Drug Disease Model Resources

ddmore

PharmML: The Pharmacometrics Markup Language

Language Specification

Version 0.2.1

Stuart L. Moodie[1,2], Maciej J. Swat[1,2]
Niels R. Kristensen[3] and Nicolas Le Novère[2,4]

25 Sep 2013

[1]These authors contributed equally to the work
[2]EMBL-EBI, Hinxton, UK
[3]Novo Nordisk A/S, Bagsværd, DK
[4]Babraham Institute, Babraham, UK

Acknowledgements

> When you know a thing, to hold that you know it; and when you do not know a thing, to allow that you do not know it - this is knowledge.
> *Confucius, The Confucian Analects*

In developing this latest version of PharmML we are building on work we documented in the previous version of this specification, the contributions to which we acknowledged (see section A.1). This latest version has been developed over a shorter timeframe and during the summer vacation period so consequently we are acknowledging a smaller number of people. As ever we would like to acknowledge the input of Marc Lavielle: the changes to the parameter and error model in this version we were considerably influenced by his input. During the early stages of development we sent out a proposal document, asking for feedback and we are grateful to Mike Smith, Emmanuelle Comets, Duncan Edwards and Marc Lavielle for the useful feedback they gave to these ideas. We are also indebted to Paolo Magni for his work encoding models in the previous version of PharmML. In doing so he highlighted some limitations in that version and helped spark some useful discussion that helped us improved this version. Duncan Edwards also provided excellent review comments to version 0.1.0 and the changes to the changes we define data and represent the variability model are all attributable to him. Roberto Bizzotto helped us to understand the different flavours of the residual error models and their encoding in MLXTRAN and NMTRAN. Since the last release there have been a number of discussions on the DDMoRe WP4 mailing list and we would like to thank the contributors (most of whom we mentioned already) include Andy Hooker and Nick Holford. We would like to thank Nick in particular for asking us to encode the model described by Chan *et al.* [Chan et al., 2005]. Together with Phylinda Chan, Nick helped us understand what was a challenging trial design that gave us a great test case to work on. Vincent Buchheit and his EFPIA colleagues provided valuable feedback on different trial design aspects. We would like to thank Mihai Glont and Raza Ali for scrutinising the specification during their work on the DDMoRe repository. Raza in particular provided some useful feedback on the maths section in the language overview chapter. Sarah Keating helped us understand the complexity of unit consistency checking and although this feature is not in this version, we believe her experience will help us get it right in the future.

We'd also like to thank Lutz Harnisch for his leadership of the DDMoRe project and the administrative team at Interface Europe for their help ensuring its smooth running. Finally we would like to thank Wendy Aartsen for many things, but specifically for organising the April consortium meeting, which provided such an excellent forum for the language specification.

The DDMoRe project and consequently this work, is funded by the Innovative Medicines Initiative (IMI), a large-scale public-private partnership between the European Union and the pharmaceutical industry association EFPIA. We would like to gratefully acknowledge their support.

Contents

Part I

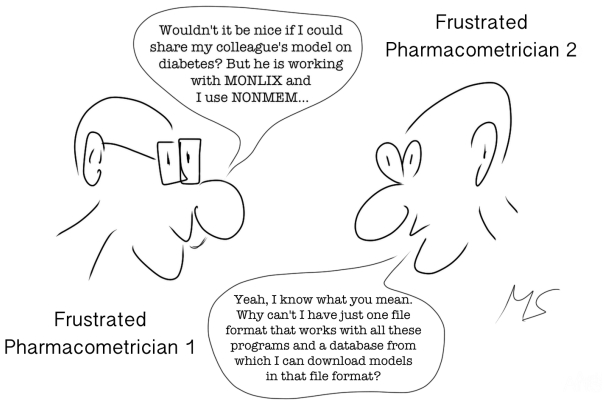
PharmML Primer

CHAPTER 1

What is PharmML?

1.1 The Problem

The cartoon on the right summarises the principal problem that PharmML addresses: namely, the reliable exchange of pharmacometric models between software tools. What we are aiming for is illustrated in figure 1.1, where PharmML is the exchange medium for pharmacometric models for the main modelling and simulation tools in the field. This is not an unreasonable goal and has been successfully realised in the field of Systems Biology.



In Systems Biology the problems our Frustrated Pharmacometricans complain of do not exist. Software tools exchange models using the Systems Biology Markup Language (SBML; `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 their favourite 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.

This lesson has not be 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].

