a model is implemented using the `<ModelDefinition>` section only. This will be explained again when dealing with examples with more complex variability models (section 9.5.3).

9.2.5.2 Covariate Model

The `<CovariateModel>` block corresponds to the covariate model defined in section 3.6.5. In this example, as shown in this listing

```
<CovariateModel blkId="cm1">
    <SimpleParameter symbId="pop_W"/>
    <SimpleParameter symbId="omega_W"/>
    <Covariate symbId="W">
        <Continuous>
            <NormalDistribution xmlns="http://www.uncertml.org/3.0"
                    definition="http://www.uncertml.org/distributions/normal">
                <mean>
                    <var varId="pop_W"/>
                </mean>
                <variance>
                    <var varId="omega_W"/>
                </variance>
            </NormalDistribution>
            <Transformation>
                <TransformedCovariate symbId="logW70"/>
                <Equation xmlns="http://www.pharmml.org/pharmml/0.6/Maths">
                    <Uniop op="log">
                        <Binop op="divide">
                            <ct:SymbRef symbIdRef="W"/>
                            <ct:Real>70.0</ct:Real>
                        </Binop>
                    </Uniop>
                </Equation>
            </Transformation>
        </Continuous>
    </Covariate>
</CovariateModel>
```

we are defining a continuous covariate, W, indicated by the `<Continuous>` element and the covariate is sampled from a normal distribution as in equation 9.3. The element `<Transformation>` beneath the definition of the distribution describes the transformation applied to this covariate. We can either refer to the untransformed or transformed form of the covariate whatever form is required. The later will be refered to as *logW70* as defined in the `<TransformedCovariate>` element. In this case the transformation being applied is defined in equation 9.4.

9.2.5.3 Parameter Model

All parameters in the current example can be defined using the Gaussian model with linear covariates (see section 3.6). We will start with a simple case in this example, shown in the following listing

```
<ParameterModel blkId="p1">
    <!-- omitted other parameters -->
    <!-- ka -->
    <SimpleParameter symbId="pop_ka"/>
    <SimpleParameter symbId="omega_ka"/>
    <RandomVariable symbId="eta_ka">
        <ct:VariabilityReference>
            <ct:SymbRef blkIdRef="vm1" symbIdRef="indiv"/>
        </ct:VariabilityReference>
        <NormalDistribution xmlns="http://www.uncertml.org/3.0"
            definition="http://www.uncertml.org/distributions/normal">
            <mean><rVal>0</rVal></mean>
            <stddev><var varId="omega_ka"/></stddev>
        </NormalDistribution>
    </RandomVariable>
    <IndividualParameter symbId="ka">
        <GaussianModel>
```

```
            <Transformation>log</Transformation>
            <LinearCovariate>
                <PopulationParameter>
                    <ct:Assign>
                        <ct:SymbRef symbIdRef="pop_ka"/>
                    </ct:Assign>
                </PopulationParameter>
            </LinearCovariate>
            <RandomEffects>
                <ct:SymbRef symbIdRef="eta_ka"/>
            </RandomEffects>
        </GaussianModel>
    </IndividualParameter>
```

the absorption rate constant *ka*. First the typical value for *ka*, *pop_ka* and the standard deviation of the random effect, *omega_ka*, are defined as `<SimpleParameter>`'s. Then the random effect *eta_ka* is defined, which follows a normal distribution with mean 0 and standard deviation *omega_ka*.

Then the actual individual parameter *ka* is defined with the `<Transformation>` attribute set to `log`. This tells us that the parameter is log-transformed. We are using the Gaussian model described previously (section 3.6), and it follows a log-normal distribution with the mean *pop_ka* and the standard deviation *omega_ka*, as shown in equation 9.5. Even though *ka* is modelled without a covariate, we still use the element `<LinearCovariate>`, which contains here logically only the typical value element `<PopulationParameter>`.

The `<RandomVariable>` element is very important here. It tells us what the random effect is, and, by assigning a variability level to the attribute `symbIdRef` within the `<VariabilityReference>` element, it effectively tells us that the random effect is sampled for every subject. The importance of this will become clearer in later examples that describe multiple levels of variability. Note that the `<RandomVariable>` element also defines a new symbol *eta_ka*, which is a parameter.

Of course not all parameter definitions are so straight forward. In the following listing

```
<ParameterModel blkId="p1">
    <!-- V -->
    <SimpleParameter symbId="beta_V"/>
    <SimpleParameter symbId="pop_V"/>
    <SimpleParameter symbId="omega_V"/>
    <RandomVariable symbId="eta_V">
        <ct:VariabilityReference>
            <ct:SymbRef blkIdRef="vm1" symbIdRef="indiv"/>
        </ct:VariabilityReference>
        <NormalDistribution xmlns="http://www.uncertml.org/3.0"
            definition="http://www.uncertml.org/distributions/normal">
            <mean><rVal>0</rVal></mean>
            <stddev><var varId="omega_V"/></stddev>
        </NormalDistribution>
    </RandomVariable>
    <IndividualParameter symbId="V">
        <GaussianModel>
            <Transformation>log</Transformation>
            <LinearCovariate>
                <PopulationParameter>
                    <ct:Assign>
                        <ct:SymbRef symbIdRef="pop_V"/>
                    </ct:Assign>
                </PopulationParameter>
                <Covariate>
                    <ct:SymbRef blkIdRef="cm1" symbIdRef="logW70"/>
                    <FixedEffect>
                        <ct:SymbRef symbIdRef="beta_V"/>
                    </FixedEffect>
                </Covariate>
            </LinearCovariate>
            <RandomEffects>
                <ct:SymbRef symbIdRef="eta_V"/>
            </RandomEffects>
        </GaussianModel>
    </IndividualParameter>
```

you can see the definition of a parameter, V, that is related to the covariate, W. We do this using the element `<Covariate>` to indicate there is a relationship. Then we reference the specific covariate (using the `<SymbRef>` element) and describe the fixed effect relating the covariate to this parameter: in this case we

are referring to the parameter $beta\_V$. We define here a linear covariate model when relating covariates to parameters (c.f. section 3.6.5), so this example corresponds to equation (9.6).

Equation (9.6) uses the transformed covariate $\log(W/70)$ as defined in (9.4), and implemented in the `<CovariateModel>` (section 9.2.5.2). It is the decision of the modeller whether to use an untransformed or transformed form of a covariate and part of the covariate model building.

Having defined the individual and other parameters in the parameter model we need to describe the correlation of the random effects, if any. As described above (see section 3.6.4 for a complete description) this is done using a covariance matrix. In PharmML, if the random effects of both parameters follow a normal distribution, then we assume that the diagonal of the covariance matrix can be derived from the variance or standard deviation of each parameter (c.f section 3.6.4). This is true in this example, so one only needs to define the parts of the covariance structure that define the correlation between parameters V and CL as shown in this listing

```
<Correlation>
    <ct:VariabilityReference>
        <ct:SymbRef blkIdRef="vm1" symbIdRef="indiv"/>
    </ct:VariabilityReference>
    <Pairwise>
        <RandomVariable1>
            <ct:SymbRef symbIdRef="eta_V"/>
        </RandomVariable1>
        <RandomVariable2>
            <ct:SymbRef symbIdRef="eta_Cl"/>
        </RandomVariable2>
        <CorrelationCoefficient>
            <ct:SymbRef symbIdRef="rho_V_Cl"/>
        </CorrelationCoefficient>
    </Pairwise>
</Correlation>
```

With the `<VariabilityReference>` element and its `symbIdRef` attribute the correlation is associated with a certain variability level as defined at the beginning of the model definition. To specify the correlation structure we can either defined the full covariance matrix using `<Matrix>` or as in this case the `<Pairwise>` element because only two random effects are correlated. Accordingly we need to define the random effects of interest and either correlation coefficient or their covariance. This single `<Correlation>` element is all we need to define the sparse covariance matrix mentioned above (section 9.2.1.3).

9.2.5.4 Structural Model

In PharmML the structural model can be described using algebraic equations or using ODEs/DDEs. In this case the model is defined as a combination of algebraic and ODE equations (c.f. equation 9.1). We demonstrate how to encode two different assignment types, e.g.

$$
\begin{aligned}
k &= Cl/V \\
\frac{dAd}{dt} &= -ka \times Ad, \quad Ad(t=0) = 0 \\
\frac{dAc}{dt} &= ka \times Ad - k \times Ac, \quad Ac(t=0) = 0
\end{aligned}
$$

The following listing

```
<StructuralModel blkId="sm1">
    <SimpleParameter symbId="k">
        <ct:Assign>
            <Equation xmlns="http://www.pharmml.org/pharmml/0.6/Maths">
                <Binop op="divide">
                    <ct:SymbRef blkIdRef="p1" symbIdRef="Cl"/>
                    <ct:SymbRef blkIdRef="p1" symbIdRef="V"/>
                </Binop>
            </Equation>
        </ct:Assign>
    </SimpleParameter>
    <ct:DerivativeVariable symbId="Ad" symbolType="real">
        <ct:Assign>
            <Equation xmlns="http://www.pharmml.org/pharmml/0.6/Maths">
                <Binop op="times">
                    <Uniop op="minus">
```

```
                        <ct:SymbRef blkIdRef="p1" symbIdRef="ka"/>
                    </Uniop>
                    <ct:SymbRef symbIdRef="Ad"/>
                </Binop>
            </Equation>
        </ct:Assign>
        <ct:IndependentVariable>
            <ct:SymbRef symbIdRef="t"/>
        </ct:IndependentVariable>
        <ct:InitialCondition>
            <ct:InitialValue>
                <ct:Assign>
                    <ct:Real>0</ct:Real>
                </ct:Assign>
            </ct:InitialValue>
        </ct:InitialCondition>
    </ct:DerivativeVariable>
    <!-- omitted dAc/dt = ... -->
```

illustrates how the parameter k is defined, i.e. as the ratio of CL and V, which are parameters as well but defined in the `<ParameterModel>`, *p1*. Then the derivative Ad is defined using the element `<DerivativeVariable>` (see section 6.7 for a more detailed explanation) with t as the independent variable, the time. The absorption parameter *ka* is again defined in *p1*. Finally the `<InitialValue>` element within the `<InitialCondition>` defines the initial value for Ad at time 0, which is the default initial time in PharmML and doesn't have to be specified in this case explicitly. Alternatively, any initial time values can be encoded using `<InitialTime>` element.

9.2.5.5 Structural model – using PK macros

The structural model is this time formulated using the PK macros. The PK model corresponds to the ADVAN2/TRANS combination in the PREDPP library, see Table 5.5. Following table shows the two allowed parameterizations for ADVAN2 and the re-parameterization formula.

TRANS1	TRANS2	Formula
K Rate constant of elimination	CL Clearance	K=CL/V
	V Volume of distribution	

Table 9.1: ADVAN2 with two available parameterization routines, TRANS1 and TRANS2.

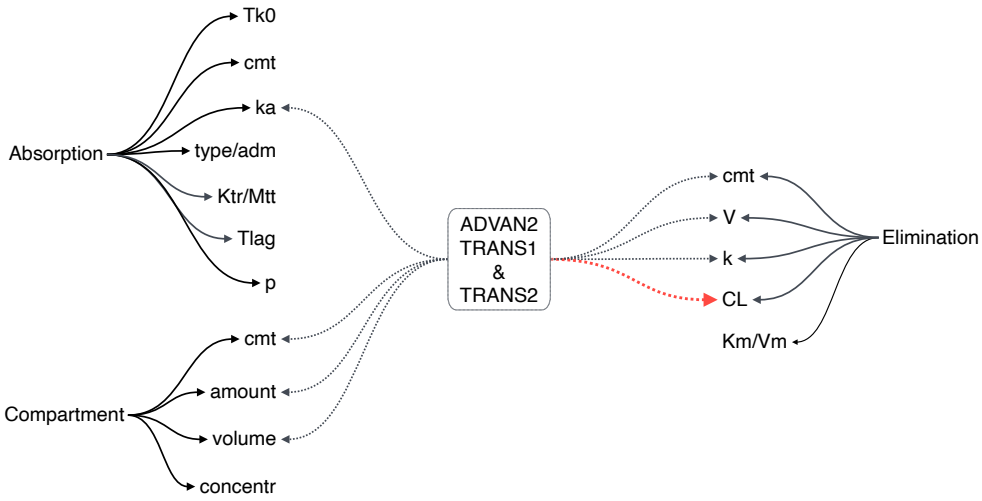


Figure 9.6: Relationship between PK macros and the predefined PREDPP models and alternative parameterizations for the structural model, corresponding to ADVAN2/TRANS1 and ADVAN2/TRANS2.

Table 9.2 shows the PK macros for the two options and the following code snippet shows how this can be encoded in PharmML

ADVAN2, TRANS1	ADVAN2, TRANS2
compartment(cmt=1, amount=Ac, volume=V)	compartment(cmt=1, amount=Ac, volume=V)
oral(cmt=1,ka)	oral(cmt=1,ka)
elimination(cmt=1, k)	elimination(cmt=1, CL)
Cc = Ac/V	Cc = Ac/V

Table 9.2: The set of macros for the PK model with additional equation for the concentration, *Cc*. It corresponds to a ADVAN2/TRANS2 model, see Section 5.3 and Table 5.5 for detailed description. Here the two allowed parameterizations are used.

```
<StructuralModel blkId="sm1">
    <ct:Variable symbolType="real" symbId="Ac"/>
    <ct:Variable symbolType="real" symbId="Cc">
        <ct:Assign>
            <math:Equation>
                <math:Binop op="divide">
                    <ct:SymbRef symbIdRef="Ac"/>
                    <ct:SymbRef blkIdRef="pm1" symbIdRef="V"/>
                </math:Binop>
            </math:Equation>
        </ct:Assign>
    </ct:Variable>

    <PKmacros>
        <Compartment>
            <Value argument="cmt">
                <ct:Int>1</ct:Int>
            </Value>
            <Value argument="amount">
                <ct:SymbRef symbIdRef="Ac"/>
            </Value>
            <Value argument="volume">
                <ct:SymbRef blkIdRef="pm1" symbIdRef="V"/>
            </Value>
        </Compartment>
        <IV>
            <Value argument="adm">
                <ct:Int>1</ct:Int>
            </Value>
            <Value argument="cmt">
                <ct:Int>1</ct:Int>
            </Value>
        </IV>
        <Elimination>
            <Value argument="cmt">
                <ct:Int>1</ct:Int>
            </Value>
            <Value>
                <ct:SymbRef blkIdRef="pm1" symbIdRef="k"/>
            </Value>
        </Elimination>
    </PKmacros>
</StructuralModel>
```

with the re-paramerization encoded in the `<ParameterModel>`, `pm1`, as the following code shows

```
<ParameterModel blkId="pm1">
    <SimpleParameter symbId="V"/>
    <SimpleParameter symbId="CL"/>
    <SimpleParameter symbId="k">
        <ct:Assign>
            <math:Equation>
                <math:Binop op="divide">
                    <ct:SymbRef symbIdRef="CL"/>
                    <ct:SymbRef symbIdRef="V"/>
                </math:Binop>
            </math:Equation>
        </ct:Assign>
    </SimpleParameter>
    <!-- omitted other parameters and IIV -->
```

9.2.5.6 Observation Model

In this example there are two observations for the continuous variables Cc and E, which are outputs from the structural model. As described above each has a residual error model applied to it.

The XML is very similar for both variables so in the following listing

```
<ObservationModel blkId="om1">
    <ContinuousData>
        <SimpleParameter symbId="a"/>
        <SimpleParameter symbId="b"/>
        <RandomVariable symbId="epsilon_Cc">
            <ct:VariabilityReference>
                <ct:SymbRef blkIdRef="vm2" symbIdRef="residual"/>
            </ct:VariabilityReference>
            <NormalDistribution xmlns="http://www.uncertml.org/3.0"
                definition="http://www.uncertml.org/distributions/normal">
                <mean><rVal>0</rVal></mean>
                <variance><prVal>1</prVal></variance>
            </NormalDistribution>
        </RandomVariable>
        <Standard symbId="Cc_obs">
            <Output>
                <ct:SymbRef blkIdRef="sm1" symbIdRef="Cc"/>
            </Output>
            <ErrorModel>
                <ct:Assign>
                    <math:Equation>
                        <math:FunctionCall>
                            <ct:SymbRef symbIdRef="combinedErrorModel"/>
                            <math:FunctionArgument symbId="a">
                                <ct:SymbRef symbIdRef="a"/>
                            </math:FunctionArgument>
                            <math:FunctionArgument symbId="b">
                                <ct:SymbRef symbIdRef="b"/>
                            </math:FunctionArgument>
                            <math:FunctionArgument symbId="f">
                                <math:Equation>
                                    <ct:SymbRef blkIdRef="sm1" symbIdRef="Cc"/>
                                </math:Equation>
                            </math:FunctionArgument>
                        </math:FunctionCall>
                    </math:Equation>
                </ct:Assign>
            </ErrorModel>
            <ResidualError>
                <ct:SymbRef symbIdRef="epsilon_Cc"/>
            </ResidualError>
        </Standard>
    </ContinuousData>
</ObservationModel>
```

we only show how the residual error model for Cc, the continuous PK variable, is defined. First the parameters of the residual error model a and b are defined using `<SimpleParameter>`; here a combined additive and proportional error model is applied. Then *epsilon_Cc* as `<RandomVariable>` of the residual error model with a mean of 0 and standard deviation *sigma_Cc* is defined to be used in the subsequent parts of the model. Note that we need to indicate the type and level of the according random effect which were defined in the 'Variability Model' section above. This is done using the `<VariabilityReference>`

The subsequent element `<Standard>` states that we are defining a standard continuous observation model. This means in short that we can define an observation model of the form $y_{ij} = f_{ij} + g \times \epsilon_{ij}$ (for details, see the discussion below and section 3.7).

The attribute `symbId` of `<Standard>` defines the name of the variable that will hold the result of the applied residual error model. The `<Output>` element then defines the variable in the structural model that the residual error model applies to (in this case Cc). Next we define the actual residual error model with the `<ErrorModel>` element. The error model is invoked by calling the function *combinedErrorModel* defined in the symbol definition at the beginning of the PharmML document (see section 9.2.4). We pass in the appropriate parameter values (see section 3.7.1.1 for a description of the residual error model) and this defines our residual error model.

Finally we use the `<ResidualError>` element to reference the *epsilon_Cc* specified before.

Residual model implementation Most residual error model types have two or three equivalent forms, by which we mean they have the same variance, although they use one or more residual errors, ϵ_{ij}, see examples in section 3.7.1.3. Other types contain two or more predictions from the structural model, f_{ij}. From a computational point of view it makes a lot of sense to reflect such differences in the language structure. This was the motivation to allow for the implementation of two types of observation models

- `<Standard>` – any observation model of the form

$$u(y_{ij}) = u(f_{ij}) + g \times \epsilon_{ij}$$

  which can be defined using exactly one of the following items

  - a transformation, u, e.g. *log* or *logit*
  - one structural model prediction, f_{ij}
  - one standard deviation function, g
  - one random variable, ϵ_{ij}

- `<General>` – using any number of the items listed above and arbitrary functional relationship between them.

This chapter contains more examples illustrating these constructs.

9.2.6 Trial Design

9.2.6.1 Structure

In this fairly simple case, there is only one epoch and four arms, *Arm_1*, *Arm_2* etc., cf. Figure 9.4. The resulting four cells *Cell_1*, *Cell_2* etc. each correspond to one arm and contain one segment. Figure 9.7 (right) illustrates the inter-relationship between cells, arms, epochs and segments. Here a segment consists of one activity – a certain type of treatment. In general, a segment can contain more than one activity. Dosing regimen can be defined for different compartments in the structural model. In this example there are four treatments with one dosing regimen per treatment.

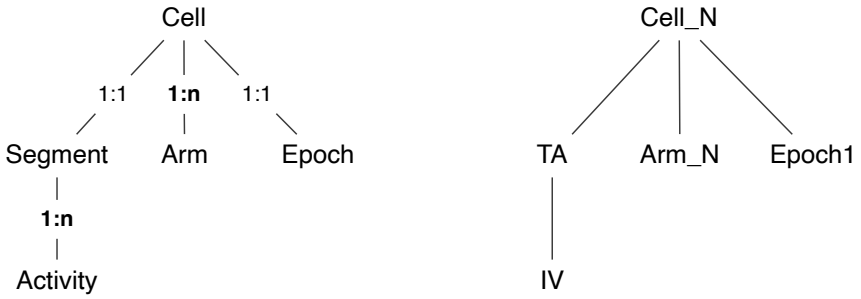


Figure 9.7: (left) General cell hierarchy; (right) How it is applied in example (1). There is only one epoch and four arms, *Arm_1*, *Arm_2* etc. The resulting four cells *Cell_1*, *Cell_2*, etc. each correspond to one arm and contain one segment. See the listing bellow for how these features are implemented and structured.