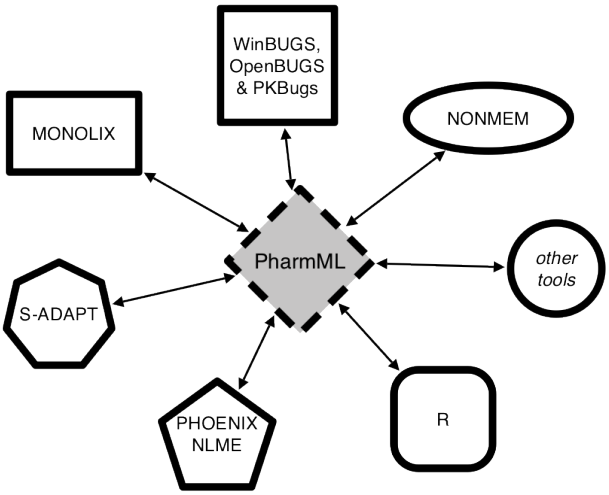


Figure 1.1: Interoperability platform to exchange models via PharmML.

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].

So what is the solution to the problem? PharmML. 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) to help tool providers incorporate support of PharmML and to facilitate its general adoption in the field.

So there is hope for the Frustrated Pharmacometricians. PharmML will solve the problem.

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.

[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.

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.

1.3 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 enormous. As the standard evolves their role in using and then promoting the standard to the wider community will be invaluable.

1.4 How PharmML was developed

We, the authors, have organised and designed PharmML, but its development has very much been a collaborative process. When embarking on 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.

The use-case based development cycle is illustrated in figure 1.2 and you can see how the generation of 'executable' prototype models was important in ensuring assumptions were correct before expanding the scope of the design. For this to be possible the use cases were documented with a mathematical description and the expected outputs and results of the model were also described in detail [Swat and Moodie, 2013]. In some cases reference MATLAB® implementations were also included to assist prototype code generation. Although this development cycle was our objective, in the later stages there was insufficient time to upgrade the code generation framework to produce executable models from a PharmML document. Consequently some parts of the specification, in particular the parts related to estimation tasks, have not been tested as thoroughly as we would like.

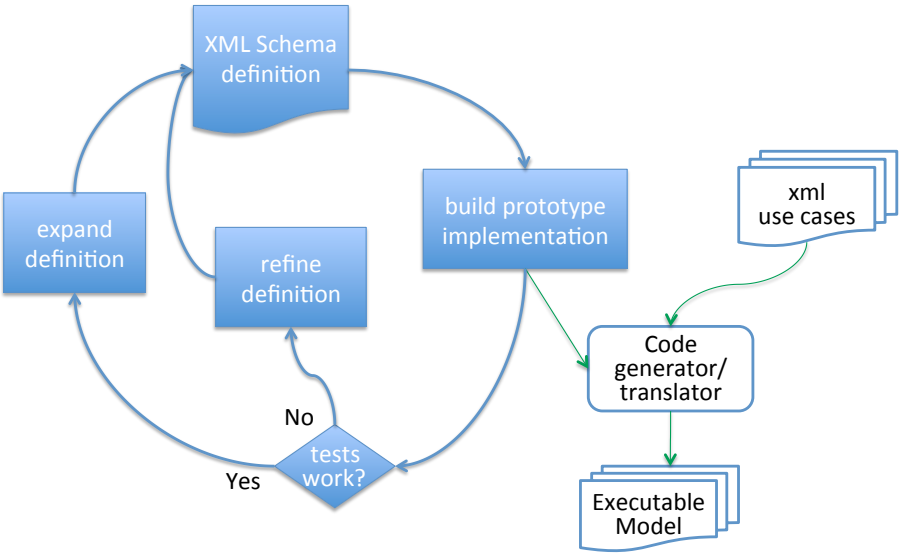


Figure 1.2: An overview of the development cycle adopted by the PharmML team.

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

- The DDMoRe consortium meeting in Leiden, 24–26 Apr 2012.
- The DDMoRe consortium meeting in Noordwijkerhout, 11–13 Sep 2012.
- The DDMoRe technical workshop hosted by Novo Nordisk in Copenhagen, 28–30 Jan 2013.

1.5 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 usually many ways to do the same thing in the imperative language. Therefore, generating a PharmML document from a NONMEM control stream may be challenging.

1.6 The Evolution of PharmML

This document represents the start of an evolution for PharmML. 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 as it is subject to the same influences. To manage this process we plan to adopt the following strategy to record versions of the PharmML specification.

1.6.1 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. Not backwards compatible.

y **minor** New features or evolutionary design changes. Always backwards compatible.

z **patch** Error corrections. Always backwards compatible.

By backwards compatible we mean that a PharmML parser that supports version 1.9.0 must be able to read a file that is compliant with version 1.4.0 of the PharmML specification.

1.6.2 Release Process

Before each release we will submit a release candidate of the specification to the community for comments and criticism. This is typically called a Request for Comments (RFC) and is standard practise. Based on review comments the specification will either be revised, possibly significantly, and submitted for review again or (if revisions are minor) revised and released.

The RFC period for this specification will end on 26 April 2013.

1.7 Feedback

Whether it is during the RFC process or after a release your feedback is important to us. Typically, this feedback will be in the form of a specific issue: either to report defects you have identified in the specification or to request new features. Either way we would urge you to submit specific issues to our tracker at: `sf.net/ddmore/tracker`. In some cases you may wish to ask for clarification or raise an issue that is broader than a specific issue or requires some discussion within the community. Here you should contact the PharmML forum at `sf.net/ddmore/forum`.

CHAPTER 2

This Document

2.1 Overview

In this specification we describe the PharmML standard. We define what the language is, how it is encoded, how it should be understood, and how software can validate it. Initially, we define in chapter 3 what is in and out of scope of the current version of PharmML. Underlying PharmML is a mathematical model that describes a pharmacometric model. Understanding this is important if you are to completely understand the language. In chapter 4 we go through this model in detail, providing a detailed mathematical description of the model and its associated assumptions. Also important for the understanding of PharmML is the discussion of trial designs in chapter 5, where the basic assumptions behind the way trial designs are handled in PharmML are discussed. Next we move on to the description of the implementation of PharmML. In chapter 6 we describe the organisation of a PharmML document and then go on to explain key features of the language, such as variable scoping, that are important to understand before investigating the language further. Chapter 7 explains the details of PharmML using examples. These examples are designed to highlight specific features of the language and by taking you through a series of worked examples we hope to help you to fully understand PharmML. The final chapter in Part I (Chapter 8) lists a number of unresolved issues.

At the end of this document we have the technical reference (Part II on page 101). This provides the fine detail of the technical implementation of PharmML, including the XML Schema design and the rules to which a correct PharmML document must conform. This is reference material and not intended to be read like a novel from start to finish.

To conclude, helping you to understand PharmML is the main goal of this specification. We want you to use this document to review and critique PharmML. We want you to suggest improvements to the language. Most of all we want you to use it.

2.2 How to read this Specification

PharmML is a language of benefit to pharmacometric modellers, but ironically is not designed to be read by them when in every day use. We expect software support for PharmML to be developed by software engineers, who do not have a deep understanding of pharmacometrics and modelling. Therefore, the challenge for us in drafting this specification is to satisfy both readerships: modellers and software engineers. To help, we have come up with the following advice on how each readership could read this specification.

If you are a pharmacometric modeller or mathematician then you will want to start with the mathematical definition of PharmML in chapter 4 (page 13). From there you may want to skim the language

overview (chapter 6) before working through the examples in chapter 7 (page 58). You may want to revisit chapter 6 (page 37) after reading the examples.

If you are a software engineer then we recommend that you start with the language overview in chapter 6 (page 37). After this, work through the examples in chapter 7 (page 58) and try to understand the language features in this way. Those of you with a strong mathematics background will find it helpful to read through the mathematical definition as well (chapter 4). Finally, when it comes to implementing PharmML support, you will find the technical reference (Part II on page 101) very important, but only after you have an understanding of PharmML from Part I.

CHAPTER 3

Features supported by PharmML

3.1 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. In particular what information a PharmML document can represent and what it cannot. As is common practise in software engineering we have described the functionality as a set of "features". These features are said to be *current* if they are provided by this release of PharmML, *planned* if they are not in the current version, but planned for a future release. It is important to remember that this specification is just the beginning of PharmML and over time we expect many of the features currently out of scope 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.

3.2 Current Features

3.2.1 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.

3.2.2 Model Definition

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.* Such systems of ODEs can include algebraic equations as well.

– *A structural model defined in SBML format.*

– *A structural model from an external resource or library.*

– *Standard PK models* Including those encoded by Monolix [Bertrand and Mentré, 2008] or PREDPP in NONMEM [Beal et al., 1992].

– *Standard PD models.* Including those defined by Monolix [Bertrand and Mentré, 2008].

Covariate Model

PharmML can encode:

– *Continuous covariates.* These can be sampled from a probability distribution and used with an applied transformation.

– *Categorical covariates.* These can also be sampled from a probability distribution.

Parameter Model

PharmML can encode:

– *The population mean/typical value for a parameter.*

– *Linear covariate model.*

– *Random effects at arbitrary levels of variability.*

– *Correlation of the random effects, described by a correlation or covariance matrix.*

– *Non-linear parameter models, such as those described in [Keizer and Karlsson, 2011].*

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.

Observations Model

PharmML currently only supports the following observation model:

– *Continuous observation model.* A residual error model applied to one or more variables in the structural model.

– *Autocorrelation of the residual errors in a continuous observation model.*

3.2.3 Trial Design

PharmML can encode the following features of a trial design *explicitly*[1]:
– *Bolus dosing.*
– *Infusion dosing.*
– *Multiple dosing regimens including mixed bolus and infusion.*
– *Repeated dosing.*
– *Dosing at arbitrary time points.*
– *Steady state dosing.*
– *Dosing to more than one compartment.*
– *Cross-over designs.*
– *Parallel designs.*
– *Washout periods.*
– *Run-in periods.*
– *Occasions – defined by time interval within a treatment epoch.*
– *Trials with different centres or other levels of organisation above study groups (aka arms)*

3.2.4 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*

3.3 Planned Features

3.3.1 General

– *Units* Support for unit definitions and unit consistency checking.