In the following listing

```
<Activity oid="d1">
    <Bolus>
        <DoseAmount inputTarget="derivativeVariable">
            <ct:SymbRef blkIdRef="sm1" symbIdRef="Ad"/>
            <ct:Assign>
                <Equation xmlns="http://www.pharmml.org/pharmml/0.6/Maths">
                    <Binop op="times">
                        <ct:Real>0.25</ct:Real>
                        <ct:SymbRef blkIdRef="cm1" symbIdRef="W"/>
                    </Binop>
                </Equation>
            </ct:Assign>
```

```
                </DoseAmount>
                <DosingTimes>
                    <ct:Assign>
                        <ct:Sequence>
                            <ct:Begin><ct:Int>0</ct:Int></ct:Begin>
                            <ct:StepSize><ct:Int>24</ct:Int></ct:StepSize>
                            <ct:End><ct:Int>192</ct:Int></ct:End>
                        </ct:Sequence>
                    </ct:Assign>
                </DosingTimes>
            </Bolus>
        </Activity>
```

we show an activity defined for *Arm_1*, here treatment in the form of a bolus administration: the others are very similar. The `<Activity>` element is given an identifier, "`d1`", and the element `<Bolus>` defines a bolus administration[4]. `<DoseAmount>` defines the amount of drug to be administered. The are four options here. The dose can be assigned to the variable defined by an ODE, such is the case here, for which we use `inputTarget="derivativeVariable"` attribute. Alternatively, the target can be the dosing variable D as in algebraic equations, an arbitrary model variable or in case we use PK macros the according administration type.

Additionally, in this case the administration is adjusted for body weight using the expression $0.25W$. The element `<SymbIdRef>` defines the target of dosing. This effectively defines an input function that adds the dose amount to the variable Ad at the specified dosing time. The dosing times themselves are given by the `<DosingTimes>` element as the sequence $0 : 24 : 192$.

Once the segments with their according treatments/activities are defined, we can define epochs using the element `<Epoch>` with their start and stopping time along with their order. After the arms are specified, the `<Cell>` and `<Segment>` elements are used to put all of the components together as the following this listing shows

```
    <Structure>
        <Epoch oid="e1">
            <Start>
                <ct:Real>0</ct:Real>
            </Start>
            <End>
                <ct:Real>300</ct:Real>
            </End>
            <Order>1</Order>
        </Epoch>
        <Arm oid="a1"/>
        <Cell oid="c1">
            <EpochRef oidRef="e1" />
            <ArmRef oidRef="a1"/>
            <SegmentRef oidRef="ta"/>
        </Cell>
        <Segment oid="ta">
            <ActivityRef oidRef="d1"/>
        </Segment>
```

9.2.6.2 Population

The `<Population>` is the second and in the case of a simulation example, also the last block in the trial design definition. As explained in chapter 4 this is the place to define the number of subjects per study arm, constant or time-varying covariates and other properties, if available. In this particular case covariates are simulated according to the definition in eq.9.3. This is the reason the following table with population data will have only three columns. (In case of estimation the covariates would have to be listed explicitly in this table as well, this will be described in the next example 9.5)

The definition of the table is in the `<IndividualTemplate>` block where the columns *id*, *arm* and *reps* are specified, see the following listing

```
    <Population>
        <ct:VariabilityReference>
            <ct:SymbRef blkIdRef="vm1" symbIdRef="indiv"/>
        </ct:VariabilityReference>
```

[4]Bolus and infusion are the only types of dosing regimen permitted. We make no distinction between oral and IV administration as such differences are handled by the structural model used.

```
        <ds:DataSet>
            <ds:Definition>
                <ds:Column columnId="id" columnType="id" valueType="id" columnNum="1"/>
                <ds:Column columnId="arm" columnType="arm" valueType="id" columnNum="2"/>
                <ds:Column columnId="reps" columnType="replicate" valueType="int" columnNum="3"/>
            </ds:Definition>
            <ds:Table>
                <ds:Row><ct:Id>i1</ct:Id><ct:Id>a1</ct:Id><ct:Int>20</ct:Int></ds:Row>
                <ds:Row><ct:Id>i2</ct:Id><ct:Id>a2</ct:Id><ct:Int>20</ct:Int></ds:Row>
                <ds:Row><ct:Id>i3</ct:Id><ct:Id>a3</ct:Id><ct:Int>40</ct:Int></ds:Row>
                <ds:Row><ct:Id>i4</ct:Id><ct:Id>a4</ct:Id><ct:Int>40</ct:Int></ds:Row>
            </ds:Table>
        </ds:DataSet>
    </Population>
```

Using the *reps* variable allows us to shorten the definition in cases where all other features are the same across subjects in an arm, which is the case here. Here four arms with 20, 20, 40 and 40 subjects, respectively, are defined. An alternative would be to list all subjects explicitly, in which case only two columns would be sufficient, *id* and *arm*. The attribute `columnType` informs the target tool about the meaning of each column to allow for the correct association of store data and model elements. The allowed values for this attribute has been listed in Table 6.6 in section 6.2.
In the estimation case another block `<IndividualDosing>` would have to be defined as next. This will be explained in the example 9.5.

9.2.7 NONMEM dataset

As explained before in the specification document, see for example sections 6.5 or 9.1, a NONMEM dataset carries the entire information about a trail design and any required numerical data, and is the default option used by the two main target tools, Monolix and NONMEM.

In such case, according to the schema in Figure 9.1, the only remaining model parts to be encoded are in `<SimulationStep>` of the `<ModellingSteps>` section, starting with the definition of the dataset and data-model mappings. The Table 9.3 shows the data for two subjects from the first and second study arm. The

ID	WT	ARM	TIME	DV	ORIG	AMT	MDV	EVID
1	60	1	0		.	0.25	1	1
1	60	1	0.5		1	.	0	0
1	60	1	4		1	.	0	0
...	...	...	...	...	...	...	...	...
1	60	1	20		1	.	0	0
1	60	1	24		.	0.25	1	1
1	60	1	24		1	.	0	0
1	60	1	24		2	.	0	0
...	...	...	...	...	...	...	...	...
1	60	1	250		1	.	0	0
1	60	1	264		2	.	0	0
1	60	1	288		2	.	0	0
...	...	...	...	...	...	...	...	...
21	72	2	0		.	0.5	1	1
21	72	2	0.5		1	.	0	0
21	72	2	4		1	.	0	0
...	...	...	...	...	...	...	...	...

Table 9.3: A dataset for first two subjects in *Arm1* and one subject in *Arm2* used in example 1.

next listing shows how to

- define the columns of the dataset in Table 9.3
- reference an external csv-dataset
- define the mapping between the columns and elements in the model

- scale the dose amount with respect to body weight.

```
<ExternalDataSet toolName="NONMEM" oid="nmOid">
    <ColumnMapping>
        <ds:ColumnRef columnIdRef="ID"/>
        <ct:SymbRef blkIdRef="vm1" symbIdRef="indiv"/>
    </ColumnMapping>
    <ColumnMapping>
        <ds:ColumnRef columnIdRef="TIME"/>
        <ct:SymbRef symbIdRef="t"/>
    </ColumnMapping>
    <ColumnMapping>
        <ds:ColumnRef columnIdRef="WT"/>
        <ct:SymbRef blkIdRef="cm1" symbIdRef="W"/>
    </ColumnMapping>

    <!-- Bodyweight scaled AMT, defined in transformation T1, is mapped to target Ad -->
    <ColumnMapping>
        <ds:ColumnRef columnIdRef="AMT" transformIdRef="T1"/>
        <ct:SymbRef blkIdRef="sm1" symbIdRef="Ad"/>
    </ColumnMapping>
    <ColumnTransformation transformId="T1">
        <math:Equation>
            <math:Binop op="times">
                <ds:ColumnRef columnIdRef="AMT"/>
                <ct:SymbRef blkIdRef="cm1" symbIdRef="W"/>
            </math:Binop>
        </math:Equation>
    </ColumnTransformation>
    <MultipleDVMapping>
        <ds:ColumnRef columnIdRef="DV"/>
        <!-- DV is mapped to 'Cc_obs' in observation model 'om2' if ORIG=1 -->
        <Piecewise>
            <math:Piece>
                <ct:SymbRef blkIdRef="om1" symbIdRef="E_obs"/>
                <math:Condition>
                    <math:LogicBinop op="eq">
                        <ds:ColumnRef columnIdRef="ORIG"/>
                        <ct:Real>1</ct:Real>
                    </math:LogicBinop>
                </math:Condition>
            </math:Piece>
            <!-- DV is mapped to E_obs in observation model 'om1' if ORIG=2 -->
            <math:Piece>
                <ct:SymbRef blkIdRef="om2" symbIdRef="Cc_obs"/>
                <math:Condition>
                    <math:LogicBinop op="eq">
                        <ds:ColumnRef columnIdRef="ORIG"/>
                        <ct:Real>2</ct:Real>
                    </math:LogicBinop>
                </math:Condition>
            </math:Piece>
        </Piecewise>
    </MultipleDVMapping>

    <ds:DataSet>
        <ds:Definition>
            <ds:Column columnId="ID" columnType="id" valueType="string" columnNum="1"/>
            <ds:Column columnId="WT" columnType="covariate" valueType="real" columnNum="2"/>
            <ds:Column columnId="ARM" columnType="arm" valueType="id" columnNum="3"/>
            <ds:Column columnId="TIME" columnType="idv" valueType="real" columnNum="4"/>
            <ds:Column columnId="DV" columnType="dv" valueType="real" columnNum="5"/>
            <ds:Column columnId="ORIG" columnType="undefined" valueType="real" columnNum="6"/>
            <ds:Column columnId="AMT" columnType="dose" valueType="int" columnNum="7"/>
            <ds:Column columnId="MDV" columnType="mdv" valueType="real" columnNum="8"/>
            <ds:Column columnId="EVID" columnType="evid" valueType="real" columnNum="9"/>
        </ds:Definition>
        <ds:ImportData oid="dataOid">
            <ds:path>example1.csv</ds:path>
            <ds:format>CSV</ds:format>
            <ds:delimiter>COMMA</ds:delimiter>
        </ds:ImportData>
```

```
        </ds:DataSet>
    </ExternalDataSet>
```

Note, that although the listing starts with column mapping and scaling it is recommended to start with the dataset and the external file reference definitions, using `<Definion>` and `<ImportData>` with `<DataSet>`. Once this is done, using the attribute `columnType` as described in Section 6.6.2, the mappings are straightforward. The basic idea is to specify the first the column we want to map, with `<ColumnRef>` and then the according target element in the model with `<SymbRef>`. For example, the column *TIME* is mapped with the symbol t as defined by the `<IndependentVariable>` element at the beginning of the model file. The ID column doesn't have to be mapped, the attribute `columnType="id"`, the *subject identifiers*, assigns the proper meaning to this symbol, see Table 6.2.

The dose scaling with respect to the body weight, as described in the table in Section 9.2.2, is defined in the `<ColumnTransformation>`. One needs to assign a user-defined value to the attribute `transformId` of the element, here e.g. *T1*, which is then referred to when mapping the *AMT* column.

The last column mapping we will look is a conditional mapping which applies to situations when the dataset contains multiple observations in one and the same DV column. In such cases one needs an additional column, here ORIG to point the target tool to correct value. In this example two different DV's are stored and the column ORIG carries the numbers to facilitate the mapping of DV values to the appropriate targets, i.e. 1 to identify *E_obs* from the `<ObservationModel>` `om1` and 2 to identify *Cc_obs* from `om2`.

For this purpose we use the element `<MultipleDVMapping>`. It works similar to the `<ColumnMapping>` tag in that its first child element refers to the appropriate dataset column. The second child element refers as before to the target variable but this time using a piecewise statement to condition the mapping based on the value of the ORIG column.

9.2.8 Modelling Steps

Independent whether the model file is equipped with a `<TrailDesign>` section or whether it sources the design from a NONMEM dataset, a modelling task needs to the defined. As mentioned before, right now, only two tasks are supported explicitly, estimation and simulation. Although a model can contain the description for multiple tasks, for now we limit the use cases to single tasks.

9.2.8.1 Simulation settings and dependencies

In the following listing

```
<ModellingSteps xmlns="http://www.pharmml.org/pharmml/0.6/ModellingSteps">
    <SimulationStep oid="s1">
        <!-- omitted initial values and observations -->
    </SimulationStep>
    <StepDependencies>
        <Step>
            <ct:OidRef oidRef="s1"/>
        </Step>
    </StepDependencies>
</ModellingSteps>
```

you can see the structure of the `<ModellingSteps>` section of PharmML. In this example we are describing a simulated model and so use the `<SimulationStep>` element.

The first part of the simulation block sets initial values for parameters in the model and defines the outputs of the simulation. We will go into more detail on these elements below. Then at the end of the modelling steps block is the `<StepDependencies>` element. This describes the ordering of the steps in the modelling steps section (see section 6.2.3), but in this case it is redundant as we only have one step in this example.

9.2.8.2 Initial Values

The code snippet in listing

```
        <ct:VariableAssignment>
            <ct:SymbRef blkIdRef="c1" symbIdRef="pop_W"/>
            <ct:Assign>
                <ct:Real>70.07</ct:Real>
            </ct:Assign>
        </ct:VariableAssignment>
```

```
<ct:VariableAssignment>
    <ct:Description>omega_W = 1409/100 assignment</ct:Description>
    <ct:SymbRef blkIdRef="c1" symbIdRef="omega_W"/>
    <ct:Assign>
        <math:Equation>
            <math:Binop op="divide">
                <ct:Real>1409</ct:Real>
                <ct:Real>100</ct:Real>
            </math:Binop>
        </math:Equation>
    </ct:Assign>
</ct:VariableAssignment>
<ct:VariableAssignment>
    <ct:SymbRef blkIdRef="p1" symbIdRef="pop_ka"/>
    <ct:Assign>
        <ct:Real>1</ct:Real>
    </ct:Assign>
</ct:VariableAssignment>
```

shows how we set initial values. Very simply we refer to a previously defined variable, here the typical value for a covariate pop_W, or parameter and then assign it a numerical value within the `<VariableAssignment>` element. In this example the value for $omega_W$ is calculated from a mathematical expression, which is allowed, if this expression resolves to a numerical value. The order of the `<VariableAssignment>` elements is not significant and the ordering is based on variable dependencies (as described in section 8.2 on page 86).

9.2.8.3 Observations

The following applies to the case when the trial design is explicitly encoded in PharmML, in the NONMEM dataset driven scenario the observations are defined in the dataset as described in Section 9.2.7.

Typically, what drives a simulation task are its outputs. One needs to simulate the time courses of the variables of interest at well specified time points. In PharmML the `<Observations>` element where this is defined, as can be seen in this listing

```
<Observations>
    <Timepoints>
        <ct:Vector>
            <ct:VectorElements>
                <ct:Real>0.5</ct:Real>
                <ct:Sequence>
                    <ct:Begin><ct:Int>4</ct:Int></ct:Begin>
                    <ct:StepSize><ct:Int>4</ct:Int></ct:StepSize>
                    <ct:End><ct:Int>48</ct:Int></ct:End>
                </ct:Sequence>
                <ct:Sequence>
                    <ct:Begin><ct:Int>52</ct:Int></ct:Begin>
                    <ct:StepSize><ct:Int>24</ct:Int></ct:StepSize>
                    <ct:End><ct:Int>192</ct:Int></ct:End>
                </ct:Sequence>
                <ct:Sequence>
                    <ct:Begin><ct:Int>192</ct:Int></ct:Begin>
                    <ct:StepSize><ct:Int>4</ct:Int></ct:StepSize>
                    <ct:End><ct:Int>250</ct:Int></ct:End>
                </ct:Sequence>
            </ct:VectorElements>
        </ct:Vector>
    </Timepoints>
    <Continuous>
        <ct:SymbRef blkIdRef="sm1" symbIdRef="Cc"/>
        <ct:SymbRef blkIdRef="om2" symbIdRef="Cc_obs"/>
    </Continuous>
</Observations>
```

In this example we define, using the `<Vector>` construct explained in section 6.9, a set of time points, $0.5, 4:4:48, 52:24:192, 192:4:250$, and the variables we would like to see simulated at those points in time. One or more output variables can be defined here using the `<Continuous>` element. It is noteworthy here that by choosing the Cc and Cc_{obs} defined in the structural model, `sm1`, and observation model, `om1`, blocks, respectively, we can instruct the target tool to output the results from the structural model and the observation model.

9.3 Example 2: Simulation with steady state dosing

9.3.1 Description

The following example is taken from [Bonate, 2011], p.535, and represents another simple case of a PK simulation[5]. However, here we discuss a system under steady state resulting from a twice daily dosing in 50 adult subjects who received a dose of 100 mg per administration. The drug concentration follows a 1-comp model with first order absorption with a proportional residual error model. The only essential new aspect

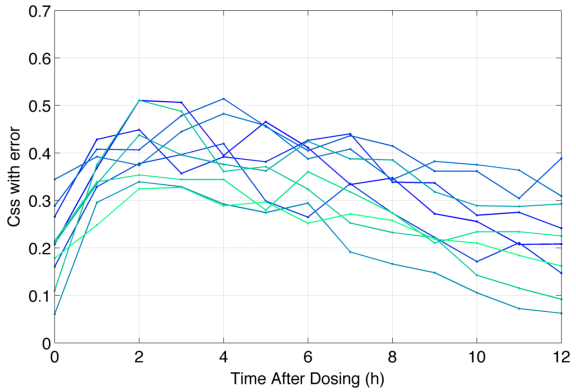


Figure 9.8: Simulated PK model as defined in the example for 10 subjects.

compared to the previous example is the fact that we have here so called steady-state administration. For this to be defined one needs to provide the time point of the last dosing event and the dose interval.

9.3.2 Model Definition

9.3.2.1 Variability model

The variability structure is identical to that in the previous example – there is only one level of subject related variability, see Figure 9.5.

9.3.2.2 Parameter model

The model uses the following parameters:

$$\begin{aligned} \theta_{1,i} &= pop_{\theta_1} + \eta_{\theta_1,i} \\ \log(V_i) &= \log(pop_V) + \eta_{V,i} \\ \theta_2 &= 0.75 \\ CL_i &= \theta_{1,i}\left(\frac{W_i}{70}\right)^{\theta_2} \\ K_a &= 0.5 \end{aligned}$$

where

$$\eta_{\theta_{1,i}} \sim N(0, \omega_{\theta_1}), \quad \eta_{V,i} \sim N(0, \omega_V)$$

and with

$$pop_{\theta_1} = 25, \quad \omega_{\theta_1} = 5 \quad pop_V = 250, \quad \omega_V = 100.$$

9.3.2.3 Covariate model

Body weight is the only covariate used in this model. It is used in the model for the individual clearance only.

[5]The example is encoded in two versions, `example2.xml` and `example2_NONMEM.xml`, with explicit encoded trial design and design sourced from a NONMEM datafile, respectively.

	Weight
Type	Continues
Transformation	$(W/70)^{\theta_2}$
Distribution	Normal
Mean, pop_W	80
Standard deviation, ω_W	9.6

Table 9.4: Covariates overview.

9.3.2.4 Structural model

$$k = \frac{CL}{V}$$
$$C_{SS}(t) = \frac{D}{V}\frac{K_a}{K_a - k}\left(\frac{e^{-k(t-t_D)}}{1-e^{-k\tau}} - \frac{e^{-K_a(t-t_D)}}{1-e^{-K_a\tau}}\right) \tag{9.8}$$

9.3.2.5 Observation model

We apply a residual error models to the output variable C_{SS}.

Output Variable	C_{SS}
Observations Name	Concentration
Units	mg/l
Observations Type	Continuous
Residual Error Model	Proportional
Error Model Parameters	$b = 0.1$

9.3.2.6 Trial design

Table below summarises the information about the design in this example.

Arm	**1**
Number of subjects	50
Dose variable	D
Dosing Amount	100
Dose Units	mg
Dose per kg	no
Dosing times (h)	0
Dose intervals (h)	12

9.3.3 Simulation Step

The concentration C_{SS} is going to be read out at equidistant time points after the dosing:

Output Variable	C_{SS}
Observation times	0,1,2,3,4,5,6,7,8,9,10,11,12

9.3.4 Structural model

In the last example we defined the structural model by using an ODE system. Here, we implement an algebraic formula for the calculation of the drug concentration in steady-state, C_{SS}, as shown in the following listing

```
<ct:Variable symbolType="real" symbId="Css">
    <ct:Assign>
        <Equation xmlns="http://www.pharmml.org/pharmml/0.6/Maths">
```

```
                    <Binop op="times">
                        <Binop op="divide">
                            <ct:SymbRef symbIdRef="D"/>
                            <ct:SymbRef blkIdRef="pm1" symbIdRef="V"/>
                        </Binop>
                        <Binop op="times">
                            <Binop op="divide">
                                <ct:SymbRef blkIdRef="pm1" symbIdRef="Ka"/>
                                <Binop op="minus">
                                    <ct:SymbRef blkIdRef="pm1" symbIdRef="Ka"/>
                                    <ct:SymbRef symbIdRef="k"/>
                                </Binop>
                            </Binop>
                            <Binop op="minus">
                                <Binop op="divide">
                                    <Uniop op="exp">
                                        <Binop op="times">
                                            <Uniop op="minus">
                                                <ct:SymbRef symbIdRef="k"/>
                                            </Uniop>
                                            <Binop op="minus">
                                                <ct:SymbRef symbIdRef="t"/>
                                                <ct:SymbRef symbIdRef="tD"/>
                                            </Binop>
                                        </Binop>
                                    </Uniop>
                                    <Binop op="minus">
                                        <ct:Real>1</ct:Real>
                                        <Uniop op="exp">
                                            <Binop op="times">
                                                <Uniop op="minus">
                                                    <ct:SymbRef symbIdRef="k"/>
                                                </Uniop>
                                                <ct:SymbRef blkIdRef="pm1" symbIdRef="tau"/>
                                            </Binop>
                                        </Uniop>
                                    </Binop>
                                </Binop>
                                <Binop op="divide">
                                    <Uniop op="exp">
                                        <Binop op="times">
                                            <Uniop op="minus">
                                                <ct:SymbRef blkIdRef="pm1" symbIdRef="Ka"/>
                                            </Uniop>
                                            <Binop op="minus">
                                                <ct:SymbRef symbIdRef="t"/>
                                                <ct:SymbRef symbIdRef="tD"/>
                                            </Binop>
                                        </Binop>
                                    </Uniop>
                                    <Binop op="minus">
                                        <ct:Real>1</ct:Real>
                                        <Uniop op="exp">
                                            <Binop op="times">
                                                <Uniop op="minus">
                                                    <ct:SymbRef blkIdRef="pm1" symbIdRef="Ka"/>
                                                </Uniop>
                                                <ct:SymbRef blkIdRef="pm1" symbIdRef="tau"/>
                                            </Binop>
                                        </Uniop>
                                    </Binop>
                                </Binop>
                            </Binop>
                        </Binop>
                    </Binop>
                </Equation>
            </ct:Assign>
        </ct:Variable>
```

Consequently, we use for C_{SS} the `<Variable>`, instead of `<DerivativeVariable>`, element as in the previous example which is of `real` type.