3.3.2 Model Definition

Covariate Model

PharmML does not support the following covariate related features:
– *Conditional distributions of continuous covariates.*
– *Clusters of categorical covariates.*
– *Selection/exclusion criteria for covariates.*

Structural Model

PharmML does not support the following model types:
– *(Hidden) Markov models.*
– *Delay Differential Equations (DDEs).*
– *Partial Differential Equations (PDEs).*
– *Stochastic Differential Equations (SDEs).*

[1] As opposed to the implicit trial design definition present within a data file.

Observations Model

PharmML does not support the following types of observation models:

– *Count data models.* Poisson, negative binomial, zero-inflated Poisson models etc.

– *Nominal and ordered categorical models.* Logistic regression, proportional odds models etc.

– *Time-to-event models.* Parametric (e.g. exponential, Gompertz, Weibull) or semi-parametric Cox models.

3.3.3 Trial Design

PharmML cannot encode the following features in a trial design:

– *Reset of all or single compartments.*

3.3.4 Modelling Steps

PharmML cannot encode the following features related to the task(s) associated with a model:

– *Bayesian inference methods.*

– *Writing estimation results to a file or other external resource.*

– *Writing simulation results to a file or other external resource.*

– *Exchange of results from one modelling step to another.* Currently it is not possible to pass on the results of an estimation task to a subsequent estimation step.

– *Model exploration* Tasks such as sensitivity analysis are not supported.

– *Optimal design*

CHAPTER 4

Mathematical Representation

All models are wrong but some are useful
George E.P. Box

4.1 Introduction

This chapter deals with the mathematical description of models which can be encoded in PharmML. The figure 4.2 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 4.1.

Additional aspect is the difference between individual and population approach. In the former case,

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

Figure 4.1: A basic equation visualising the relationship between experimental data and model prediction.

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 4.2. In clinical practise however, the situation is quite different, Figure 4.3. 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.

4.2 Non-linear mixed effect models

The approach which proved to be very effective for the analysis of population data is that of **non-linear mixed effect models**, NLME. The nonlinearity means that it can handle 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 4.6.5.

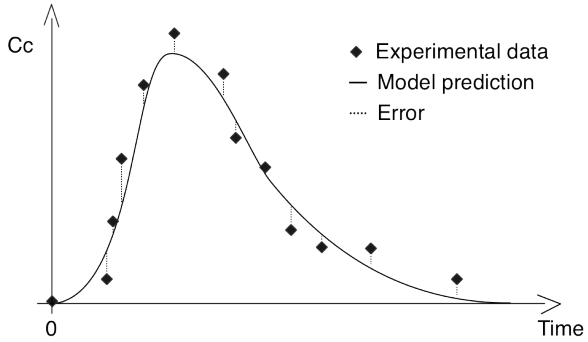


Figure 4.2: Frequently sampled individual PK data. The black diamonds stand for experimental data, the solid line for the time course of concentration as predicted by a mathematical model.

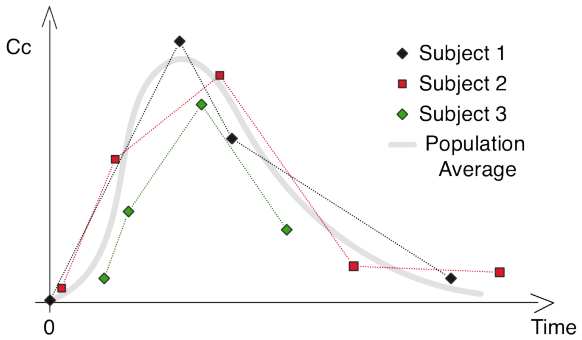


Figure 4.3: Population PK data – for three subjects with different observation times and varying characteristics, such as area under the curve, maximum values, time of the maximum etc. The grey line of the population average is to be estimated along with individual estimates.

- **random effects** – subject/occasion specific, e.g. *inter-individual* or *inter-individual within subject within centre variability*, see section 4.5 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 4.3 we formulate a general nonlinear mixed effects model for continuous data, section 4.4 introduces the structural model, section 4.5 discusses the variability as nested hierarchy, section 4.6 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 4.7.1 is about the residual error model.

4.3 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, 2012]):

$$\underbrace{y_{ij}}_{\substack{\text{Experimental}\\ \text{data}}} = \underbrace{f(x_{ij},\psi_i)}_{\substack{\text{Model}\\ \text{prediction}}} + \underbrace{g(x_{ij},\psi_i,\xi)\,\epsilon_{ij}}_{\text{Error}} \qquad 1 \le i \le N, \quad 1 \le j \le n_i \tag{4.1}$$

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

4.4 Structural model

This section deals with the first term of the right hand side in eq.4.1

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

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

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

4.5 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 Copenhagen focus meeting [DDMoRe/Copenhagen meeting, 2013]. 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 4.7.1, while the latter is described in this section.

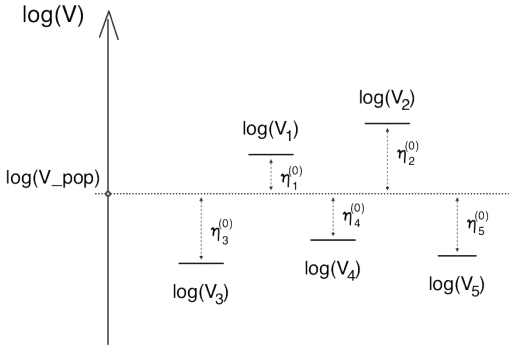


Figure 4.4: Inter-individual variability typically occurring in an experiment, here $\log(V_{i=1\cdots5})$ i.e. values for five subjects, varying around a typical value $\log(V_{pop})$, are shown.

4.5.1 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 4.4. 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 4.5. 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' as a new variability level. If a clinical trial has been conducted in various countries, it is reasonable to ask if the geographic location influences the outcome of the study.

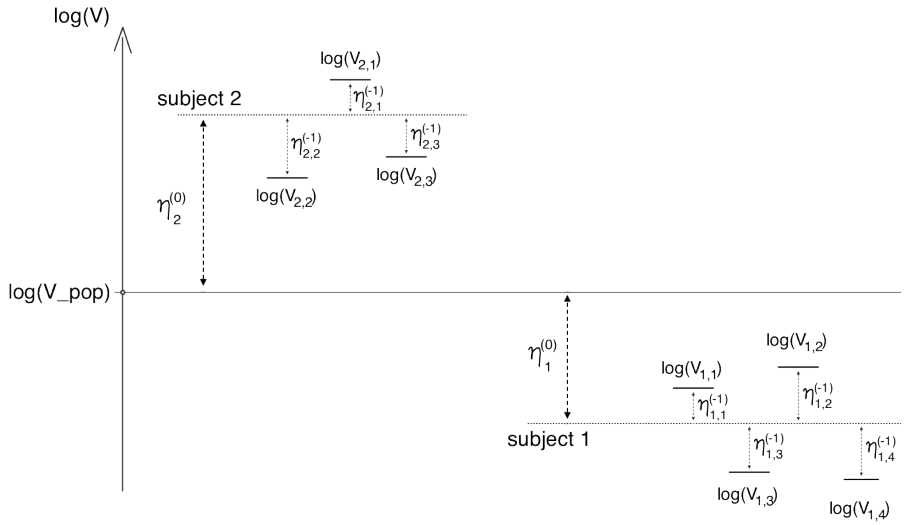


Figure 4.5: Subject variability level, (0), and within-subject (or occasion) variability level, (-1), typically occurring in an experiment, with index i for subjects and k for occasion. Here two subjects only are visualised, each of them having four or three occasions, respectively.

4.5.2 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 4.6, 4.7.

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 4.6. 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 '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 'zero' 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} – also called BSV (between subject variability) or IIV (inter-individual variability).
- Ω^{-n} – for levels below the 'zero' level – called WSV (within-subject variability) or IOV (inter-occasion 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 or use 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 4.8. The following symbols are used

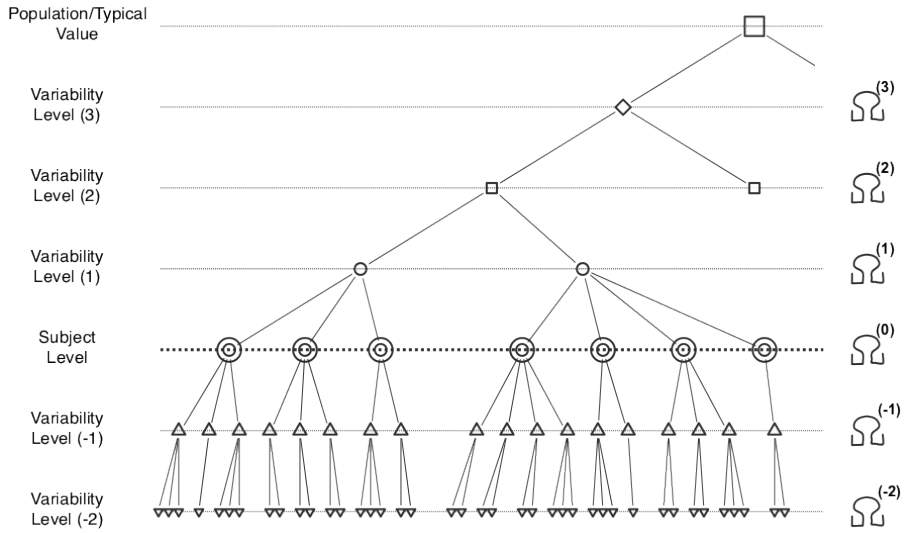


Figure 4.6: General nested hierarchy of the variability structure – as tree. Note that PharmML doesn't require or use numbers to be assigned to the various levels of variability. Instead the user can define meaningful identifiers.