9.3.5 Trial design model

9.3.5.1 Structure

Figure 9.9 shows the *Structure* of this simple example consisting of 1 arm and one epoch, meaning one treatment type for everybody.

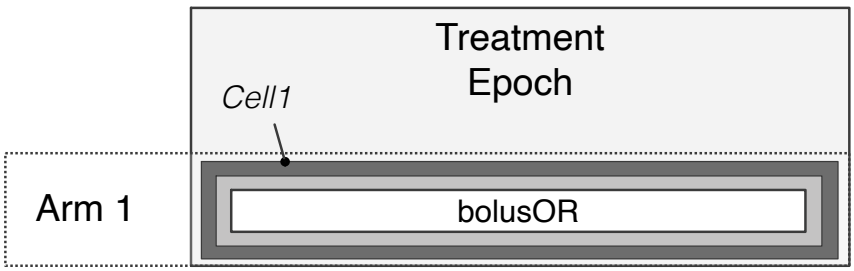


Figure 9.9: Design overview: this study consists of one arm and one epoch.

Segment	Activity	Treatment	DoseTime	DoseSize	Target Variable
TA	bolusOR	OR bolus	0	100	D

Table 9.5: Segment/activity overview.

Epoch	Start time	End time
Treatment Epoch	0	12

Table 9.6: Epoch definition – there is only one epoch here.

While the implementation of epoch, arm, cell and segment is analog to that in the previous example, the `<Activity>` element contains new items. After we defined `<DoseAmount>` as before as well, the steady-state administration is easily implemented using the `<SteadyState>` with last doing event as `<EndTime>` and the dose interval as `<Interval>` as can be seen in the following listing

```
<Activity oid="bolusOR">
    <Bolus>
        <DoseAmount inputTarget="parameter">
            <ct:SymbRef blkIdRef="sm1" symbIdRef="D"/>
            <ct:Assign>
                <ct:Real>100</ct:Real>
            </ct:Assign>
        </DoseAmount>
        <SteadyState>
            <EndTime>
                <ct:SymbRef blkIdRef="sm1" symbIdRef="tD"/>
                <ct:Assign>
                    <ct:Real>0</ct:Real>
                </ct:Assign>
            </EndTime>
            <Interval>
                <ct:SymbRef blkIdRef="pm1" symbIdRef="tau"/>
                <ct:Assign>
                    <ct:Real>12</ct:Real>
                </ct:Assign>
            </Interval>
        </SteadyState>
    </Bolus>
</Activity>
```

9.3.6 NONMEM dataset

Table 9.7 shows the typical NONMEM dataset to be used with this model for few selected subject. The subsequent listing shows then its definition and mappings

ID	TIME	WT	DV	EVID	MDV
1	0	82		0	0
1	1	82		0	0
1	2	82		0	0
...	...	...	...	...	...
1	11	82		0	0
1	12	82		0	0
2	0	77		0	0
2	1	77		0	0
2	2	77		0	0
...	...	...	...	...	...
2	11	77		0	0
2	12	77		0	0
3	0	94		0	0
...	...	...	...	...	...

Table 9.7: A dataset used in example 2. WT data has to be provided as sampled from normal distribution as specified in Table 9.4.

```
<ExternalDataSet toolName="NONMEM" oid="NMoid">

    <ColumnMapping>
        <ColumnRef xmlns="http://www.pharmml.org/pharmml/0.6/Dataset" columnIdRef="ID"/>
        <ct:SymbRef blkIdRef="vm1" symbIdRef="indiv"/>
    </ColumnMapping>
    <ColumnMapping>
        <ColumnRef xmlns="http://www.pharmml.org/pharmml/0.6/Dataset" columnIdRef="TIME"/>
        <ct:SymbRef symbIdRef="t"/>
    </ColumnMapping>
    <ColumnMapping>
        <ColumnRef xmlns="http://www.pharmml.org/pharmml/0.6/Dataset" columnIdRef="WT"/>
        <ct:SymbRef blkIdRef="cm1" symbIdRef="Weight"/>
    </ColumnMapping>
    <ColumnMapping>
        <ColumnRef xmlns="http://www.pharmml.org/pharmml/0.6/Dataset" columnIdRef="DV"/>
        <ct:SymbRef blkIdRef="om1" symbIdRef="Css_obs"/>
    </ColumnMapping>

    <DataSet xmlns="http://www.pharmml.org/pharmml/0.6/Dataset">
        <Definition>
            <Column columnId="ID" columnType="id" valueType="string" columnNum="1"/>
            <Column columnId="TIME" columnType="idv" valueType="real" columnNum="2"/>
            <Column columnId="WT" columnType="covariate" valueType="real" columnNum="3"/>
            <Column columnId="DV" columnType="dv" valueType="real" columnNum="4"/>
            <Column columnId="EVID" columnType="evid" valueType="int" columnNum="5"/>
            <Column columnId="MDV" columnType="mdv" valueType="int" columnNum="6"/>
        </Definition>
        <ImportData oid="dataOid">
            <path>example2.csv</path>
            <format>CSV</format>
            <delimiter>COMMA</delimiter>
        </ImportData>
    </DataSet>
</ExternalDataSet>
```

Note, that the sampling distribution defined in the covariate model cannot be used, this applies to situations when e.g. NONMEM is the target tool, and the dataset needs to provide the sampled normal distributed weight values for each subject. This is why the column WT is in the dataset, Table 9.7. This continues covariate is mapped to the `<CovariateModel>` which shorter then the one before and contains

only the allometric transformation as the following snippet shows

```
<CovariateModel blkId="cm1">
    <Covariate symbId="Weight">
        <Continuous>
            <Transformation>
                <TransformedCovariate symbId="theta2Weight"/>
                <Equation xmlns="http://www.pharmml.org/pharmml/0.6/Maths">
                    <Binop op="power">
                        <Binop op="divide">
                            <ct:SymbRef symbIdRef="Weight"/>
                            <ct:Real>70</ct:Real>
                        </Binop>
                        <ct:SymbRef blkIdRef="pm1" symbIdRef="theta2"/>
                    </Binop>
                </Equation>
            </Transformation>
        </Continuous>
    </Covariate>
</CovariateModel>
```

Here the dose or dosing time are not stored in the dataset, they can be defined and assigned in the `<StructuralModel>` and `<SimulationStep>`, respectively This corresponds to the situation when these data together with the structural model are implemented in the *$PRED* section in NMTRAN.

9.4 Example 3: Estimation, Warfarin PK

9.4.1 Description

This model describes the PK of warfarin[6].

9.4.1.1 Structural model

The model is a one compartment model with first-order absorption with lag time and first-order elimination.

$\boldsymbol{D}$ Dosing variable.

$\boldsymbol{t_D}$ Time of the dose.

$\boldsymbol{C}$ Concentration of drug in the compartment.

$$k = \frac{CL}{V}$$

$$C(t) = \begin{cases} 0 & \text{if} \quad t - t_D < T_{lag} \\ \frac{D}{V}\frac{k_a}{k_a - k}\left[e^{-k\,(t-t_D-T_{lag})} - e^{-k_a\,(t-t_D-T_{lag})}\right] & \text{otherwise} \end{cases}$$

9.4.1.2 Covariate model

Body weight, W, is the only continue covariate used in this model. It is used in the model for the individual clearance and volume.

	Weight
Type	Continues
Transformation	$\log(W/70)$

Table 9.8: Covariates overview.

[6]The example is encoded in two versions, `example3.xml` and `example3_NONMEM.xml`, with explicit encoded trial design and design sourced from a NONMEM datafile, respectively.

9.4.1.3 Parameters

PK Parameters The following PK parameters are used in the model:

T_{lag} The lag time.

ka The absorption rate constant.

V The volume of distribution.

CL Clearance of elimination.

The parameters are defined as follows:

$$\begin{aligned}\log(T_{lag}) &= \log(pop\_T_{lag}) + \eta_{T_{lag}}\\ \log(ka) &= \log(pop\_ka) + \eta_{ka}\\ \log(V) &= \log(pop\_V) + \beta_{1,V}\log(W_i/70) + \eta_V\\ \log(CL) &= \log(pop\_CL) + \beta_{1,CL}\log(W_i/70) + \eta_{CL}\end{aligned}$$

where

$$\begin{aligned}\eta_{T_{lag}} \sim \mathcal{N}(0,\omega_{T_{lag}}), \quad \eta_{ka} \sim \mathcal{N}(0,\omega_{ka}),\\ \eta_V \sim \mathcal{N}(0,\omega_V), \quad \eta_{CL} \sim \mathcal{N}(0,\omega_{CL})\end{aligned}$$

Note please that, in this case, $\beta_{1,V} = 0.75$ and $\beta_{1,CL} = 1$, i.e. are fixed and will not be estimated.

Variance-covariance matrix The full variance-covariance matrix for the random effects is :

$$\Omega = \begin{pmatrix} \omega^2_{Tlag} & 0 & 0 & 0\\ & \omega^2_{ka} & 0 & 0\\ & & \omega^2_V & 0\\ & & & \omega^2_{CL}\end{pmatrix}$$

9.4.1.4 Observation model

We apply a residual error models to the output variable C.

Output Variable	***C***
Observations Name	Concentration
Units	mg/l
Observations Type	Continuous
Residual Error Model	Combined2
Error Model Parameters	$a = 0.1, \quad b = 0.1$

9.4.1.5 Trial Design

The dosing regimen for the trial is given below — there is only one for each arm. Note that all dosing is bolus dosing (discrete administration at specific times) and all doses are administered to the same compartment.

Arm	**1**
Number of subjects	33
Dose variable	D
Dosing Amount	100
Dose Units	mg
Dose per kg	no
Dosing times (h)	0

9.4.1.6 Modelling Steps

The observations for the output variable is shown below. These time-points correspond to those define in the data-file. The task to be performed is a parameter estimation which will involved the following steps:

- Estimation of population paramaters.
- Estimation of Fisher information matrix.
- Estimation of the individual parameters.

Output Variable	*C*
Observation times	0.5,1,2,3,6,9,24,36,48,72,96,120

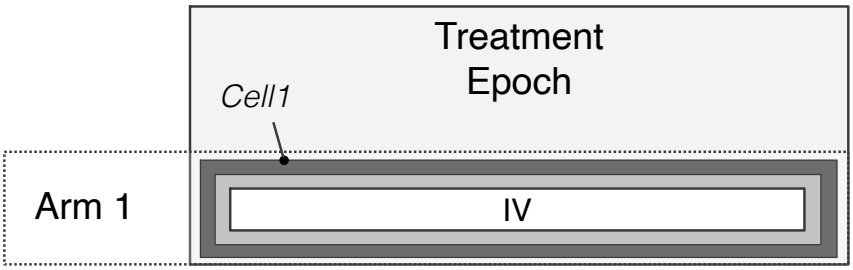


Figure 9.10: Design overview: this study consists of one arm and one epoch.

Figure 9.10 shows the *Structure* of this simple example consisting of 1 arm and one epoch, meaning one treatment type for everybody.

9.4.2 Overview

This our first estimation case. From figure 9.2 it follows that we can expect significant differences in the Trial Design and Modelling Steps sections compared to the previous simulation case.

Accordingly, the Model Definition is very similar to that in the previous example. The main PharmML feature we have not seen previously is the algebraic structural model (i.e., the model is not defined as a system of ODEs). The XML is too long to show here and is similar to the examples shown previously (section 6.8). If you are interested then please consult the full example associated with this specification.

We will start with the description of the Trail design. Note that because every subject receives the same dosing regimen, this can be encoded in the `<Activity>` block. Otherwise we would have to define the individual dosing regimens in the `<IndividualDosing>` element, see section 9.6.2.3 in example 9.6 how this is done.

9.4.3 Trial Design

9.4.3.1 Structure

As explained in the chapter 4 on trial design, we base the following structure on the CDISC standard Study Design Model [CDISC consortium, 2011]. The design elements are contained in the `<Structure>` block and you can see in following listing

```
<Structure>
    <Epoch oid="epoch1">
        <Start><ct:Real>0</ct:Real></Start>
        <End><ct:Real>180</ct:Real></End>
        <Order>1</Order>
    </Epoch>
    <Arm oid="arm1"/>
    <Cell oid="cell1">
        <EpochRef oidRef="epoch1"/>
        <ArmRef oidRef="arm1"/>
        <SegmentRef oidRef="segment1"/>
    </Cell>
```

```
        <Segment oid="segment1">
            <ActivityRef oidRef="d1"/>
        </Segment>
        <Activity oid="d1">
            <Bolus>
                <DoseAmount inputTarget="parameter">
                    <ct:SymbRef blkIdRef="sm1" symbIdRef="D"/>
                    <ct:Assign>
                        <ct:Real>100</ct:Real>
                    </ct:Assign>
                </DoseAmount>
                <DosingTimes>
                    <ct:SymbRef blkIdRef="sm1" symbIdRef="tD"/>
                    <ct:Assign>
                        <ct:Real>0</ct:Real>
                    </ct:Assign>
                </DosingTimes>
            </Bolus>
        </Activity>
    </Structure>
```

how the study is constructed of a single epoch, with a single arm and a single cell that contains a single segment. Note, though that this structure is not hierarchical and the `<Cell>` element joins the arm, epoch and segments together.

The last section of the structure, the `<Activity>` element, is of interest for the discussion. This is because, as already mentioned above, the administration and dosing regimen is identical for every patient. The dose amount is $D = 100\ mg$ and the dose time is $t_D = 0$. As in the previous example, the structural model is defined using an algebraic function with the dosing variable D which means the `<DoseAmount>` element has the attribute `inputType="parameter"`. Additionally the dosing time variable tD is referenced here and initialised.

9.4.3.2 Population

This is the place where we describe the individuals in the study, which *Arm* they belong to and any possible individual characteristics, such as body weight, age and other covariates. In this example we only know the body weight of the subjects. We define the known attributes of all individuals using the `<IndividualTemplate>` and then map each individual to this template using a `<Dataset>`. In the following listing

```
    <Population>
        <ColumnMapping>
            <ds:ColumnRef columnIdRef="WEIGHT"/>
            <ct:SymbRef blkIdRef="cm1" symbIdRef="W"/>
        </ColumnMapping>

        <ds:DataSet>
            <ds:Definition>
                <ds:Column columnId="ID" columnType="id" valueType="string" columnNum="1"/>
                <ds:Column columnId="ARM" columnType="arm" valueType="id" columnNum="2"/>
                <ds:Column columnId="WEIGHT" columnType="covariate" valueType="real" columnNum="3"/>
            </ds:Definition>
            <ds:Table>
                <ds:Row><ct:String>1</ct:String><ct:Id>arm1</ct:Id><ct:Real>70.1</ct:Real></ds:Row>
                <ds:Row><ct:String>2</ct:String><ct:Id>arm1</ct:Id><ct:Real>60.0</ct:Real></ds:Row>
                <ds:Row><ct:String>3</ct:String><ct:Id>arm1</ct:Id><ct:Real>93.2</ct:Real></ds:Row>
                <ds:Row><ct:String>4</ct:String><ct:Id>arm1</ct:Id><ct:Real>85.7</ct:Real></ds:Row>
                <ds:Row><ct:String>5</ct:String><ct:Id>arm1</ct:Id><ct:Real>78.3</ct:Real></ds:Row>
                <!-- SNIP -->
                <ds:Row><ct:String>33</ct:String><ct:Id>arm1</ct:Id><ct:Real>94.1</ct:Real></ds:Row>
            </ds:Table>
        </ds:DataSet>
    </Population>
```

you can see how this is implemented in PharmML. Column *2* in the table is equal for every subject because they all belong to one arm, here denoted as *a1*.

9.4.4 NONMEM dataset

Now we will describe the case when the data and trial design are sourced from the NONMEM dataset. Table 9.9 show a typical dataset required for an estimation task.

ID	TIME	WT	AMT	DVID	DV	MDV
1	0	66.7	100	0	.	1
1	0.5	66.7	.	1	0	0
1	1	66.7	.	1	1.9	0
1	2	66.7	.	1	3.3	0
1	3	66.7	.	1	6.6	0
...	...	...	...	...	...	...
1	120	66.7	.	1	0.8	0
2	0	80	100	0	.	1
2	0.5	80	.	1	9.2	0
2	1	80	.	1	8.5	0
2	2	80	.	1	6.4	0
2	3	80	.	1	4.8	0
...	...	...	...	...	...	...

Table 9.9: A fragment of the dataset used in example 3 for two first subjects.

The following code shows how the dataset definition and column mappings are done

```
<ExternalDataSet toolName="NONMEM" oid="NMoid">

    <ColumnMapping>
        <ds:ColumnRef columnIdRef="ID"/>
        <ct:SymbRef blkIdRef="vm1" symbIdRef="indiv"/>
    </ColumnMapping>
    <ColumnMapping>
        <ds:ColumnRef columnIdRef="TIME"/>
        <ct:SymbRef symbIdRef="t"/>
    </ColumnMapping>
    <ColumnMapping>
        <ds:ColumnRef columnIdRef="WT"/>
        <ct:SymbRef blkIdRef="cm1" symbIdRef="W"/>
    </ColumnMapping>
    <ColumnMapping>
        <ds:ColumnRef columnIdRef="DV"/>
        <ct:SymbRef blkIdRef="om1" symbIdRef="C_obs"/>
    </ColumnMapping>
    <ColumnMapping>
        <ColumnRef xmlns="http://www.pharmml.org/pharmml/0.6/Dataset" columnIdRef="AMT"/>
        <ct:SymbRef blkIdRef="sm1" symbIdRef="D"/>
    </ColumnMapping>
    <ColumnMapping>
        <ColumnRef xmlns="http://www.pharmml.org/pharmml/0.6/Dataset" columnIdRef="TIME"/>
        <Piecewise xmlns="http://www.pharmml.org/pharmml/0.6/Dataset">
            <math:Piece>
                <ct:SymbRef blkIdRef="sm1" symbIdRef="tD"/>
                <math:Condition>
                    <math:LogicBinop op="neq">
                        <ColumnRef columnIdRef="AMT"/>
                        <ct:Real>0</ct:Real>
                    </math:LogicBinop>
                </math:Condition>
            </math:Piece>
        </Piecewise>
    </ColumnMapping>
    <ds:DataSet>
        <ds:Definition>
            <ds:Column columnId="ID" columnType="id" valueType="string" columnNum="1"/>
            <ds:Column columnId="TIME" columnType="time" valueType="real" columnNum="2"/>
            <ds:Column columnId="WT" columnType="covariate" valueType="real" columnNum="3"/>
            <ds:Column columnId="AMT" columnType="dose" valueType="real" columnNum="4"/>
```

```
                <ds:Column columnId="DVID" columnType="dvid" valueType="real" columnNum="5"/>
                <ds:Column columnId="DV" columnType="dv" valueType="real" columnNum="6"/>
                <ds:Column columnId="MDV" columnType="mdv" valueType="real" columnNum="7"/>
            </ds:Definition>
            <ds:ImportData oid="dataOid">
                <ds:path>example3.csv</ds:path>
                <ds:format>CSV</ds:format>
                <ds:delimiter>COMMA</ds:delimiter>
            </ds:ImportData>
        </ds:DataSet>
    </ExternalDataSet>
```

Contrary to the previous example, we store now the dosing data in the dataset and therefore require to provide the additioanl column mapping. The column AMT is mapped in the usual fashion to its target, the dose variable, D. The mapping of the dosing time variable, tD, is a bit more interesting as it requires a mapping of the $TIME$ column to variable tD only when the actual doing happens. This is the case when the $AMT \neq 0$. Which is what is encoded in the last `<ColumnMapping>` of the listing above using the `<Piecewise>` element.

9.4.5 Modelling Steps

9.4.5.1 Objective data

Remember that according to Figure 9.2 this element is required only when working with explicit trial design.

An advantage of PharmML is that we do not have to define the design in the data file. Instead of using NONMEM dataset as demonstrated in the previous section and after the structure of the trial is defined as above, see Section 9.4.3, we just need to encode the measured experimental data, here the time and the independent variable, the concentration values. To achieve that we define a table in `<Dataset>` with columns: *ID*, *time* and *dv* and populate it with given experimental values, see `<ObjectiveDataSet>` block in the following listing

```
<ObjectiveDataSet>
    <ColumnMapping>
        <ds:ColumnRef columnIdRef="time"/>
        <ct:SymbRef symbIdRef="t"/>
    </ColumnMapping>
    <ColumnMapping>
        <ds:ColumnRef columnIdRef="dv"/>
        <ct:SymbRef blkIdRef="om1" symbIdRef="C_obs"/>
    </ColumnMapping>
    <ds:DataSet>
        <ds:Definition>
            <ds:Column columnId="ID" columnType="id" valueType="string" columnNum="1"/>
            <ds:Column columnId="time" columnType="time" valueType="real" columnNum="2"/>
            <ds:Column columnId="dv" columnType="dv" valueType="real" columnNum="3"/>
        </ds:Definition>
        <ds:Table>
            <!-- SUBJECT 1 -->
            <ds:Row><ct:String>1</ct:String><ct:Real>0.5</ct:Real><ct:Real>0</ct:Real></ds:Row>
            <ds:Row><ct:String>1</ct:String><ct:Real>1</ct:Real><ct:Real>1.9</ct:Real></ds:Row>
            <ds:Row><ct:String>1</ct:String><ct:Real>2</ct:Real><ct:Real>3.3</ct:Real></ds:Row>
            <ds:Row><ct:String>1</ct:String><ct:Real>3</ct:Real><ct:Real>6.6</ct:Real></ds:Row>
            <ds:Row><ct:String>1</ct:String><ct:Real>6</ct:Real><ct:Real>9.1</ct:Real></ds:Row>
            <ds:Row><ct:String>1</ct:String><ct:Real>9</ct:Real><ct:Real>10.8</ct:Real></ds:Row>
            <!-- SUBJECT 2 -->
            <!-- SNIP -->
        </ds:Table>
    </ds:DataSet>
</ObjectiveDataSet>
```

Similarly to the situation before we have to make sure that these values are correctly mapped to variable used in the model which is implemented in the `<ColumnMapping>` element. The ID column doesn't have to be mapped, the attribute `columnType="id"`, the *subject identifiers*, assigns the proper meaning to this symbol, see Table 6.2. Here the *time* as in the data is mapped to model time t and the measured concentration is mapped to the variable C_{obs} as in the observation model.

9.4.5.2 Parameter estimation

In a parameter estimation you do not necessarily want to estimate all the parameters in your model or you may wish to define bounds within which your parameter should be estimated, or provide an initial estimate. The `<ParametersToEstimate>` element controls this. As you can see in the following listing

```
<ParameterEstimation>
    <ct:SymbRef blkIdRef="pm1" symbIdRef="pop_V"/>
    <InitialEstimate fixed="false">
        <ct:Real>10</ct:Real>
    </InitialEstimate>
</ParameterEstimation>
<ParameterEstimation>
    <ct:SymbRef blkIdRef="pm1" symbIdRef="omega_V"/>
    <InitialEstimate fixed="false">
        <ct:Real>1</ct:Real>
    </InitialEstimate>
</ParameterEstimation>
```

we use a `<ParameterEstimation>` element that refers to the parameter in the model definition. In its simplest form you can decide whether the parameter is to be estimated by setting the `fixed` attribute (false indicates the parameter should be estimated). If a parameter is not defined here, then it is assumed that it will not be estimated, in which case it would be assigned an initial value elsewhere in the PharmML document. One of the validation rules (see chapter 8) is that every parameter has to be initialised.

9.4.5.3 Step dependencies

Then at the end of the `<ModellingSteps>` block is the `<StepDependencies>` element. This describes the ordering of the steps in the modelling process, but in this case it is almost trivial as we only have one step in this example:

```
<mstep:StepDependencies>
    <mstep:Step>
        <ct:OidRef oidRef="estimStep1"/>
    </mstep:Step>
</mstep:StepDependencies>
```

9.5 Example 4: Estimation with IOV

9.5.1 Description

In this example we will look at a more complex trial design and a correspondingly complex variability model[7]. The model also includes categorical covariates, which is again something we have not encountered thus far. The example is based on example IOV1 from Monolix 4.1 (see [Lixoft, 2012] for a detailed description) and features a cross-over design and inter-occasion variability (see section 3.5). As before we will go through the key elements of the model before we look at the PharmML examples, but given the complex nature of the trial design we will describe that first then move onto the model definition.

Arm	**1**	**2**
Number of subjects	33	33
Dose variable	*D*	*D*
Dosing Amount	100	150
Dose Units	*mg*	*mg*
Dose per kg	no	no
Dosing times (h)	0	0

Table 9.10: Arms overview with dosing specification.

[7]The example is encoded in two versions, `example4.xml` and `example4_NONMEM.xml`, with explicit encoded trial design and design sourced from a NONMEM datafile, respectively.

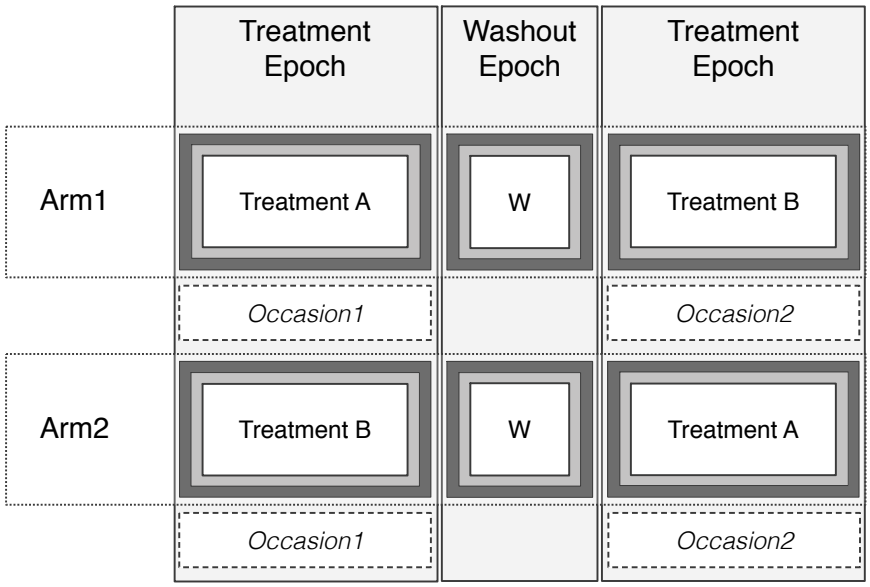


Figure 9.11: Schematic representation of a crossover design with washout. See tables 9.12 and 9.13 for the detailed definition of segments, cells, arms, epochs and occasions in this example.

9.5.1.1 Trial Design

The model features a basic crossover design (see Figure 9.11) with washout period and inter-occasion variability (IOV). There are two treatments and the subjects are organised into two arms that start with a different treatment. In between each treatment there is a washout period during which time the drug is eliminated from each subject. In the model the occasions, provide a second level of variability – IOV (see section 3.5). This is summarised in Figure 9.12 (see also the listing in section 9.5.3, showing relevant code within the element `<VariabilityModel>`).

The model also uses covariates to model the variability within the model and so the treatments, the sequence of treatments (i.e. treatments A, B or B,A) and the occasion itself are described in the covariate section below.

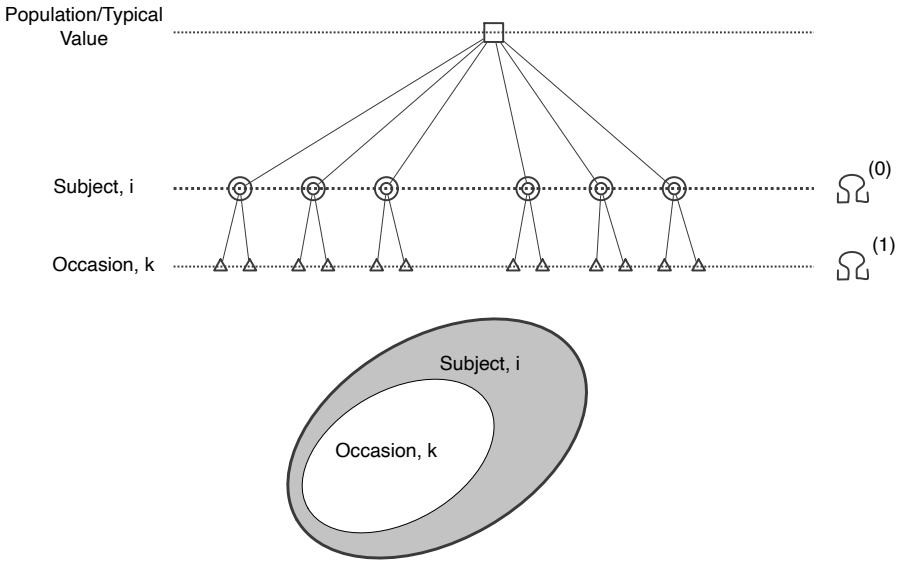


Figure 9.12: Two levels of variability – inter-individual and inter-occasion within individual variability.

9.5.1.2 Covariate Model

As discussed about all covariates are used to capture the explained variability in the model, see Table 9.11, and the *Occasion* additionally the unexplained expressed as the $\eta^{(+1)}$ random effects associated with CL and V.

	Sex	**Treat**	**TreatSeq**	**Occasion**
Type	Categorical	Categorical	Categorical	Categorical
Category Count	2	2	2	2
Categories	F, M	A, B	AB,BA	1, 2
Reference	F	A	AB	1

Table 9.11: Covariates overview.

9.5.1.3 Parameter Model

The parameter model includes random effects that represent the IIV and IOV levels of variability. It also relates the parameters to the covariates described above:

$$\log(ka_i) = \log(ka_{pop}) + \beta_{ka,TreatSeq} 1_{TreatSeq_i=AB} + \eta_{ka,i} \tag{9.9}$$

$$\begin{aligned}\log(V_{ik}) = {} & \log(V_{pop}) + \boldsymbol{\beta}_V 1_{S_i=F} + \beta_{V,OCC} 1_{OCC_{ik}=1} \\ & + \beta_{V,Treat} 1_{Treat_{ik}=A} + \beta_{V,TreatSeq} 1_{TreatSeq_i=AB} \\ & + \eta^{(0)}_{V,i} + \eta^{(+1)}_{V,ik}\end{aligned} \tag{9.10}$$

$$\begin{aligned}\log(CL_{ik}) = {} & \log(CL_{pop}) + \boldsymbol{\beta}_{CL} 1_{S_i=F} + \beta_{CL,OCC} 1_{OCC_{ik}=1} \\ & + \eta^{(0)}_{CL,i} + \eta^{(+1)}_{Cl,ik}\end{aligned}$$

where

$$\eta^{(0)}_{ka,i} \sim \mathcal{N}(0, \omega_{ka}), \quad \eta^{(0)}_{V,i} \sim \mathcal{N}(0, \omega_V), \quad \eta^{(0)}_{CL,i} \sim \mathcal{N}(0, \omega_{CL}),$$
$$\eta^{(+1)}_{V,ik} \sim \mathcal{N}(0, \gamma_V), \quad \eta^{(+1)}_{CL,ik} \sim \mathcal{N}(0, \gamma_{CL})$$

The full variance-covariance matrices for our model read

$$\Omega^{(0)} = \begin{pmatrix} \omega^2_{ka} & 0 & 0 \\ & \omega^2_V & 0 \\ & & \omega^2_{CL} \end{pmatrix} \tag{9.11}$$

$$\Omega^{(+1)} = \begin{pmatrix} 0 & 0 & 0 \\ & \gamma^2_V & 0 \\ & & \gamma^2_{CL} \end{pmatrix} \tag{9.12}$$

9.5.1.4 Structural model

The model is first order absorption with linear elimination. This is the equivalent to oral1_1cpt_kaVCl (model 8) from [Bertrand and Mentré, 2008, Appendix I].

9.5.1.5 Observation model

We apply a residual error models to the output variable C.

Output Variable	***C***
Observations Name	Concentration
Units	mg/l
Observations Type	Continuous
Residual Error Model	Combined
Error Model Parameters	$a = 0.1, \quad b = 0.1$

9.5.1.6 Modelling Steps

Compared to the last example, we have define here two tasks:

- Estimation of population paramaters.
- Estimation of the individual parameters.

9.5.2 Trial Design

We have summaries the dosing regimen and organisation of the trial design below, see also Figure 9.11.

Segment	Activity	Treatment	DoseTime	DoseSize	Target Variable
TA	OR1	OR bolus	0 : 12 : 72	150	D
TA	OR2	OR bolus	0 : 24 : 72	100	D

Table 9.12: Segment/activity overview.

Epoch	Occasion	Start time	End time
Treatment Epoch	OCC1	0	180
Washout	–	0	10
Treatment Epoch	OCC2	0	180

Table 9.13: Epoch and occasion definition.

9.5.2.1 Structure

The implementation of the treatments, in PharmML we use the `<Activity>` element, is different compared to the previous example. See Table 9.12 for the details. The difference is that now we have one dose administered at multiple dosing time points instead of single time point. See the following listing

```
<Activity oid="d1">
   <Bolus>
      <DoseAmount inputTarget="parameter">
         <ct:SymbRef blkIdRef="sm1" symbIdRef="D"/>
         <ct:Assign>
            <ct:Real>150</ct:Real>
         </ct:Assign>
      </DoseAmount>
      <DosingTimes>
         <ct:SymbRef blkIdRef="sm1" symbIdRef="tD"/>
         <ct:Assign>
            <ct:Sequence>
               <ct:Begin><ct:Real>0</ct:Real></ct:Begin>
               <ct:StepSize><ct:Real>12</ct:Real></ct:StepSize>
               <ct:End><ct:Real>72</ct:Real></ct:End>
            </ct:Sequence>
         </ct:Assign>
      </DosingTimes>
   </Bolus>
</Activity>
```

how one can describe it within the `<DosingTimes>` element using the `<Sequence>` structure defining the start/end times and step size.

Table 9.13 gives an overview of the *Epochs* and *Occasions* in this example. Here, the occasions overlap with the epochs, the start and end times are identical, this is not always the case, the occasions can span one or more epochs. The *Washout* epoch is given here with start/end times as well which is in fact a redundant piece of information (but required by construction of an *Epoch*) as a *Washout* always assumes total reset of all drug amounts.

As discussed in the section 4.2.1, in `<Structure>` block we encode the variability which is located below the subject (see the hierarchy of the random variability discussed in section 3.5). We call it the *inter-occasion variability*, IOV. The following listing

```
        <ObservationsEvent oid="occasions">
            <ArmRef oidRef="a1"/>
            <ArmRef oidRef="a2"/>
            <ct:VariabilityReference>
                <ct:SymbRef blkIdRef="vm1" symbIdRef="iov1"/>
            </ct:VariabilityReference>
            <ObservationGroup oid="occ1">
                <EpochRef oidRef="ep1"/>
            </ObservationGroup>
            <ObservationGroup oid="occ2">
                <EpochRef oidRef="ep3"/>
            </ObservationGroup>
        </ObservationsEvent>
    </Structure>
```

shows how this is done. In this case the occasions coincide with the epochs so we use the `<EpochRef>` element. Alternatively, we could use the `<Period>` element to define explicitly the start and end times of the occasions as shown in this listing:

```
        <!-- alternative -->
        <ObservationsEvent oid="occasions">
            <ArmRef oidRef="a1"/>
            <ArmRef oidRef="a2"/>
            <ct:VariabilityReference>
                <ct:SymbRef blkIdRef="vm1" symbIdRef="iov1"/>
            </ct:VariabilityReference>
            <ObservationGroup oid="occ1">
                <Period>
                    <Start><ct:Real>0</ct:Real></Start>
                    <End><ct:Real>180</ct:Real></End>
                </Period>
            </ObservationGroup>
            <ObservationGroup oid="occ2">
                <Period>
                    <Start><ct:Real>0</ct:Real></Start>
                    <End><ct:Real>180</ct:Real></End>
                </Period>
            </ObservationGroup>
        </ObservationsEvent>
    </Structure>
```

This is of course very useful if the occasions do not coincide with the epochs, or there are two or more occasions within one epoch. In this case we set the *Start* and *End* times to 0 and 180, respectively. These are exactly the same time points as are used in the epoch definition (see the first listing in section 9.4.3 for how to encode epochs in the `<Structure>` definition).

9.5.2.2 Population

We pick up where we left off in the `<Structure>`, implementing the hooks to the variability structure. The aspect we have not covered yet is related to IIV. The `<Population>` element is the place to define any subject related variability and those levels above it. The following listing shows how this works

```
    <Population>
        <ct:VariabilityReference>
            <ct:SymbRef blkIdRef="vm1" symbIdRef="indiv"/>
        </ct:VariabilityReference>
        <!-- SKIP -->
```

In this example we deal only with the IIV and a level below it, so this is all we have to encode variability-wise at this point.

The next part of the `<Population>` block was discussed previously. Beside the standard assignment of subjects to an *Arm* and providing information regarding *SEX*, we need to encode the information about study design related covariates, see Table 9.11. Treatment type, *TREAT* which varies in this cross-over design as the study progress from *Epoch1* to *Epoch3*, is considered here as covariate. And so are the treatment sequence, *TREATSEQ*, and occasions associated with EPOCHs. This table is populated with data as can be seen in the following listing

```
<ColumnMapping>
    <ds:ColumnRef columnIdRef="SEX"/>
    <ct:SymbRef blkIdRef="cm1" symbIdRef="Sex"/>
```

```
</ColumnMapping>
<ColumnMapping>
    <ds:ColumnRef columnIdRef="TREAT"/>
    <ct:SymbRef blkIdRef="cm1" symbIdRef="Treat"/>
</ColumnMapping>
<ColumnMapping>
    <ds:ColumnRef columnIdRef="TREATSEQ"/>
    <ct:SymbRef blkIdRef="cm1" symbIdRef="TreatSeq"/>
</ColumnMapping>
<ColumnMapping>
    <ds:ColumnRef columnIdRef="EPOCH"/>
    <ct:SymbRef symbIdRef="Occasion"/>
    <ds:CategoryMapping>
        <ds:Map dataSymbol="ep1" modelSymbol="occ1"/>
        <ds:Map dataSymbol="ep3" modelSymbol="occ2"/>
    </ds:CategoryMapping>
</ColumnMapping>
<ds:DataSet>
    <ds:Definition>
        <ds:Column columnId="ID" columnType="id" valueType="string" columnNum="1"/>
        <ds:Column columnId="ARM" columnType="arm" valueType="id" columnNum="2"/>
        <ds:Column columnId="SEX" columnType="covariate" valueType="string" columnNum="3"/>
        <ds:Column columnId="EPOCH" columnType="epoch" valueType="id" columnNum="4"/>
        <ds:Column columnId="TREAT" columnType="covariate" valueType="string" columnNum="5"/>
        <ds:Column columnId="TREATSEQ" columnType="covariate" valueType="string" columnNum="6"/>
    </ds:Definition>
    <ds:Table>
        <!-- arm1 -->
        <ds:Row><ct:String>1</ct:String><ct:Id>a1</ct:Id><ct:Id>M</ct:Id><ct:Id>ep1</ct:Id>
            <ct:Id>A</ct:Id><ct:Id>AB</ct:Id></ds:Row>
        <ds:Row><ct:String>1</ct:String><ct:Id>a1</ct:Id><ct:Id>M</ct:Id><ct:Id>ep3</ct:Id>
            <ct:Id>B</ct:Id><ct:Id>AB</ct:Id></ds:Row>
        <ds:Row><ct:String>2</ct:String><ct:Id>a1</ct:Id><ct:Id>M</ct:Id><ct:Id>ep1</ct:Id>
            <ct:Id>A</ct:Id><ct:Id>AB</ct:Id></ds:Row>
        <ds:Row><ct:String>2</ct:String><ct:Id>a1</ct:Id><ct:Id>M</ct:Id><ct:Id>ep3</ct:Id>
            <ct:Id>B</ct:Id><ct:Id>AB</ct:Id></ds:Row>
        <!-- arm2 -->
        <ds:Row><ct:String>9</ct:String><ct:Id>a2</ct:Id><ct:Id>M</ct:Id><ct:Id>ep1</ct:Id>
            <ct:Id>B</ct:Id><ct:Id>BA</ct:Id></ds:Row>
        <ds:Row><ct:String>9</ct:String><ct:Id>a2</ct:Id><ct:Id>M</ct:Id><ct:Id>ep3</ct:Id>
            <ct:Id>A</ct:Id><ct:Id>BA</ct:Id></ds:Row>
        <ds:Row><ct:String>10</ct:String><ct:Id>a2</ct:Id><ct:Id>M</ct:Id><ct:Id>ep1</ct:Id>
            <ct:Id>B</ct:Id><ct:Id>BA</ct:Id></ds:Row>
        <ds:Row><ct:String>10</ct:String><ct:Id>a2</ct:Id><ct:Id>M</ct:Id><ct:Id>ep3</ct:Id>
            <ct:Id>A</ct:Id><ct:Id>BA</ct:Id></ds:Row>
    </ds:Table>
</ds:DataSet>
</Population>
```

shows few subject data records for *Arm1* and *Arm2*. We have used in this example a new element, `<CategoryMapping>`, which use will be discussed in detail later in Section 9.5.6.