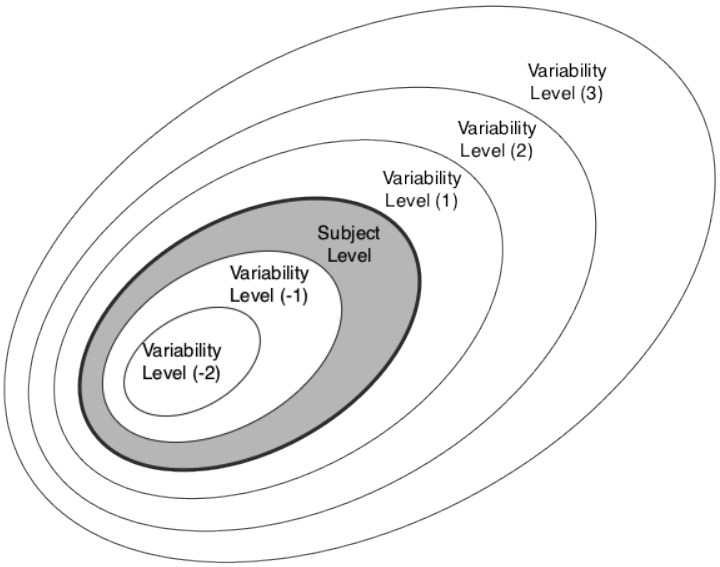


Figure 4.7: General nested hierarchy of the variability structure – as Venn diagram.

- 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 $\eta_i^{(0)} \sim \mathcal{N}\left(0, \Omega^{(0)}\right)$.

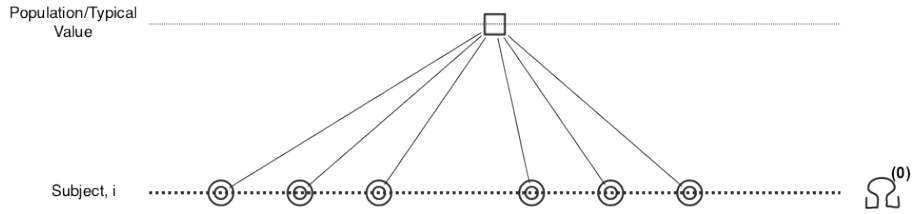


Figure 4.8: Example 1 – single level of variability: *subject*– level

Example 2 In this example there are three levels of variability: {*centre, subject, occasion*}, see Figure 4.9. 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

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

Example 3 In this example there are four levels of variability: {*country, centre, subject, occasion*}, see Figure 4.10. 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*

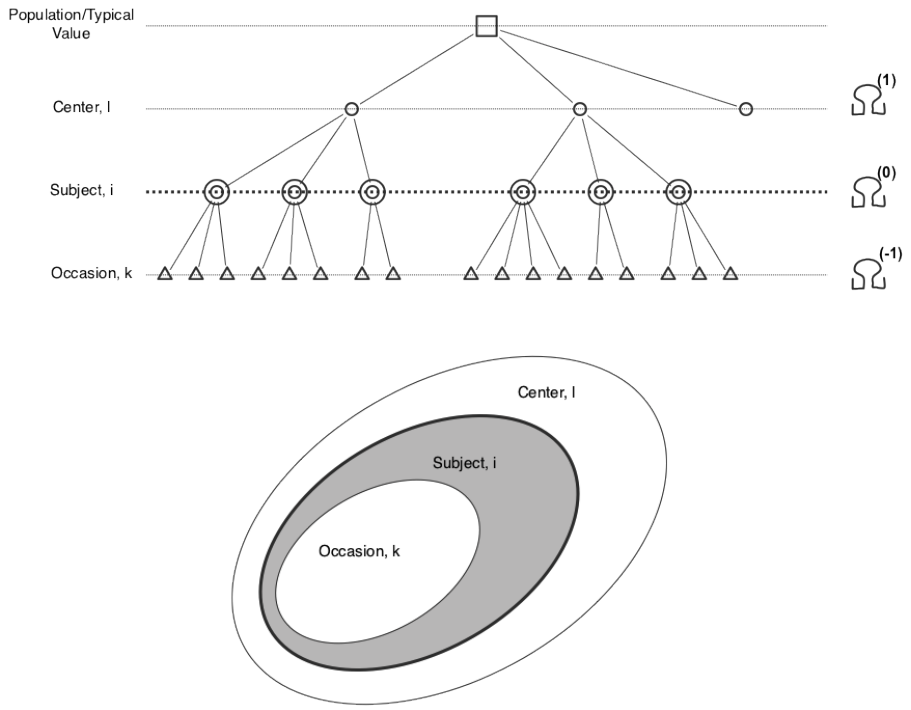


Figure 4.9: Example 1 – three levels of variability: {*centre, subject, occasion*}

- 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

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

4.6 Parameter model

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

- Type 1. Equation type

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

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.