9.5.3 Variability Model

In this example the variability model is more complex than before, with IIV and IOV levels of variability, see Figure 9.12. At this point in the PharmML document we need to define the variability levels to be used in the rest of the document. You can see in the following listing

```
<VariabilityModel blkId="vm1" type="parameterVariability">
    <Level referenceLevel="true" symbId="indiv"/>
    <Level symbId="iov1">
        <ParentLevel>
            <ct:SymbRef symbIdRef="indiv"/>
        </ParentLevel>
    </Level>
</VariabilityModel>
```

that this is done by listing the variability levels using the `<VariabilityLevel>` element and their relationship. There are few important points to note here:

1. If more then one variability levels is used/defined in a model, it is useful to specify the reference level explicitly using the attribute `referenceLevel="true"` as can be seen for the level called `indiv`. As

explained previously it is not strictly required when the dataset and column mapping are exhaustively defined but it simplifies the understudying of the variability model structure.

2. There is parent-child relationship between the levels of variability. The reference *Subject* level is referenced with the attribute `symbId="indiv"` is above the *Occasion* level, referenced with the attribute `symbId="iov1"` which is what is done using the `<ParentLevel>` in the listing.

3. The name given to a level, using the `symbId` attribute, is **not** significant. We used the names *iov1* and *indiv* to provide clarity in other parts of the example document.

4. The type of each variability level (e.g., between-subject, inter-occasion, between-centre) is not defined here or in the Model Definition as a whole[8].

In this example the PharmML document tells us that there are two variability levels, the reference `indiv` level and the lower level of variability is called "`iov1`". Moreover, we need to know their order relative to each other.

9.5.4 Covariate Model

The covariate model describes categorical covariates, listed in Table 9.11, which we have not seen in the previous examples.

Because this is an estimation example no probabilities are provided and only the categories are defined, placed in the `<Categorical>` element. Then the implementation of each covariate follows the same schema, which will be explained for the gender covariate *Sex*. There are obviously two categories the covariate can be associate with *F* or *M*, which are encoded using the `<Category>` element followed by an optional `<Name>`.

See the following listing how this is done

```
<CovariateModel blkId="cm1">
    <Covariate symbId="Sex">
        <Categorical>
            <Category catId="F">
                <ct:Name>Female</ct:Name>
            </Category>
            <Category catId="M">
                <ct:Name>Male</ct:Name>
            </Category>
        </Categorical>
    </Covariate>
    <Covariate symbId="Treat">
        <Categorical>
            <Category catId="A"/>
            <Category catId="B"/>
        </Categorical>
    </Covariate>
    <Covariate symbId="TreatSeq">
        <Categorical>
            <Category catId="AB">
                <ct:Name>AB</ct:Name>
            </Category>
            <Category catId="BA">
                <ct:Name>BA</ct:Name>
            </Category>
        </Categorical>
    </Covariate>
    <Covariate symbId="Occasion">
        <Categorical>
            <Category catId="occ1">
                <ct:Name>1</ct:Name>
            </Category>
            <Category catId="occ2">
                <ct:Name>2</ct:Name>
            </Category>
        </Categorical>
    </Covariate>
</CovariateModel>
```

[8] N.B., The numerical levels described in the variability model (section 3.5) are not used.

9.5.5 Parameter Model

In example 1 (section 9.2) we showed you how to define an individual parameter in PharmML and relate that to a continuous covariate. Now in this example we will show how PharmML can be used to describe parameters that have multiple levels of variability and are related to categorical covariates.

In the following listing

```
<SimpleParameter symbId="omega_ka"/>
<SimpleParameter symbId="pop_ka"/>
<RandomVariable symbId="eta_ka">
    <ct:VariabilityReference>
        <ct:SymbRef blkIdRef="vm1" symbIdRef="indiv"/>
    </ct:VariabilityReference>
    <NormalDistribution xmlns="http://www.uncertml.org/3.0" definition="">
        <mean><rVal>0</rVal></mean>
        <stddev><var varId="omega_ka"/></stddev>
    </NormalDistribution>
</RandomVariable>
<IndividualParameter symbId="ka">
    <GaussianModel>
        <Transformation>log</Transformation>
        <LinearCovariate>
            <PopulationParameter>
                <ct:Assign><ct:SymbRef symbIdRef="pop_ka"/></ct:Assign>
            </PopulationParameter>
            <Covariate>
                <ct:SymbRef blkIdRef="cm1" symbIdRef="TreatSeq"/>
                <FixedEffect>
                    <ct:SymbRef symbIdRef="beta_ka_treatseq"/>
                    <Category catId="AB"/>
                </FixedEffect>
            </Covariate>
        </LinearCovariate>
        <RandomEffects>
            <ct:SymbRef symbIdRef="eta_ka"/>
        </RandomEffects>
    </GaussianModel>
</IndividualParameter>
```

we show the definition of parameter *ka*, which corresponds to (9.9). You should be familiar with this structure by now, but you should take note of the `<Category>` element within the `<FixedEffect>` element. We use this to tell PharmML that this fixed effect is related to the "`AB`" category of the *TreatSeq* covariate. This is equivalent to the expression $\beta_{ka,TreatSeq} 1_{TreatSeq_i=AB}$ in (9.9). Note that it is possible to do this more than once, for example if the covariate has more than two categories.

Parameter *ka* has only one level of variability, but this

```
<SimpleParameter symbId="pop_V"/>
<SimpleParameter symbId="omega_V"/>
<SimpleParameter symbId="gamma_V"/>
<SimpleParameter symbId="beta_V"/>
<SimpleParameter symbId="beta_V_occ1"/>
<SimpleParameter symbId="beta_V_Treat"/>
<SimpleParameter symbId="beta_V_TreatSeq"/>
<RandomVariable symbId="eta_V">
    <ct:VariabilityReference>
        <ct:SymbRef blkIdRef="vm1" symbIdRef="indiv"/>
    </ct:VariabilityReference>
    <NormalDistribution xmlns="http://www.uncertml.org/3.0" definition="">
        <mean><rVal>0</rVal></mean>
        <stddev><var varId="omega_V"/></stddev>
    </NormalDistribution>
</RandomVariable>
<RandomVariable symbId="kappa_V">
    <ct:VariabilityReference>
        <ct:SymbRef blkIdRef="vm1" symbIdRef="iov1"/>
    </ct:VariabilityReference>
    <NormalDistribution xmlns="http://www.uncertml.org/3.0" definition="">
        <mean><rVal>0</rVal></mean>
        <stddev><var varId="gamma_V"/></stddev>
    </NormalDistribution>
</RandomVariable>
```

and this listing

```
<IndividualParameter symbId="V">
    <GaussianModel>
        <Transformation>log</Transformation>
        <LinearCovariate>
            <PopulationParameter>
                <ct:Assign><ct:SymbRef symbIdRef="pop_V"/></ct:Assign>
            </PopulationParameter>
            <Covariate>
                <ct:SymbRef blkIdRef="cm1" symbIdRef="sex"/>
                <FixedEffect>
                    <ct:SymbRef symbIdRef="beta_V"/>
                    <Category catId="F"/>
                </FixedEffect>
            </Covariate>
            <Covariate>
                <ct:SymbRef blkIdRef="cm1" symbIdRef="Occasion"/>
                <FixedEffect>
                    <ct:SymbRef symbIdRef="beta_V_occ1"/>
                    <Category catId="occ1"/>
                </FixedEffect>
            </Covariate>
            <Covariate>
                <ct:SymbRef blkIdRef="cm1" symbIdRef="Treat"/>
                <FixedEffect>
                    <ct:SymbRef symbIdRef="beta_V_Treat"/>
                    <Category catId="A"/>
                </FixedEffect>
            </Covariate>
            <Covariate>
                <ct:SymbRef blkIdRef="cm1" symbIdRef="TreatSeq"/>
                <FixedEffect>
                    <ct:SymbRef symbIdRef="beta_V_TreatSeq"/>
                    <Category catId="AB"/>
                </FixedEffect>
            </Covariate>
        </LinearCovariate>
        <RandomEffects>
            <ct:SymbRef symbIdRef="eta_V"/>
        </RandomEffects>
        <RandomEffects>
            <ct:SymbRef symbIdRef="kappa_V"/>
        </RandomEffects>
    </GaussianModel>
</IndividualParameter>
```

show how we describe parameter V with both IIV and IOV levels of variability. Very simply we add a `<RandomVariable>` for each level of variability and use the `symbIdRef` attribute in the `<RandomEffects>` element to map the random effect to the appropriate variability model as defined at the beginning of the `<ModelDefinition>` element. Thus *eta_V* and *kappa_V* correspond to the random effects $\eta_{V,i}^{(0)}$ and $\eta_{V,ik}^{(+1)}$ in (9.10). This parameter is related to all four covariates, but we only show the *Sex* covariate. The others defined in a very similar manner as all the covariates in this model contain just 2 categories.

We will not show parameter Cl as it does not illustrate any new concepts, nor are any of the random effects in the model correlated. This does not mean there is no covariance matrix defined within the PharmML document. There is. The matrices in (9.11) and (9.12) are implicitly defined because the standard deviations of the random effects are explicitly defined and no correlation between them is defined.

9.5.6 NONMEM dataset

Now we will describe the case when the data and trial design are sourced from the NONMEM dataset. Table 9.14 show a typical dataset required for an estimation task. and the following code shows how the according dataset definition and column mappings

```
<ExternalDataSet toolName="NONMEM" oid="NMoid">
    <ColumnMapping>
        <ds:ColumnRef columnIdRef="ID"/>
        <ct:SymbRef blkIdRef="vm1" symbIdRef="indiv"/>
    </ColumnMapping>
    <ColumnMapping>
```

ID	TIME	DV	AMT	OCC	TREAT	TREATSEQ	SEX	MDV	EVID
1	0	.	100	1	1	12	0	1	1
1	0.25	2.1243964	.	1	1	12	0	0	0
1	0.5	4.308573	.	1	1	12	0	0	0
1	1	7.6059305	.	1	1	12	0	0	0
1	2	6.9678311	.	1	1	12	0	0	0
1	3.5	7.7741686	.	1	1	12	0	0	0
...	...	...	...	...	...	...	...	...	...
1	24	1.5194051	.	1	1	12	0	0	0
1	0	.	100	2	2	12	0	0	0
1	0.25	4.2049182	.	2	2	12	0	0	0
1	0.5	7.2508737	.	2	2	12	0	0	0
1	1	8.5792413	.	2	2	12	0	0	0
1	2	8.5689542	.	2	2	12	0	0	0
1	3.5	10.1764867	.	2	2	12	0	0	0
...	...	...	...	...	...	...	...	...	...
1	24	0.8831267	.	2	2	12	0	0	0
2	0	.	100	1	2	21	1	0	0
2	0.25	2.6643053	.	1	2	21	1	0	0
...	...	...	...	...	...	...	...	0	0

Table 9.14: A dataset used in example 4.

```
    <ds:ColumnRef columnIdRef="Time"/>
    <ct:SymbRef symbIdRef="t"/>
</ColumnMapping>
<ColumnMapping>
    <ds:ColumnRef columnIdRef="Y"/>
    <ct:SymbRef blkIdRef="om1" symbIdRef="Cc_obs"/>
</ColumnMapping>
<ColumnMapping>
    <ds:ColumnRef columnIdRef="AMT"/>
    <ct:SymbRef blkIdRef="sm1" symbIdRef="D"/>
</ColumnMapping>
<!-- mapping occasions column to the covariate model -->
<ColumnMapping>
    <ds:ColumnRef columnIdRef="OCC"/>
    <ct:SymbRef blkIdRef="cm1" symbIdRef="Occasion"/>
    <ds:CategoryMapping>
        <ds:Map modelSymbol="occ1" dataSymbol="1"/>
        <ds:Map modelSymbol="occ2" dataSymbol="2"/>
    </ds:CategoryMapping>
</ColumnMapping>
<!-- mapping occasions column to the variability model -->
<ColumnMapping>
    <ds:ColumnRef columnIdRef="OCC"/>
    <ct:SymbRef blkIdRef="vm1" symbIdRef="iov1"/>
</ColumnMapping>
<ColumnMapping>
    <ds:ColumnRef columnIdRef="TREAT"/>
    <ct:SymbRef blkIdRef="cm1" symbIdRef="Treat"/>
    <ds:CategoryMapping>
        <ds:Map dataSymbol="1" modelSymbol="A"/>
        <ds:Map dataSymbol="2" modelSymbol="B"/>
    </ds:CategoryMapping>
</ColumnMapping>
<ColumnMapping>
    <ds:ColumnRef columnIdRef="TREATSEQ"/>
    <ct:SymbRef blkIdRef="cm1" symbIdRef="TreatSeq"/>
    <ds:CategoryMapping>
        <ds:Map dataSymbol="12" modelSymbol="AB"/>
        <ds:Map dataSymbol="21" modelSymbol="BA"/>
    </ds:CategoryMapping>
</ColumnMapping>
```

```
        <ColumnMapping>
            <ds:ColumnRef columnIdRef="SEX"/>
            <ct:SymbRef blkIdRef="cm1" symbIdRef="Sex"/>
            <ds:CategoryMapping>
                <ds:Map dataSymbol="0" modelSymbol="M"/>
                <ds:Map dataSymbol="1" modelSymbol="F"/>
            </ds:CategoryMapping>
        </ColumnMapping>

        <!-- map 'tD' and 'TIME' if 'AMT' != 0 -->
        <ColumnMapping>
            <ds:ColumnRef columnIdRef="TIME"/>
            <ds:Piecewise>
                <math:Piece>
                    <ct:SymbRef blkIdRef="sm1" symbIdRef="tD"/>
                    <math:Condition>
                        <math:LogicBinop op="neq">
                            <ds:ColumnRef columnIdRef="AMT"/>
                            <ct:Real>0</ct:Real>
                        </math:LogicBinop>
                    </math:Condition>
                </math:Piece>
            </ds:Piecewise>
        </ColumnMapping>

        <ds:DataSet>
            <ds:Definition>
                <ds:Column columnId="ID" columnType="id" valueType="string" columnNum="1"/>
                <ds:Column columnId="TIME" columnType="time" valueType="real" columnNum="2"/>
                <ds:Column columnId="Y" columnType="dv" valueType="real" columnNum="3"/>
                <ds:Column columnId="AMT" columnType="dose" valueType="real" columnNum="4"/>
                <ds:Column columnId="OCC" columnType="covariate" valueType="int" columnNum="5"/>
                <ds:Column columnId="TREAT" columnType="covariate" valueType="int" columnNum="6"/>
                <ds:Column columnId="TREATSEQ" columnType="covariate" valueType="int" columnNum="7"/>
                <ds:Column columnId="SEX" columnType="covariate" valueType="int" columnNum="8"/>
            </ds:Definition>
            <ds:ImportData oid="dataOid">
                <ds:path>example4.csv</ds:path>
                <ds:format>CSV</ds:format>
                <ds:delimiter>COMMA</ds:delimiter>
            </ds:ImportData>
        </ds:DataSet>
    </ExternalDataSet>
```

The mapping of the column *OCC* deserves a short description. It is mapped twice

- first, to map the occasions to the according covariate in `<CovariateModel>` *cm1*
- second, to map them to the variability level, `iov1`, associated with the occasions defined in the `<VariabilityModel>` *vm1*

Compared to the previous examples the column mappings contain a new element, the `<CategoryMapping>` and within the `<Map>` elements use when mapping categorical covariates and discrete data model categories. Its attributes `dataSymbol` and `modelSymbol` identify the according symbols used in the model definition and the dataset, respectively.

Consider for example the covariate *TREATSEQ*, in the model it is much more intuitive to use the *AB* or *BA* symbol id's then the corresponding 12 or 21 identifiers required as the NONMEM datasets are not allowing strings. The latter is also the reason why in this case we much more often use the `<ds:CategoryMapping>` element as the dataset and model use different symbols for the identical categories.

9.5.7 Covered in previous examples

The remaining elements of this example to be encoded in PharmML are nearly identical to those described before, such as `<EstimationStep>` and `<StepDependencies>` within the `<ModellingSteps>` block, and will not be discussed here.

9.6 Example 5: Estimation with individual dosing

9.6.1 Description

This example is based on [Ribba et al., 2012] and deals with a mathematical model describing the inhibition of the tumour growth of low-grade glioma treated with chemotherapy[9]. Although previous estimation examples were complex enough to illustrate most important aspects of the current PharmML specification we would like briefly to discuss this example due to its role as a use case. It also illustrates a new feature of the language, the fact that we can encode explicitly using patient specific administration scenarios within the `<TrialDesign>` section.

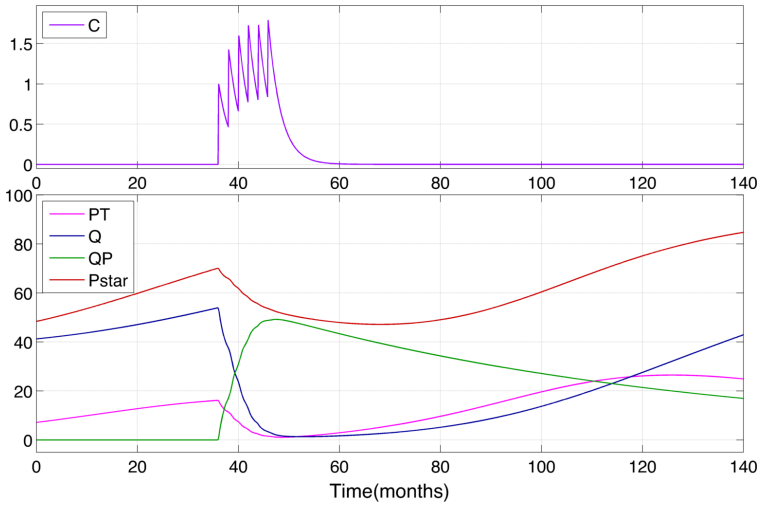


Figure 9.13: (top) PK time course of the drug concentration, C. (bottom) Tumour growth – time courses for the total tumour size, P^*, and its components for one subject with dosing times = {36.1, 38.05, 40.01, 41.96, 43.91, 45.87}.

9.6.2 Trial design

We will start with the definition of `<Structure>`, `<Population>`. The next language element, `<IndividualDosing>`, is, as mentioned above.

9.6.2.1 Structure

Figure 9.14 shows the design structure of this example consisting of one arm and one epoch, meaning there is one treatment type 'IV' for all patients. As explained in section 4 the design element `<Cell>` comprises the essential elements specifying the information about the arm, epoch and segment/activities. `<Segment>` contains treatment definition, here an IV bolus administration, defined in the `<Activity>` element. See the following listing

```
<Structure>
    <Epoch oid="epoch1">
        <Start><ct:Real>0</ct:Real></Start>
        <End><ct:Real>200</ct:Real></End>
        <Order>1</Order>
    </Epoch>
    <Arm oid="arm1"/>
    <Cell oid="cell1">
        <EpochRef oidRef="epoch1"/>
        <ArmRef oidRef="arm1"/>
        <SegmentRef oidRef="TA"/>
    </Cell>
    <Segment oid="TA">
        <ActivityRef oidRef="bolusIV"/>
```

[9]The example is encoded in two versions, `example5.xml` and `example5_NONMEM.xml`, with explicit encoded trial design and design sourced from a NONMEM datafile, respectively.

```
        </Segment>
        <Activity oid="bolusIV">
            <Bolus>
                <DoseAmount inputTarget="derivativeVariable">
                    <ct:SymbRef blkIdRef="sm1" symbIdRef="C"/>
                </DoseAmount>
            </Bolus>
        </Activity>
    </Structure>
```

for the PharmML implementation.

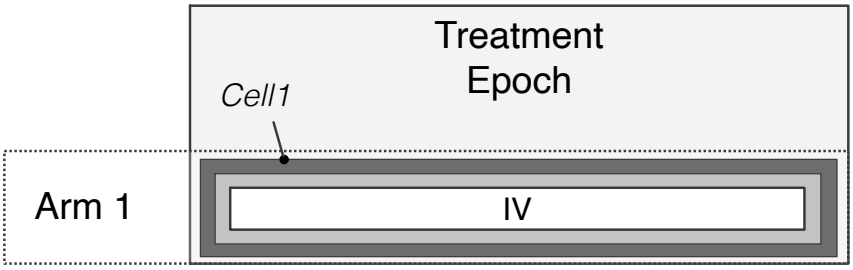


Figure 9.14: Design overview: single arm design.

Segment	Activity	Treatment	DoseTime	DoseSize	Target Variable
TA	bolusIV	IV bolus	individual	1	C

Table 9.15: Segment/activity overview.

9.6.2.2 Population

In the next step, the *Population* is defined, i.e. attributes of the individuals in the study. This means creating an individual template with columns for an identifier, arm and repetition and then populating the table with appropriate data. As no covariates are used here the *Population* description reduces to the assignment of the subjects to the single study arm, *Arm1*. As a shorthand we use the *repetition* method by defining the column 'rep', as can be seen in the following listing

```
    <Population>
        <ct:VariabilityReference>
            <ct:SymbRef blkIdRef="vm1" symbIdRef="indiv"/>
        </ct:VariabilityReference>

        <DataSet xmlns="http://www.pharmml.org/pharmml/0.6/Dataset">
            <Definition>
                <Column columnId="ID" columnType="id" valueType="id" columnNum="1"/>
                <Column columnId="ARM" columnType="arm" valueType="id" columnNum="2"/>
                <Column columnId="REP" columnType="replicate" valueType="int" columnNum="3"/>
            </Definition>
            <Table>
                <Row>
                    <ct:Id>i</ct:Id>
                    <ct:Id>arm1</ct:Id>
                    <ct:Int>21</ct:Int>
                </Row>
            </Table>
        </DataSet>
    </Population>
```

The identifiers, ID, created here are unique and will be used to the refer to specific subjects in the subsequent `<IndividualDosing>` structure element described in the following section.