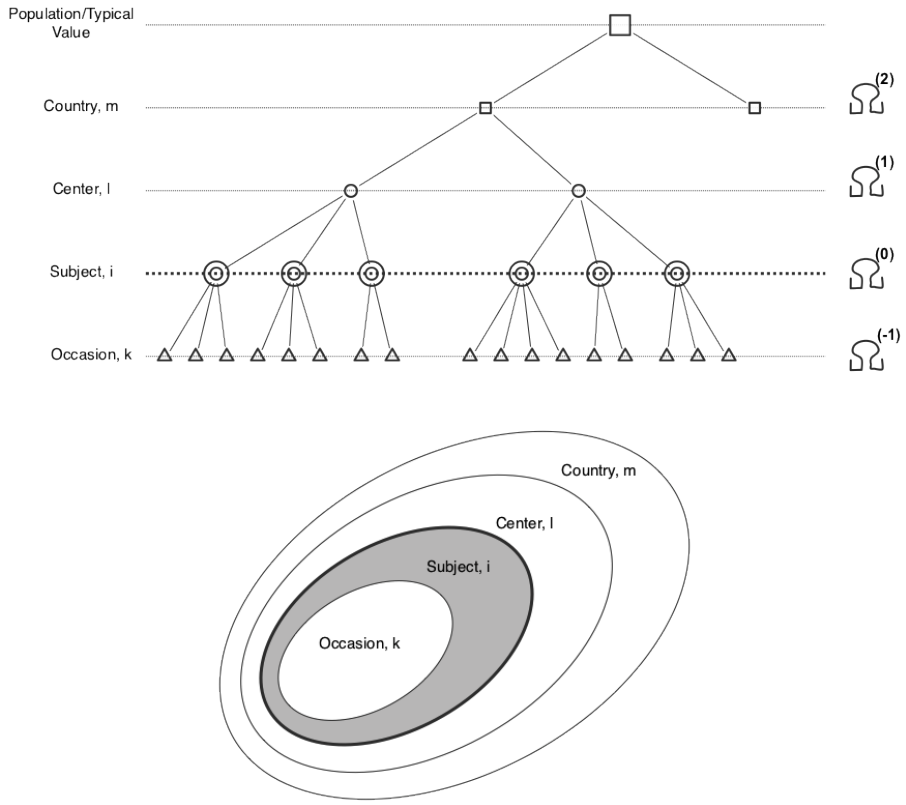


Figure 4.10: Example 2 – four levels of variability: {*country, centre, subject, occasion*}

- 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.

4.6.1 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.

4.6.2 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 4.5).

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

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

4.6.3 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 4.5).

See Figure 4.11 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 4.5 (see Figure 4.9), 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

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

with

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

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, $\eta_{lik,V}^{(-1)}$.

4.6.4 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.

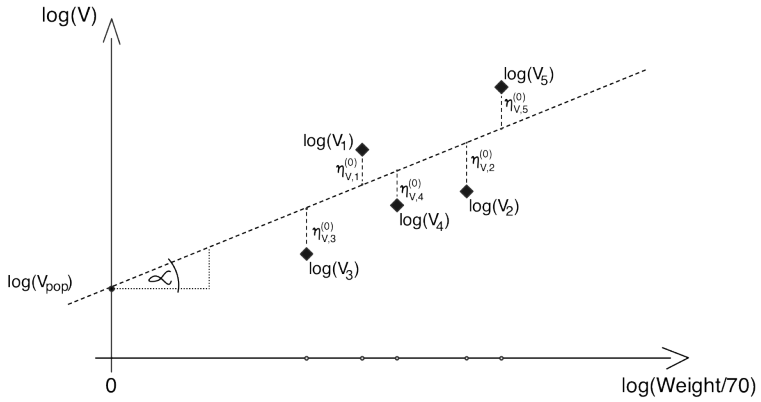


Figure 4.11: The linear relationship between the parameter and a continuous covariate after application of appropriate transformations $V_i \longrightarrow \log(V_i)$ and $W \longrightarrow \log(W/70)$ with $\beta_{V,2} = \tan\alpha$, the slope of the regression line, and $\log(V_{pop})$ as the y-axis 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

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

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

4.6.5 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 4.11). 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 7.2.1) its probability distribution has to be estimated. The information we have to provide is summarised in the

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

Categorial covariate model

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

4.6.6 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.