9.6.2.3 Individual Dosing

This model utilises the idea of the so called K-PK model, meaning that the rate of the drug entry is relevant but not its absolute value. Such models often assume, as it is the case here, that the dose is equal 1 for all subject and dosing events, see Table 9.16.

The element *IndividualDosing* is used to implementing all such subject specific dosing events. First we have to associate the data which follow to an appropriate activity, this is done by referring to the 'bolusIV' which defined previously in `<Structure>`, as as shown in the following listing

```
<IndividualDosing>
    <ActivityRef oidRef="bolusIV"/>
    <ColumnMapping>
        <ColumnRef xmlns="http://www.pharmml.org/pharmml/0.6/Dataset" columnIdRef="TIME"/>
        <ct:SymbRef symbIdRef="time"/>
    </ColumnMapping>
    <DataSet xmlns="http://www.pharmml.org/pharmml/0.6/Dataset">
        <Definition>
            <Column columnId="ID" columnType="id" valueType="string" columnNum="1"/>
            <Column columnId="TIME" columnType="idv" valueType="real" columnNum="2"/>
            <Column columnId="DOSE" columnType="dose" valueType="real" columnNum="3"/>
        </Definition>
        <Table>
            <!-- subject 1 -->
            <Row><ct:String>1</ct:String><ct:Real>54.57</ct:Real><ct:Real>1</ct:Real></Row>
            <Row><ct:String>1</ct:String><ct:Real>59.77</ct:Real><ct:Real>1</ct:Real></Row>
            <!-- SNIP -->
            <!-- subject 21 -->
            <Row><ct:String>21</ct:String><ct:Real>1.5</ct:Real><ct:Real>1</ct:Real></Row>
            <Row><ct:String>21</ct:String><ct:Real>3.17</ct:Real><ct:Real>1</ct:Real></Row>
            <Row><ct:String>21</ct:String><ct:Real>4.85</ct:Real><ct:Real>1</ct:Real></Row>
            <Row><ct:String>21</ct:String><ct:Real>6.52</ct:Real><ct:Real>1</ct:Real></Row>
            <Row><ct:String>21</ct:String><ct:Real>8.19</ct:Real><ct:Real>1</ct:Real></Row>
            <Row><ct:String>21</ct:String><ct:Real>9.87</ct:Real><ct:Real>1</ct:Real></Row>
        </Table>
    </DataSet>
</IndividualDosing>
```

Next we map the subject's identifier *ID* to that created in the population definition. Finally a data set template using `<Definition>` element is defined, i.e. the columns *ID*, *TIME* and *DOSE*. Then the table is populated with subject specific values as shown here for subjects 1, 2 and 21.

9.6.3 Structural model definition

The following ODE system is defined:

$$\frac{dC}{dt} = -KDE \times C$$
$$\frac{dP}{dt} = \lambda_P \times P\left(1 - \frac{P^\star}{K}\right) + k_{QPP} \times Q_P - k_{PQ} \times P - \gamma \times C \times KDE \times P$$
$$\frac{dQ}{dt} = k_{PQ} \times P - \gamma \times C \times KDE \times Q$$
$$\frac{dQ_P}{dt} = \gamma \times C \times KDE \times Q - k_{QPP} \times Q_P - \delta_{QP} \times Q_P$$

$$P^\star = P + Q + Q_P$$

with initial conditions

$$C(t=0) = 1; \quad P(t=0) = P0; \quad Q(t=0) = Q0; \quad Q_P(t=0) = 0.$$

9.6.3.1 Estimating initial conditions

This example differs from the previous ones. It requires, in addition to model parameters, the estimation of the initial conditions of two tumour growth related variables. Moreover, the inter-individual variability is assumed for these variables. The value for $Q_P(t=0) = Q_{P_0}$ is fixed to 0 but the values for $P(t=0) = P_0$ and $Q(t=0) = Q_0$ are allowed to vary according to a log-normal distribution, see the following listing

```
<SimpleParameter symbId="pop_P0"/>
<SimpleParameter symbId="omega_P0"/>
<RandomVariable symbId="eta_P0">
    <ct:VariabilityReference>
        <ct:SymbRef blkIdRef="vm1" symbIdRef="indiv"/>
    </ct:VariabilityReference>
    <NormalDistribution xmlns="http://www.uncertml.org/3.0" definition="">
        <mean><rVal>0</rVal></mean>
        <stddev><var varId="omega_P0"/></stddev>
    </NormalDistribution>
</RandomVariable>
<IndividualParameter symbId="P0">
    <GaussianModel>
        <Transformation>log</Transformation>
        <LinearCovariate>
            <PopulationParameter>
                <ct:Assign>
                    <ct:SymbRef symbIdRef="pop_P0"/>
                </ct:Assign>
            </PopulationParameter>
        </LinearCovariate>
        <RandomEffects>
            <ct:SymbRef symbIdRef="eta_P0"/>
        </RandomEffects>
    </GaussianModel>
</IndividualParameter>
```

where the definition of the distribution for the initial condition P_0 is shown.

9.6.4 NONMEM dataset

Now we will describe the case when the data and trial design are sourced from the NONMEM dataset. Table 9.14 show a typical dataset required for an estimation task.

ID	TIME	DV	MDV	AMT	EVID
1	0	.	1	.	0
1	3.43	45.7	0	.	0
1	52.63	79.3	0	.	0
1	54.57	.	1	1	1
1	57.53	72.3	0	.	0
1	59.77	.	1	1	1
1	63.3	72.07	0	.	0
...	...	...	...	...	...
1	121.87	90.16	0	.	0
2	0	50.17	0	.	0
2	12	.	1	1	1
2	14.09	.	1	1	1
2	14.17	52.82	0	.	0
...	...	...	...	...	...

Table 9.16: A dataset used in example 5. The columns are: the identifier, ID, time for measurements and dosing events, dependent variable, DV, which stands for *PSTAR* – the total tumour size and the dose, DOSE. As common for K-PD models, the dose is equal 1 for all subjects and dosing events.

The following code shows how the dataset definition and column mappings

```
<ExternalDataSet toolName="NONMEM" oid="NMoid">

    <ColumnMapping>
        <ColumnRef xmlns="http://www.pharmml.org/pharmml/0.6/Dataset" columnIdRef="ID"/>
        <ct:SymbRef blkIdRef="vm1" symbIdRef="indiv"/>
    </ColumnMapping>
    <ColumnMapping>
        <ColumnRef xmlns="http://www.pharmml.org/pharmml/0.6/Dataset" columnIdRef="TIME"/>
```

```
            <ct:SymbRef symbIdRef="time"/>
        </ColumnMapping>
        <ColumnMapping>
            <ColumnRef xmlns="http://www.pharmml.org/pharmml/0.6/Dataset" columnIdRef="DV"/>
            <ct:SymbRef blkIdRef="om1" symbIdRef="PSTAR_obs"/>
        </ColumnMapping>
        <ColumnMapping>
            <ColumnRef xmlns="http://www.pharmml.org/pharmml/0.6/Dataset" columnIdRef="AMT"/>
            <ct:SymbRef blkIdRef="om1" symbIdRef="C"/>
        </ColumnMapping>

        <!--columns: ID TIME DV MDV DOSE EVID-->
        <DataSet xmlns="http://www.pharmml.org/pharmml/0.6/Dataset">
            <Definition>
                <Column columnId="ID" columnType="id" valueType="string" columnNum="1"/>
                <Column columnId="TIME" columnType="time" valueType="real" columnNum="2"/>
                <Column columnId="DV" columnType="dv" valueType="real" columnNum="3"/>
                <Column columnId="MDV" columnType="mdv" valueType="int" columnNum="4"/>
                <Column columnId="AMT" columnType="dose" valueType="real" columnNum="5"/>
                <Column columnId="EVID" columnType="evid" valueType="int" columnNum="6"/>
            </Definition>
            <ImportData oid="dataOid">
                <path>datasets/example5.csv</path>
                <format>CSV</format>
                <delimiter>COMMA</delimiter>
            </ImportData>
        </DataSet>
    </ExternalDataSet>
```

9.6.5 Modelling steps

This requires the specification of the following items: *EstimationStep* and *StepDependencies*. It has been described in previous examples in detail and will be skipped here.

9.7 Example 6: Joint PKPD model with count data

The following examples feature a new aspect of the PharmML, the support of discrete data models[10]. The first one is using the Poisson distribution to describe count data[11], from the MLXTRAN tutorial, [Lixoft, 2014a].

9.7.1 Description

The essential bit of information for this task is the probability distribution, the probability mass function (PMF), of count data Y, which can be defined in either un-transformed, P(Y=k), or transformed, log(P(Y=k)), form. As in example 9.2, the underlying PK model is 1-compartmental oral model and will be omitted here. We assume the basic Poisson model which reads

$$P\big(Y_{ij} = k | Cc_{ij}, \psi_i\big) = \frac{e^{-\lambda_{ij}} \lambda_{ij}^k}{k!} \tag{9.13}$$

with concentration dependent mean λ, defined as

$$\lambda_{ij} = \lambda_0 \Big(1 - \frac{Cc_{ij}}{IC_{50} + Cc_{ij}}\Big) \tag{9.14}$$

also called *Poisson intensity*. Here, λ, depends on the parameters λ_0 and IC_{50} which are sampled from log-normal distribution. λ_0, stands here for the baseline seizure count prior to any drug. The value of λ, parameterised with IC_{50} and λ_0, is reduced by the concentration in the central compartment, Cc, which is visualised for three different values of $Cc = \{1, 5, 15\}$ in 9.15 ($1 \equiv$ green, $5 \equiv$ red, $15 \equiv$ blue).

[10] The discrete data models are coming in a large variety, which would be worth a detailed discussion. This and the next example cover only the very basic cases. More than 20 additional discrete date examples, encoded completely in PharmML, can be found on our webpage `http://pharmml.org`. See also discrete data model templates in the appendix, A.

[11] The example is encoded in `example6_NONMEM.xml` with design sourced from a NONMEM datafile.

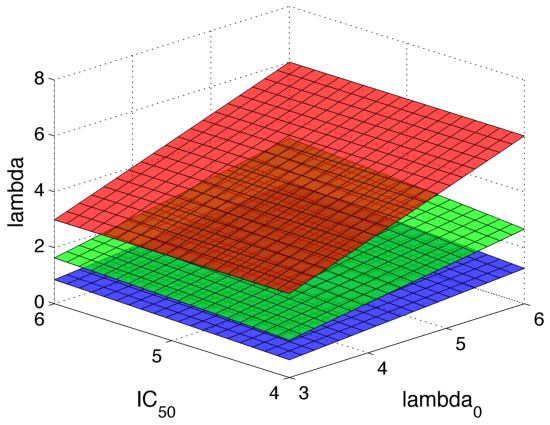


Figure 9.15: λ–surface as function of Cc, λ_0 and IC_{50} plotted for $Cc = \{1, 5, 15\}$.

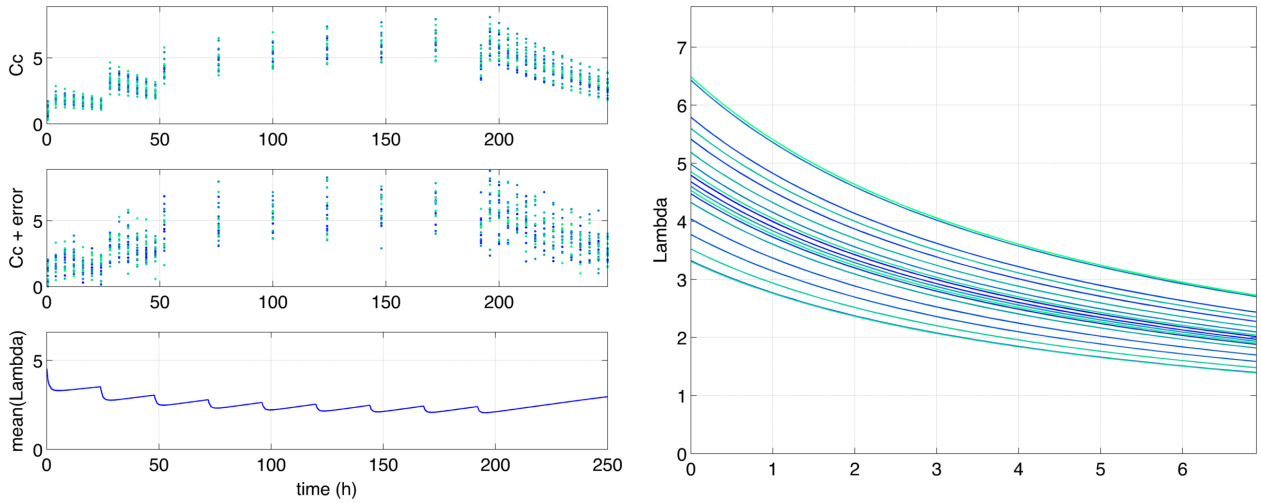


Figure 9.16: (left) The time courses for the concentration and mean Poisson intensity, λ. (right) Poisson intensity as given by eq.(9.14) for all subject as function of the concentration between 0 and the maximum achievable concentration across 20 subjects.

9.7.1.1 Individual parameters model

$$\begin{aligned} \lambda_0 &\sim \text{logNormal}(pop_{\lambda_0}, \omega_{\lambda_0}); \quad pop_{\lambda_0} = 5, \quad \omega_{\lambda_0} = 0.2 \\ IC_{50} &\sim \text{logNormal}(pop_{IC_{50}}, \omega_{IC_{50}}); \quad pop_{IC_{50}} = 5, \quad \omega_{IC_{50}} = 0 \end{aligned}$$

9.7.1.2 Observation model

We apply a combined residual error model to the continuous PK output variable Cc and Poisson error distribution for the discrete PD component, defined by Y, as the following table shows

Output Variable	***Cc***	***Y***
Observation Name	Concentration	State
Units	mg/l	–
Type	Continuous	Discrete/Count
Model	Combined	Poisson
Parameters	a, b	λ_0, IC_{50}

Two additional important bits of information which are part of the discrete observation model to be provided are

- Intensity Parameter, λ, given by eq. (9.14)
- Link function – log.

9.7.1.3 Modelling Steps

Additionally to an estimation task (not described further), the PK and PD output variables are to be generated by the simulation with their associated time points are shown below:

Output Variable	***Cc***	λ
Observation times	$[0.5, 4 : 4 : 48, 52 : 24 : 192, 196 : 4 : 250]$	$0 : 24 : 192$

See Figure 9.16 for a typical result plots for a group of subjects.

9.7.2 Observation model

While the continuous observation model and its features have been described in very detail in the previous examples, the error distribution of the discrete effect given by eqs.(9.13) and (9.14) will be described in the following for the first time.

First we have to indicate the type of the observation model using the elements `<Discrete>` and specifically for this example the `<CountData>`. The next mandatory elements are the count index, k, (required only when the PMF is explicitly specified, see below) and the `<CountVariable>` which will be used later to establish the link between the model and the related date set, see Table 9.17. Then the characteristic parameter for the Poisson distribution as function of the drug concentration, Cc, encoded as *Lambda*, is implemented using `<IntensityParameter>` as the following snippet shows

```
<ObservationModel blkId="om1">
    <Discrete>
        <CountData>
            <ct:Variable symbolType="int" symbId="k"/>
            <CountVariable symbId="Y"/>

            <!-- Poisson intensity - function of drug concentration, Cc -->
            <IntensityParameter symbId="Lambda">
                <ct:Assign>
                    <math:Equation>
                        <math:Binop op="times">
                            <ct:SymbRef blkIdRef="pm1" symbIdRef="lambda0"/>
                            <math:Binop op="minus">
                                <ct:Real>1</ct:Real>
                                <math:Binop op="divide">
                                    <ct:SymbRef blkIdRef="sm1" symbIdRef="Cc"/>
                                    <math:Binop op="plus">
                                        <ct:SymbRef blkIdRef="pm1" symbIdRef="IC50"/>
                                        <ct:SymbRef blkIdRef="sm1" symbIdRef="Cc"/>
                                    </math:Binop>
                                </math:Binop>
                            </math:Binop>
                        </math:Binop>
                    </math:Equation>
                </ct:Assign>
            </IntensityParameter>
            <!-- see next listing for the continuation of the observation model -->
```

Now the Poisson probability mass function (PMF) can be defined, but here in the transformed format

$$\log(P(Y_{ij} = k|Cc_{ij}, \psi_i)) = -\lambda_{ij} + k \times \lambda_{ij} - \log(k!)$$

which can be done in various ways, either

- using the UncertML standard and referring to *Lambda* as in the following snippet

```
<PMF linkFunction="log">
    <PoissonDistribution xmlns="http://www.uncertml.org/3.0"
        definition="http://www.uncertml.org/3.0">
        <rate>
            <var varId="Lambda"/>
```

```
                        </rate>
                    </PoissonDistribution>
                </PMF>
            </CountData>
        </Discrete>
    </ObservationModel>
```

with `linkFunction` attribute indicating the logarithmic transformation, or

- by encoding explicitly the PMF in the transformed format and specifying as before the applied link function, as the following snippet shows

```
                <PMF linkFunction="log">
                    <math:LogicBinop op="eq">
                        <ct:SymbRef symbIdRef="Y"/>
                        <ct:SymbRef symbIdRef="k"/>
                    </math:LogicBinop>
                    <ct:Assign>
                        <Equation xmlns="http://www.pharmml.org/pharmml/0.6/Maths">
                            <Binop op="minus">
                                <Binop op="plus">
                                    <Uniop op="minus">
                                        <ct:SymbRef symbIdRef="Lambda"/>
                                    </Uniop>
                                    <Binop op="times">
                                        <ct:SymbRef symbIdRef="k"/>
                                        <Uniop op="log">
                                            <ct:SymbRef symbIdRef="Lambda"/>
                                        </Uniop>
                                    </Binop>
                                </Binop>
                                <Uniop op="factln">
                                    <ct:SymbRef symbIdRef="k"/>
                                </Uniop>
                            </Binop>
                        </Equation>
                    </ct:Assign>
                </PMF>
            </CountData>
        </Discrete>
    </ObservationModel>
```

Note, that although the UncertML driven solution is very simple and straightforward to implement, it is also limited to only this case, see also Section 3.7.2. In short, for count data models, only the basic Poisson model is available in the UncertML standard.

9.7.3 NONMEM dataset

The remaining part is the the data and trial design as sourced from the NONMEM dataset (using dummy values). Table 9.17 show a typical dataset required for an estimation task.

```
<mstep:ExternalDataSet toolName="NONMEM" oid="NMoid">
    <mstep:ColumnMapping>
        <ds:ColumnRef columnIdRef="TIME"/>
        <ct:SymbRef symbIdRef="t"/>
    </mstep:ColumnMapping>
    <mstep:ColumnMapping>
        <ds:ColumnRef columnIdRef="AMT"/>
        <ct:SymbRef blkIdRef="sm1" symbIdRef="Ad"/>
    </mstep:ColumnMapping>
    <mstep:MultipleDVMapping>
        <ds:ColumnRef columnIdRef="DV"/>
        <mstep:Piecewise>
            <math:Piece>
                <ct:SymbRef blkIdRef="om1" symbIdRef="Y"/>
                <math:Condition>
                    <math:LogicBinop op="eq">
                        <ds:ColumnRef columnIdRef="DVID"/>
                        <ct:Int>2</ct:Int>
                    </math:LogicBinop>
```

ID	TIME	AMT	Y	DVID
1	0	0.25	.	.
1	4	.	19.6	1
1	8	.	16	2
1	12	.	9.5	1
1	18	.	3.4	1
1	24	.	2	2
2	0	0.25	16	.
2	4	.	14.8	1
2	8	.	6	2
2	12	.	8.1	1
2	18	.	2.2	1
2	24	.	0	2
...	...	...	...	...

Table 9.17: A dataset used in example for first two subjects. The additional column DVID is used to specify the type of data. Here, DVID = 1 is used for a continuous response and DVID = 2 for count data.

```
                    </math:Condition>
                </math:Piece>
                <math:Piece>
                    <ct:SymbRef blkIdRef="om2" symbIdRef="Cc_obs"/>
                    <math:Condition>
                        <math:LogicBinop op="eq">
                            <ds:ColumnRef columnIdRef="DVID"/>
                            <ct:Int>1</ct:Int>
                        </math:LogicBinop>
                    </math:Condition>
                </math:Piece>
            </mstep:Piecewise>
        </mstep:MultipleDVMapping>
        <ds:DataSet>
            <ds:Definition>
                <ds:Column columnId="ID" columnType="id" valueType="string" columnNum="1"/>
                <ds:Column columnId="TIME" columnType="idv" valueType="real" columnNum="2"/>
                <ds:Column columnId="AMT" columnType="dose" valueType="real" columnNum="3"/>
                <ds:Column columnId="DV" columnType="dv" valueType="real" columnNum="4"/>
                <ds:Column columnId="DVID" columnType="dvid" valueType="real" columnNum="5"/>
            </ds:Definition>
            <ds:ExternalFile oid="dataOid">
                <ds:path>datasets/example_poisson.csv</ds:path>
            </ds:ExternalFile>
        </ds:DataSet>
    </mstep:ExternalDataSet>
```

Mapping of dosing related data has been described previously on multiple occasions and will not be discussed here. The conditional mapping required in the case when multiple observations have to be mapped using the `<mstep:MultipleDVMapping>` has been described in example 1, see section 9.2.7. What is new however is the mapping target when dealing with count data. The count variable, Y, as defined in the very beginning of the `<CountData>` element in the observation model `om1` with

```
<CountVariable symbId="Y"/>
```

is the target for this conditional mapping. The observation column, DV, is mapped to y if DVID=2 and to *Cc_obs* if DVID=1.

9.8 Example 7: Joint PKPD model with categorical data

This last example features another type of discrete models supported by PharmML– the categorical data models.

9.8.1 Description

In this example, there are two categories, $k \in 0, 1$, i.e. the effect outcome is either 0 or 1. The underlying PK model is again an 1-compartmental oral model and will not be described again. The probability for the category 1 is given by the following formula

$$p1 = \frac{1}{1 + \exp(-\theta_1 - \theta_2 \log(Cc))} \tag{9.15}$$

which is plotted in 9.17. This $p1$–surface is also function of θ_1, θ_2 and $log(Cc)$ which is visualised for three different values of $Cc = \{1, 5, 15\}$.

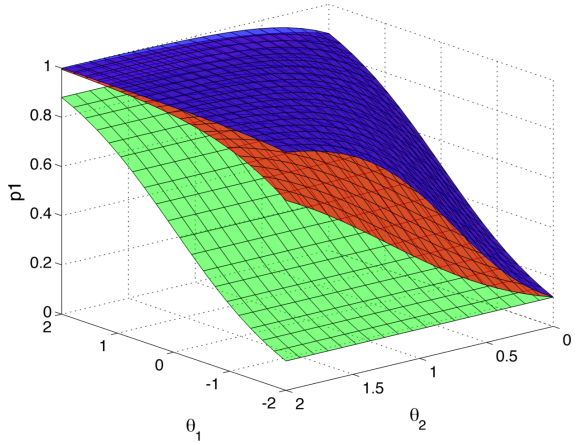


Figure 9.17: p1 probability surface as function of θ_1 and θ_2 plotted for Cc = {1,5,15} (1 ≡ green, 5 ≡ red, 15 ≡ blue).

9.8.1.1 Individual parameters model

$$\begin{aligned} theta1 &\sim \text{Normal}(pop_{theta1}, \omega_{theta1}); \quad pop_{theta1} = -1, \quad \omega_{theta1} = 0.3 \\ theta2 &\sim \text{logNormal}(pop_{theta2}, \omega_{theta2}); \quad pop_{theta2} = 1, \quad \omega_{theta2} = 0.2 \end{aligned}$$

9.8.1.2 Observation model

We apply a combined residual error model to the continuous PK output variable Cc and Poisson error distribution for the discrete PD component, defined by Y, as the following table shows

Output Variable	***Cc***	***Y***
Observation Name	Concentration	State
Units	mg/l	–
Type	Continuous	Discrete/Categorical
Model	Combined	Binomial
Parameters	a, b	θ_1, θ_2

9.8.1.3 Modelling Steps

Apart from an estimation task (not described further), the PK and PD output variables are to be generated by the simulation with their associated time points are shown below:

Output Variable	***Cc***	$p1$
Observation times	[0.5, 4 : 4 : 48, 52 : 24 : 192, 192 : 4 : 250]	0 : 24 : 288

See Figure 9.18 for a typical result plots for a group of subjects.

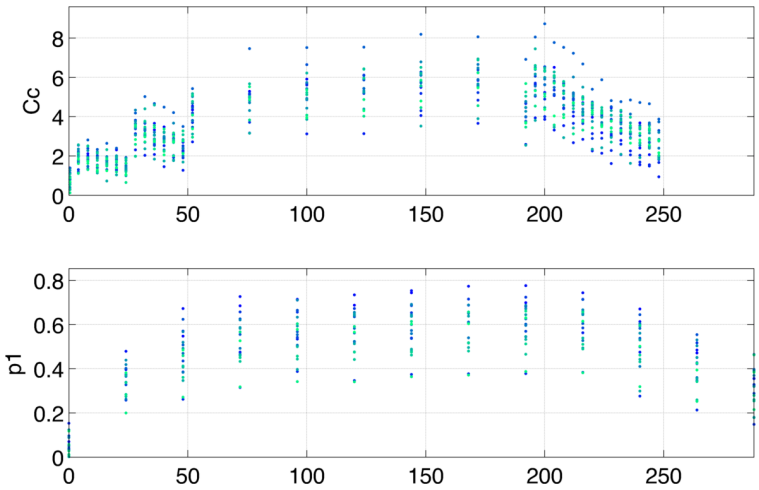


Figure 9.18: Plots for first arm in the study. (top) Concentration, Cc, time course of the PK model. (bottom) Probability, $p1$, time course as defined in eq.9.15.

9.8.2 Observation model

The `<ObservationModel>` for count data is the new element in this example. There are different ways to define such model and we present one of those.[12] Because we are dealing with nominal categorical data, we store this information using the attribute `ordered`. Next we encode the probability for category 1 as defined by equation 9.15, as the following example shows, which will be used later in the definition of the PMF.

```
<ObservationModel blkId="om1">
    <Discrete>
        <CategoricalData ordered="no">

            <ct:Variable symbolType="real" symbId="p1">
                <ct:Assign>
                    <math:Equation>
                        <math:Binop op="divide">
                            <ct:Real>1</ct:Real>
                            <math:Binop op="plus">
                                <ct:Real>1</ct:Real>
                                <math:Uniop op="exp">
                                    <math:Binop op="minus">
                                        <math:Uniop op="minus">
                                            <ct:SymbRef blkIdRef="pm1" symbIdRef="theta1"/>
                                        </math:Uniop>
                                        <math:Binop op="times">
                                            <ct:SymbRef blkIdRef="pm1" symbIdRef="theta2"/>
                                            <math:Uniop op="log">
                                                <ct:SymbRef blkIdRef="sm1" symbIdRef="Cc"/>
                                            </math:Uniop>
                                        </math:Binop>
                                    </math:Binop>
                                </math:Uniop>
                            </math:Binop>
                        </math:Binop>
                    </math:Equation>
                </ct:Assign>
            </ct:Variable>
```

Next the categories are encoded withn `<ListOfCategories>`, the `<CategoryVariable>`, y. At last, the aforementioned probability mass function using UncertML is specified, i.e. the binomial distribution with the probability of success equal $p1$ specified above, as the following code snippet shows

```
            <ListOfCategories>
                <Category symbId="cat0"/>
                <Category symbId="cat1"/>
            </ListOfCategories>
```

[12] For an alternative encoding see examples on our website `http://pharmml.org`.

```
                <CategoryVariable symbId="y"/>

                <PMF linkFunction="identity">
                    <BinomialDistribution xmlns="http://www.uncertml.org/3.0" definition="">
                        <numberOfTrials>
                            <nVal>1</nVal>
                        </numberOfTrials>
                        <probabilityOfSuccess>
                            <var varId="p1"/>
                        </probabilityOfSuccess>
                    </BinomialDistribution>
                </PMF>
            </CategoricalData>
        </Discrete>
    </ObservationModel>
```

This is because UncertML does not allow the encoding of assignments within their elements, such as `<probabilityOfSuccess>`, so that we first have to define $p1$ and used its reference in the PMF definition.

9.8.3 NONMEM dataset

The remaining part is the data and trial design as sourced from the NONMEM dataset (with dummy values). Table 9.18 show a typical dataset required for an estimation task.

ID	TIME	AMT	Y	DVID
1	0	0.25	.	.
1	1	.	1	2
1	4	.	2.9	1
1	8	.	1	2
1	10	.	0	2
1	12	.	8.5	1
1	18	.	6.4	1
1	24	.	0	2
2	0	0.25	6.1	.
2	4	.	8.5	1
2	8	.	0	2
2	12	.	1.1	1
2	18	.	5.3	1
2	24	.	1	2
...	...	...	...	...

Table 9.18: A dataset used in example for first two subjects. The column DVID is used to specify the type of data. Here, DVID = 1 is used for a continuous response and DVID = 2 for categorical data, with only two values allowed, either 0 for `cat0` or 1 for `cat1`.

***Nested* mapping** Because of the fact that we are dealing with a joint PK/PK model, we have to make use of the `<MultipleDVMapping>` element described in examples before. However, due to the fact that we are dealing with a categorical observed variable, whose categories have to matched to their symbol identifiers as defined in the observation model, we need additional `<CategoryMapping>` as shown in the following code snippet.

```
    <mstep:ExternalDataSet toolName="NONMEM" oid="NMoid">

        <mstep:ColumnMapping>
            <ds:ColumnRef columnIdRef="TIME"/>
            <ct:SymbRef symbIdRef="t"/>
        </mstep:ColumnMapping>
        <mstep:ColumnMapping>
            <ds:ColumnRef columnIdRef="AMT"/>
            <ct:SymbRef blkIdRef="sm1" symbIdRef="Ad"/>
```

```
            </mstep:ColumnMapping>
            <mstep:MultipleDVMapping>
                <ds:ColumnRef columnIdRef="DV"/>
                <mstep:Piecewise>
                    <math:Piece>
                        <ct:SymbRef blkIdRef="om1" symbIdRef="y"/>
                        <math:CategoryMapping>
                            <ds:Map dataSymbol="0" modelSymbol="cat0"/>
                            <ds:Map dataSymbol="1" modelSymbol="cat1"/>
                        </math:CategoryMapping>
                        <math:Condition>
                            <math:LogicBinop op="eq">
                                <ds:ColumnRef columnIdRef="DVID"/>
                                <ct:Int>2</ct:Int>
                            </math:LogicBinop>
                        </math:Condition>
                    </math:Piece>
                    <math:Piece>
                        <ct:SymbRef blkIdRef="om2" symbIdRef="C_obs"/>
                        <math:Condition>
                            <math:LogicBinop op="eq">
                                <ds:ColumnRef columnIdRef="DVID"/>
                                <ct:Int>1</ct:Int>
                            </math:LogicBinop>
                        </math:Condition>
                    </math:Piece>
                </mstep:Piecewise>
            </mstep:MultipleDVMapping>

            <!-- Dataset omitted, identical as in previous example -->
        </mstep:ExternalDataSet>
```

In this *nested* mapping we map the symbols used in the data set, e.g. `dataSymbol="0"` with category symbols, `modelSymbol="cat0"`, of the variable y.

9.9 More examples

A number of additional continuous and discrete data, PharmML coded, model examples is available on our website, such as

- Basic and complex PK macros – 12 examples, among others models corresponding to seven predefined PREDPP models, i.e. ADVAN1-4, 10-12 [Beal et al., 2009].
- Discrete data models – 22 examples for
  - count data models, e.g. zero-inflated, negative binomial, generalized Poisson, Poisson Mixture model.
  - categorical data models, e.g. nominal and ordinal regression models such as proportional odds or models with discrete- and continuous-time Markov dependency and
  - time-to-event data models, e.g. constant or concentration dependent hazard, Weibull model etc.
- DDEs – 3 examples (e.g. arthritis and SEIRS models [Lavielle and Lixoft Team, 2014]).

Chapter 10

Changelog

The first public release, version 0.2.1, was on 21^{st} November 2013. The developers involved were Stuart Moodie, Maciej Swat, Niels Rode Kristensen and Nicolas Le Novère. At the time of the first public release, PharmML structure allowed to implement continuous data models using not only a flexible parameter model but also variability, covariate and observation models. Already then the user was able to specify estimation and simulation tasks. The explicit encoded trial design was the only option to inform the model about the underlying study structure, i.e. the available covariates, dosing and observations with inline dataset support for every required data component. Mid October 2013 the team underwent a reconstruction and includes until today: Maciej Swat, Niels Rode Kristensen, Sarala Wimalaratne and Florent Yvon.

Apart from the public releases, PharmML went through a number of internal changes. The following tables give a detailed overview of this evolution (see also the timeline, Figure 1.4, for an brief overview).

10.1 Changes in version 0.3 & 0.3.1

PharmML element or modelling aspect	version 0.2.1	**version 0.3**
Experimental data	Inline data support only with up to 3 tables for – Observations – Dosing – Covariates	Following options are available 1. Inline data support (see left column for details) 2. External data files support 3. NONMEM-format dataset support with `<NONMEMdataSet>` 4. Lookup tables
Column Mapping	`<Population>` table	
	Multiple mapping elements: – `<Demographic>` – `<IndividualTemplate>` — `<IndividualMapping>` — `<ArmMapping>` — `<CovariateMapping>` — `<DemographicMapping>` — `<ReplicateMapping>` — `<IVDependentMapping>` with `<IndependentVariableMapping>`, `<EpochMapping>` and `<CovariateMapping>`	Only one type is available: – `<ColumnMapping>` everything else defined using new attribute `columnType`
	`<IndividualDosing>` table	
	Multiple mapping elements:	Activity reference is required

	– <ActivityRef> – <IndividualRef> ...	<ActivityRef> and <ColumnMapping>
	<ObjectiveDataSet> table	
	– <IndividualMapping> – <VariableMapping>	<ColumnMapping>
ODE's	<InitialCondition> with no child elements	<InitialCondition> with two new elements: <InitialTime> & <InitialValue>
Matrix	*not supported*	<Matrix> element with attributes: – VariabilityReference – matrixType with values CovMatrix, CorrMatrix, StDevCorrMatrix, Cholesky – RowNames – ColumnNames (optional)
External tools	*not supported*	<TargetTool> tag with *mandatory* element <TargetToolName> and *optional* <CodeInjection>
Trial Design	Explicitly coded in <TrialDesign>	Following options are available 1. Explicitly coded in <TrialDesign> 2. Design & data from NONMEM-format dataset
Interpolation of covariates and variables	*not supported*	<Interpolation> element with – <Algorithm> taking a value from list: constant, nearest, linear, ..., cubic or – user-defined in element <FunctionDefinition>
Changes in attribute names	in <DoseAmount> and <LookupTable>/<Target> elements	
	inputType with values – dose – target	inputTarget with values – parameter – derivativeVariable – variable
	in <VariabilityReference> element values of the type attribute	
	model error	parameterVariability residualError
Nested tables	supported in all datasets	*not supported*
Gaussian parameter model		identity transformation in <Transformation> added
Missing mathematical functions		new BinOp: min, max UniOp with normcdf, sqrt, ln, factln, gammaln
Metadata attribute	not defined for number of elements	added in the root <PharmML> and child elements of <TrialDesign>
Random errors	multiple <SymbRef> allowed in <RandomEffects>	only one <SymbRef> allowed per <RandomEffects>

		version 0.3.1
Mapping multiple dependent variables	*not supported*	`<MultipleDVMapping>` element
Dosing mapping from NONMEM datasets	*not supported*	– `cmt` attribute – `compartmentNo` attribute
File referencing	using `<filename>` and `<url>`	– `<path>` element instead – `SEMICOLON` delimiter
Referencing NONMEM datasets	using `<TargetToolReference>`	`<NONMEMdataSetReference>`
Simulation support	– only explicitly via `<TrialDesign>` – `<Observations>` element mandatory	– `<Observations>` optional – NONMEM datasets can be used
Referencing categorical covariates	possible using strings only	extended to numerical identifiers

10.2 Changes in version 0.4 & 0.4.1

PharmML element or modelling aspect	version $\leq$ 0.3.1	**version 0.4**
Discrete data models	*not supported*	`<Discrete>` with child elements such as – `<CountData>` with `<CountVariable>`, `<PreviousCountVariable>`, `<Dependance>`, `<IntensityParameter>`, etc. and `<PMF>` with `linkFunction` attribute – `<CategoricalData>` with child elements such as `<ListOfCategories>`, `<CategoryVariable>`, `<InitialStateVariable>` (plus few other variable elements), `<ProbabilityAssignment>` and its child elements: `<Probability>`, `<CurrentState>`, `<PreviousState>` and `<Condition>` – `<TimeToEventData>` with child elements `<EventVariable>`, `<HazardFunction>`, `<SurvivalFunction>`, `<Censoring>` and `<MaximumNumberEvents>`
Observation model	–	new `<DiscreteData>` and `<ContinuousData>` tags
Delay differential equations (DDE)	*not supported*	`<Delay>` and `<History>` with child elements – `<HistoryValue>` and – `<HistoryTime>`
Extended vector definition	basic support	`<Vector>` with a number of child elements such as `<VectorElements>`, `<VectorCell>`, `<VectorSegment>` and attributes `default`, `length`
Extended matrix definition	basic support	`<Matrix>` with a number of child elements such as `<MatrixRow>`, `<MatrixCell>`, `<MatrixBlock>` and attributes `matrixType`, `diagDefault`, `offDiagDefault` and others
	version $\leq$ 0.4	**version 0.4.1**
Mapping between model elements and dataset or `<TrialDesign>`	*limited support*	– `compartmentNo` attribute has been removed – `<TargetMapping>` and `<Map>` element with attributes `dataSymbol` and `modelSymbol`

Covariate model	*not supported*	`<TransformedCovariate>` element
Parameter model	element was mandatory *feature not supported*	– `<FixedEffect>` element in optional – `<FixedEffect>` takes numerical children
Vector/Matrix	*not supported*	lower/upper indexes can be defined using expressions
Vector elements renamed	`<VectorIndex>` `<SegmentIndex>`	`<CellIndex>` `<SegmentStartIndex>`
`<Sum>` and `<Product>` elements	*features missing*	– lower/upper indexes can be defined using arbitrary expressions
Dose scaling when working with datasets	*not supported*	`<ColumnTransformation>` element with `tranformId` attribute and new `tranformIdRef` attribute within `<ColumnRef>`
Discrete data models	*link functions missing*	link functions `loglog`, `comploglog`
Simulations	*missing element*	`<Discrete>` element within `<Observations>` tag
`<Operation>` definition	limited number of options	arbitrary string allowed in the `opType` attribute

10.3 Changes in version 0.5 & 0.5.1

PharmML element or modelling aspect	version $\leq$ 0.4.1	**version 0.5**
PK macros	*not supported*	`<PKmacros>` element with child elements, such as `<Compartment>`, ..., `<Elimination>` and their attributes `cmt`, `amount` etc.
Monolix datasets		`<MONOLIXdataSet>` and `<MONOLIXdataSetReference>` elements
Annotation reference	*not supported*	new `metadataFile` attribute in the `<PharmML>` root element
	version $\leq$ 0.5	**version 0.5.1**
Mathematical functions		new `UniOp`: `Heaviside`, `sign`
Variability model	–	new `ReferenceLevel` attribute in the `<Level>` element

10.4 Changes in version 0.6

PharmML element or modelling aspect	version $\leq$ 0.5.1	**version 0.6**
Dataset specification	`<NONMEMdataSet>` and `<MONOLIXdataSet>`	replaced by `<ExternalDataSet>` with `toolName` attribute with values `BUGS`, `Monolix` and `NONMEM`
Dataset referencing	`<NONMEMdataSetReference>` `<MONOLIXdataSetReference>`	replaced by `<ExternalDataSetReference>`
External file	`<ImportData>`	renamed to `<ExternalFile>`
Namespace URI format	.../Year/Month/...	.../pharmml/verNo/..., e.g. .../pharmml/0.6/Dataset
Multiple DV mapping	*not supported*	added `<CategoryMapping>` in the piece-wise statement

Acknowledgements

PharmML was designed and implemented by a relative small group of individuals, but its development has very much been a collaborative process. We would like to acknowledge people who influenced it and made PharmML what it is today (listed alphabetically): Andrea Mari, Alain Munafo, Andy Hooker, Benjamin Ribba, Bernard de Bono, Camille Laibe, Celine Sarr, Charlotte Kloft, Chris Franklin, Christian Laveille, Daniel Serafin, Duncan Edwards, Elodie Plan, Emmanuelle Comets, Enrica Mezzalana, Eric Blaudez, Florent Yvon, France Mentrè, Gareth Smith, Giulia Lestini, Henning Hermjakob, Henrik B. Nyberg, Iñaki F. Trocóniz, Ivan Matthews, Jonathan Chard, Joost N. Kok, Kaelig Chatel, Kajsa Harling, Lorenzo Pasotti, Lutz Harnisch, Marc Lavielle, Marylore Chenel, Mateusz Rogalski, Mats Karlsson, Mihai Glonţ, Mike Smith, Nadia Terranova, Natallia Kokash, Nick Holford, Paolo Magni, Pascal Girard, Peter A. Milligan, Phylinda Chan, Pierre Grenon, Raza Ali, Richard Kaye, Rikard Nordgren, Roberto Bizzotto, Ron Keizer, Sarah Keating, Simon Thomas, Toni Giorgino, Vijayalakshmi Chelliah, Vincent Buchheit, Zinnia P. Parra-Guillen.

We would like to acknowledge Wendy Aartsen, our scientific officer, for her outstanding support from the start of the project. The Interface Europe team with Angel Rafael, Geraldine Dupin, Landry Cochard and Elisabetta Cargnello has always been very helpful as well.

The research leading to these results has received support from the Innovative Medicines Initiative Joint Undertaking under grant agreement n° 115156, resources of which are composed of financial contributions from the European Union's Seventh Framework Programme (FP7/2007-2013) and EFPIA companies' in kind contribution. The DDMoRe project is also supported by financial contribution from Academic and SME partners.

Appendix A

Code templates for discrete data models

The code for discrete data model, as introduced briefly with two examples in Sections 9.7 and 9.8, can be quite complex compared to its continuous counterparts and the following model templates are intended to provide a quick overview of the model types covered by this PharmML version. The template are simpler to read then the full examples because here only essential elements of the structure are shown without the often lengthily mathematical formulas. These templates can be used as initial help for model encoding or just to understand the overall structure.

The full code of templates and other discrete data models examples can be downloaded from the PharmML website `http://pharmml.org`.

A.1 Count data – basic example

Simplified PharmML code for a basic Poisson count data models

```
<ObservationModel blkId="om1">
   <Discrete>
      <CountData>
         <SimpleParameter symbId="k"/>
         <CountVariable symbId="Y"/>

         <IntensityParameter symbId="Lambda">
            <!-- e.g. assignment for Lambda -->
         </IntensityParameter>

         <!-- Using UncertML -->
         <PMF linkFunction="log">
            <math:LogicBinop op="eq">
               <ct:SymbRef symbIdRef="Y"/>
               <ct:SymbRef symbIdRef="k"/>
            </math:LogicBinop>
            <PoissonDistribution xmlns="http://www.uncertml.org/3.0">
               <rate>
                  <var varId="Lambda"/>
               </rate>
            </PoissonDistribution>
         </PMF>

         <!-- ALTERNATIVELY explicit implementation -->
         <!-- log(P(Y=k)) = -Lambda+k*log(Lambda)-log(k!)-->
         <PMF linkFunction="log">
            <math:LogicBinop op="eq">
               <ct:SymbRef symbIdRef="Y"/>
               <ct:SymbRef symbIdRef="k"/>
            </math:LogicBinop>
            <ct:Assign>
               <!-- -Lambda+k*log(Lambda)-log(k!) -->
            </ct:Assign>
         </PMF>
```

```
            </CountData>
        </Discrete>
    </ObservationModel>
```

A.2 Count data – over-dispersed data models

Simplified PharmML code for count data models with over-dispersed data

```
    <ObservationModel blkId="om2">
        <Discrete>
            <CountData>
                <SimpleParameter symbId="k"/>
                <CountVariable symbId="Y"/>
                <PreviousCountVariable symbId="Yp"/>

                <Dependance type="continuousMarkov">
                    <!-- ... or discreteMarkov -->
                </Dependance>

                <IntensityParameter symbId="Lambda">
                    <!-- e.g. assignment for Lambda -->
                </IntensityParameter>

                <DispersionParameter symbId="Delta">
                    <!-- e.g. assignment for Delta -->
                </DispersionParameter>

                <OverDispersionParameter symbId="Tau">
                    <!-- e.g. assignment for Tau -->
                </OverDispersionParameter>

                <ZeroProbabilityParameter symbId="P0">
                    <!-- e.g. assignment for -->
                </ZeroProbabilityParameter>

                <MixtureProbabilityParameter symbId="Pi">
                    <!-- e.g. assignment for Pi -->
                </MixtureProbabilityParameter>

                <!-- Using UncertML -->
                <!-- log(P(Y=k)) = -Lambda+k*log(Lambda)-log(k!) -->
                <PMF linkFunction="log">
                    <math:LogicBinop op="eq">
                        <ct:SymbRef symbIdRef="Y"/>
                        <ct:SymbRef symbIdRef="k"/>
                    </math:LogicBinop>
                    <PoissonDistribution xmlns="http://www.uncertml.org/3.0">
                        <rate>
                            <var varId="Lambda"/>
                        </rate>
                    </PoissonDistribution>
                </PMF>

                <!-- ALTERNATIVELY explicit implementation -->
                <!-- log(P(Y=k)) = -Lambda+k*log(Lambda)-log(k!) -->
                <PMF linkFunction="log">
                    <math:LogicBinop op="eq">
                        <ct:SymbRef symbIdRef="Y"/>
                        <ct:SymbRef symbIdRef="k"/>
                    </math:LogicBinop>
                    <ct:Assign>
                        <!-- -Lambda+k*log(Lambda)-log(k!) -->
                    </ct:Assign>
                </PMF>
            </CountData>
        </Discrete>
    </ObservationModel>
```

Note, that the use of named distribution parameters elements as in the template, such as `<IntensityParameter>`, `<DispersionParameter>` and others, listed in section 3.7.2.1, is optional. They provide the advantage that

the translation engine to any other target tool will immediately recognise their role in the model which enhances its translation and the handling of the models in the target tool.

A.3 Nominal categorical data

Simplified PharmML code for nominal categorical models

```
<ObservationModel blkId="om1">
    <Discrete>
        <CategoricalData ordered="yes">
            <SimpleParameter symbId="p"/>

            <ListOfCategories>
                <Category symbId="cat0"/>
                ...
            </ListOfCategories>

            <CategoryVariable symbId="y"/>

            <!-- P(y = 1) = p -->
            <ProbabilityAssignment>
                <Probability linkFunction="identity">
                    <math:LogicBinop op="eq">
                        <ct:SymbRef symbIdRef="y"/>
                        <ct:SymbRef symbIdRef="cat1"/>
                    </math:LogicBinop>
                </Probability>
                <ct:Assign>
                    <ct:SymbRef symbIdRef="p"/>
                </ct:Assign>
            </ProbabilityAssignment>

            <!-- ALTERNATIVE -->
            <!-- Using UncertML -->
            <PMF linkFunction="identity">
                <un:BernoulliDistribution definition="http://www.uncertml.org/3.0">
                    <un:categoryProb definition="http://www.uncertml.org/3.0">
                        <un:name>cat0</un:name>
                        <un:probability>
                            <un:var varId="p"/>
                        </un:probability>
                    </un:categoryProb>
                </un:BernoulliDistribution>
            </PMF>
        </CategoricalData>
    </Discrete>
</ObservationModel>
```

As before, the alternative way to encode categorical models is to use the assignment based notation, as common in NONMEM. The user can simply define standard PharmML elements, such as `<SimpleParameter>` or `<Variable>` instead. The disadvantage is the lower information content of such code and its limited usability in other tools.

A.4 Ordered categorical data

Simplified PharmML code for ordered categorical models with cumulative probabilities

```
<ObservationModel blkId="om2">
    <Discrete>
        <CategoricalData ordered="yes">
            <ListOfCategories>
                <Category symbId="cat1"/>
                <!-- omitted other assignments -->
            </ListOfCategories>

            <CategoryVariable symbId="y"/>

            <!-- P(y <= 1) = a1/(a1+a2+a3) -->
            <ProbabilityAssignment>
```

```
                    <Probability>
                        <math:LogicBinop op="leq">
                            <ct:SymbRef symbIdRef="y"/>
                            <ct:SymbRef symbIdRef="cat1"/>
                        </math:LogicBinop>
                    </Probability>
                    <ct:Assign>
                        <math:Equation>
                            <math:Binop op="divide">
                                <ct:SymbRef symbIdRef="a1"/>
                                <math:Binop op="plus">
                                    <ct:SymbRef symbIdRef="a1"/>
                                    <math:Binop op="plus">
                                        <ct:SymbRef symbIdRef="a2"/>
                                        <ct:SymbRef symbIdRef="a3"/>
                                    </math:Binop>
                                </math:Binop>
                            </math:Binop>
                        </math:Equation>
                    </ct:Assign>
                </ProbabilityAssignment>
            </CategoricalData>
        </Discrete>
    </ObservationModel>
```

A.5 Ordered categorical data with Markov dependency (1^{st} and 2^{nd} order)

Simplified PharmML code for ordered categorical model with cumulative probabilities and discrete time Markov dependency for 1st and 2nd order Markov models.

```
    <ObservationModel blkId="om3">
        <Discrete>
            <CategoricalData ordered="yes">
                <SimpleParameter symbId="p01"/>
                <SimpleParameter symbId="a112"/>
                    <!-- omitted parameters -->

                <ListOfCategories>
                    <Category symbId="cat1"/>
                </ListOfCategories>

                <CategoryVariable symbId="y"/>
                <InitialStateVariable symbId="yinit"/>
                <PreviousStateVariable symbId="yp1"/>
                <PreviousStateVariable symbId="yp2"/>

                <Dependance type="discreteMarkov"/>

                <!-- Initial state probability (optional) -->
                <!-- P(y = 1) = a1 -->
                <ProbabilityAssignment>
                    <Probability>
                        <!-- yinit = 1 -->
                    </Probability>
                    <ct:Assign>
                        <ct:SymbRef symbIdRef="a1"/>
                    </ct:Assign>
                </ProbabilityAssignment>

                <!-- 1st order -->
                <!-- P(y<=1|yp1=0)=p01 -->
                <ProbabilityAssignment>
                    <Probability>
                        <CurrentState>
                            <!-- y<=1 -->
                        </CurrentState>
                        <PreviousState>
                            <!-- yp1=0 -->
```

```
                    </PreviousState>
                </Probability>
                <ct:Assign>
                    <ct:SymbRef symbIdRef="p01"/>
                </ct:Assign>
            </ProbabilityAssignment>

            <!-- 2nd order -->
            <!-- logit(P(y <= 1 | yp1 = 1, yp2 = 2)) = a112 -->
            <ProbabilityAssignment>
                <Probability linkFunction="logit">
                    <CurrentState>
                        <!-- y<=1 -->
                    </CurrentState>
                    <PreviousState MarkovOrder="1">
                        <!-- yp1=1 -->
                    </PreviousState>
                    <PreviousState MarkovOrder="2">
                        <!-- yp2=2 -->
                    </PreviousState>
                    <Condition>
                        <!-- other condition -->
                    </Condition>
                </Probability>
                <ct:Assign>
                    <ct:SymbRef symbIdRef="a112"/>
                </ct:Assign>
            </ProbabilityAssignment>
        </CategoricalData>
    </Discrete>
</ObservationModel>
```

A.6 Ordered categorical data with continuous time Markov dependency

Simplified PharmML code for ordered categorical model with cumulative probabilities and continuous Markov dependency.

```
<ObservationModel blkId="om4">
    <Discrete>
        <CategoricalData ordered="yes">
            <SimpleParameter symbId="p01"/>

            <ListOfCategories>
                <Category symbId="cat0"/>
            </ListOfCategories>

            <CategoryVariable symbId="y"/>
            <PreviousStateVariable symbId="yp1"/>

            <Dependance type="continuousMarkov"/>

            <!-- rho(y2,y1) = exp(a+b*t) -->
            <ProbabilityAssignment>
                <TransitionRate>
                    <CurrentState>
                        <!-- y=1 -->
                    </CurrentState>
                    <PreviousState>
                        <!-- yp1=1 -->
                    </PreviousState>
                </TransitionRate>
                <ct:Assign>
                    <!-- exp(a+b*t) -->
                </ct:Assign>
            </ProbabilityAssignment>
        </CategoricalData>
    </Discrete>
</ObservationModel>
```

A.7 Time-to-event data

Simplified PharmML code template for TTE data models

```
<ObservationModel blkId="om1">
    <Discrete>
        <TimeToEventData>
            <EventVariable symbId="y"/>

            <HazardFunction symbId="H">
                <!-- h0*exp(-beta*C) -->
            </HazardFunction>

            <SurvivalFunction symbId="S">
                <!-- exp(-lambda*t) -->
            </SurvivalFunction>

            <Censoring censoringType="intervalCensored">
                <IntervalLength>
                    <!-- e.g. <ct:Real>10</ct:Real> -->
                </IntervalLength>

                <RightCensoringTime>
                    <!-- e.g. <ct:Real>200</ct:Real> -->
                </RightCensoringTime>
            </Censoring>

            <MaximumNumberEvents>
                <!-- e.g. <ct:Real>5</ct:Real> -->
            </MaximumNumberEvents>
        </TimeToEventData>
    </Discrete>
</ObservationModel>
```

Bibliography

[Aho et al., 1986] Aho, A. V., Sethi, R., and Ullman, J. D. (1986). *Compilers, principles, techniques, and tools*. Addison-Wesley Pub. Co., Reading, Mass.

[Beal et al., 2006] Beal, S. L., B, S. L., and (Eds.), B. A. J. (2006). *NONMEM Users Guides, (1989-2006)*. Icon Development Solutions, Ellicott City, Maryland, USA.

[Beal et al., 2009] Beal, S. L., Sheiner, L. B., Boeckmann, A. J., and Bauer, R. J. (2009). NONMEM User's Guides. (1989-2009). Technical report, Icon Development Solutions, Ellicott City, MD, USA.

[Bertrand and Mentré, 2008] Bertrand, J. and Mentré, F. (September 2008). Mathematical expressions of the pharmacokinetic and pharmacodynamic models implemented in the Monolix software. Technical report, INSERM U738, Paris Diderot University.

[Bonate, 2011] Bonate, P. L. (2011). *Pharmacokinetic-pharmacodynamic modeling and simulation*. Springer, New York.

[CDISC, 2013] CDISC (2013). The Clinical Data Interchange Standards Consortium; link: www.cdisc.org.

[CDISC consortium, 2011] CDISC consortium (2011). CDISC Study Design Model in XML (SDM-XML), Version 1.0. Technical report, Clinical Data Interchange Standards Consortium, Inc.

[de Bono et al., 2011] de Bono, B., Hoehndorf, R., Wimalaratne, S., Gkoutos, G., and Grenon, P. (2011). The ricordo approach to semantic interoperability for biomedical data and models: strategy, standards and solutions. *BMC Res Notes*, 4:313.

[Dobson, 2002] Dobson, A. J. (2002). *An introduction to generalized linear models*. Chapman & Hall/CRC texts in statistical science series. Chapman & Hall/CRC, Boca Raton, 2nd ed edition.

[Girard et al., 1998] Girard, P., Blaschke, T. F., Kastrissios, H., and Sheiner, L. B. (1998). A markov mixed effect regression model for drug compliance. *Stat Med*, 17(20):2313–33.

[Gleeson et al., 2010] Gleeson, P., Crook, S., Cannon, R. C., Hines, M. L., Billings, G. O., Farinella, M., Morse, T. M., Davison, A. P., Ray, S., Bhalla, U. S., Barnes, S. R., Dimitrova, Y. D., and Silver, A. (2010). Neuroml: A language for describing data driven models of neurons and networks with a high degree of biological detail. *PLoS Computational Biology*, 6(6).

[Gorchetchnikov et al., 2010] Gorchetchnikov, A., Raikov, I., Hull, M., and Franc, Y. L. (2010). Network Interchange for Neuroscience Modeling Language (NineML) Specification Version 0.1. Available via the World Wide Web at `http://software.incf.org/software/nineml/wiki/nineml-specification`.

[Hucka et al., 2010] Hucka, M., Bergmann, F. T., Hoops, S., Keating, S. M., Sahle, S., Schaff, J. C., Smith, L. P., and Wilkinson, D. J. (2010). The Systems Biology Markup Language (SBML): Language Specification for Level 3 Version 1 Core. Available via the World Wide Web at `http://sbml.org/Documents/Specifications`.

[Hucka et al., 2003] Hucka, M., Finney, A., Sauro, H. M., Bolouri, H., Doyle, J. C., Kitano, H., , the rest of the, S. F., Arkin, A. P., Bornstein, B. J., Bray, D., Cornish-Bowden, A., Cuellar, A. A., Dronov, S., Gilles, E. D., Ginkel, M., Gor, V., Goryanin, I. I., Hedley, W. J., Hodgman, T. C., Hofmeyr, J. H., Hunter, P. J., Juty, N. S., Kasberger, J. L., Kremling, A., Kummer, U., Le Novère, N., Loew, L. M., Lucio, D., Mendes, P., Minch, E., Mjolsness, E. D., Nakayama, Y., Nelson, M. R., Nielsen, P. F., Sakurada, T., Schaff, J. C., Shapiro, B. E., Shimizu, T. S., Spence, H. D., Stelling, J., Takahashi, K., Tomita, M., Wagner, J.,

and Wang, J. (2003). The systems biology markup language (SBML): a medium for representation and exchange of biochemical network models. *Bioinformatics*, 19(4):524–531.

[Inria POPIX, 2013] Inria POPIX (2013). Mixed Effects Models for the Population Approach - Wiki Popix.

[Keizer and Karlsson, 2011] Keizer, R. and Karlsson, M. (2011). Stochastic models. Technical report, Uppsala Pharmacometrics Research Group, Uppsala.

[Kernighan and Ritchie, 1988] Kernighan, B. W. and Ritchie, D. M. (1988). *The C Programming Language*. Prentice Hall Professional Technical Reference, 2nd edition.

[Lavielle, 2014] Lavielle, M. (2014). *Mixed Effects Models for the Population Approach: Models, Tasks, Methods and Tools*. Chapman & Hall/CRC Biostatistics Series.

[Lavielle and Grevel, 2011] Lavielle, M. and Grevel, J. (September 1, 2011). WP6.1 Clinical Trial Simulator: Prototype 01. Description of capabilities. Technical report, INRIA Saclay.

[Lavielle and Lixoft Team, 2012] Lavielle, M. and Lixoft Team (2012). MLXTRAN, A Brief Overview. Technical report, INRIA Saclay & Lixoft.

[Lavielle and Lixoft Team, 2014] Lavielle, M. and Lixoft Team (2014). MLXTRAN, The model coding language for Monolix. Technical report, INRIA Saclay & Lixoft.

[Li et al., 2010] Li, C., Donizelli, M., Rodriguez, N., Dharuri, H., Endler, L., Chelliah, V., Li, L., He, E., Henry, A., Stefan, M. I., Snoep, J. L., Hucka, M., Le Novère, N., and Laibe, C. (2010). BioModels Database: An enhanced, curated and annotated resource for published quantitative kinetic models. *BMC Systems Biology*, 4:92.

[Lixoft, 2012] Lixoft (2012). Monolix 4.1.4 - User Guide. Technical report, Lixoft.

[Lixoft, 2014a] Lixoft (2014a). Monolix 4.3 - Model MLXTRAN Tutorial. Technical report, Lixoft.

[Lixoft, 2014b] Lixoft (2014b). Monolix 4.3.2 - User Guide. Technical report, Lixoft.

[Moodie et al., 2013] Moodie, S., Swat, M. J., Kristensen, N. R., and Le Novère, N. (2013). PharmML: The Pharmacometrics Markup Language, Language Specification Version 0.2.1.

[Nielsen and Halstead, 2004] Nielsen, P. F. and Halstead, M. D. (2004). The evolution of CellML. In *Engineering in Medicine and Biology Society, 2004. IEMBS '04. 26th Annual International Conference of the IEEE*, volume 2, pages 5411–5414.

[NLME Consortium, 2008] NLME Consortium (2008). Pharmacometrics Markup Language (PharML), Level 1, Version 1. Technical report, Novartis Pharma AG, Novo Nordisk A/S, Servier and Uppsala University.

[Parr, 2010] Parr, T. (2010). *Language implementation patterns: create your own domain-specific and general programming languages*. The pragmatic programmers. Pragmatic Bookshelf, Raleigh, N.C.

[Paule et al., 2012] Paule, I., Girard, P., Freyer, G., and Tod, M. (2012). Pharmacodynamic models for discrete data. *Clin Pharmacokinet*, 51(12):767–86.

[Plan et al., 2009] Plan, E. L., Maloney, A., Trocóniz, I. F., and Karlsson, M. O. (2009). Performance in population models for count data, part i: maximum likelihood approximations. *J Pharmacokinet Pharmacodyn*, 36(4):353–66.

[RDF Working Group, 2014] RDF Working Group (2014). Resource Description Framework (RDF). Available at `http://www.w3.org/RDF/`.

[Ribba et al., 2012] Ribba, B., Kaloshi, G., Peyre, M., Ricard, D., Calvez, V., Tod, M., Cajavec-Bernard, B., Idbaih, A., Psimaras, D., Dainese, L., Pallud, J., Cartalat-Carel, S., Delattre, J.-Y., Honnorat, J., Grenier, E., and Ducray, F. (2012). A tumor growth inhibition model for low-grade glioma treated with chemotherapy or radiotherapy. *Clin Cancer Res*, 18(18):5071–80.

[Smith and Holford, 2012] Smith, M. and Holford, N. (December 10, 2012). Warfarin PK Rosetta Stone. Technical report, Pfizer & Uppsala University.

[Swat et al., 2015] Swat, M. J., Wimalaratne, S. M., Kristensen, N. R., Yvon, F., Bizzotto, R., and Lavielle, M. (2015). PK Macros in PharmML 0.6. Technical report, EMBL-EBI, Hinxton, UK; Novo Nordisk A/S, Bagsværd, DK; Babraham Institute, Babraham, UK.

[The Data Mining Group (DMG),] The Data Mining Group (DMG). Predictive model markup language, v4.2.1.

[UncertML Team, 2014] UncertML Team (2014). Uncertainty Markup Language: UncertML Version 3.0. Avaiable at `http://www.uncertml.org`.

[W3C, 2010] W3C (2010). Mathematical Markup Language (MathML) Version 3.0.

[Waltemath et al., 2011] Waltemath, D., Bergmann, F. T., Adams, R., and Novère, N. L. (2011). Simulation Experiment Description Markup Language (SED-ML): Level 1 Version 1. Available via the World Wide Web at `http://sedml.org/specifications.html`.