Log-Normal distributed

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

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

for a typical value V_{pop} and standard deviation ω_V as described in [Lavielle, 2012].
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}
```

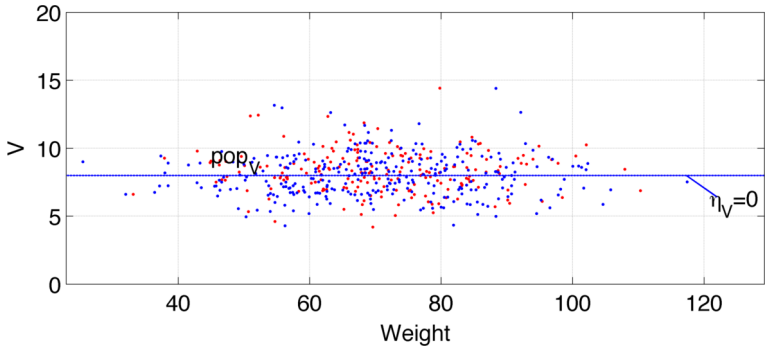


Figure 4.12: Log-normally distributed 'V' with $V_{pop} = 8$ and $\omega_V = 0.2$

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

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

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

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

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

Logit-Normal distributed

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

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

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

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

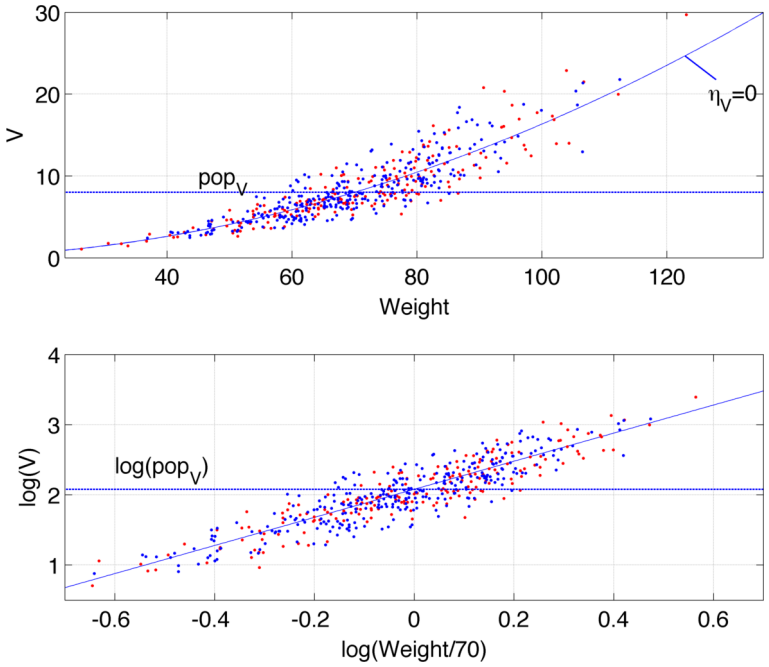


Figure 4.13: Log-normally distributed 'V' with 'Weight' as covariates.

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

```
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}
```

Logit-Normal distributed with a continuous covariate

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

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

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

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

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}
```

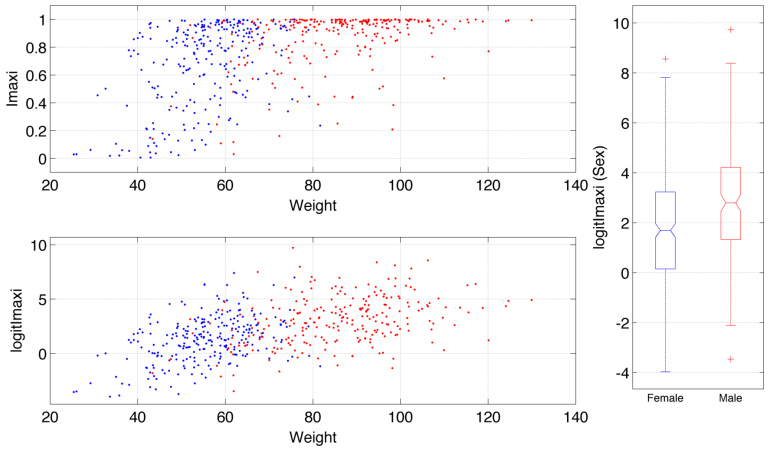


Figure 4.14: Logit-normally distributed 'Imax' with 'Weight' as covariate.

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 4.9, reads as follows

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

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

with

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

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

4.7 Observation model

Figure 4.15 gives an overview of the Observation Model as implemented in the current version of PharmML, which covers only continuous data models. A future release will cover discrete data models, such as categorical, count and time-to-event (greyed out in the figure). An essential component of

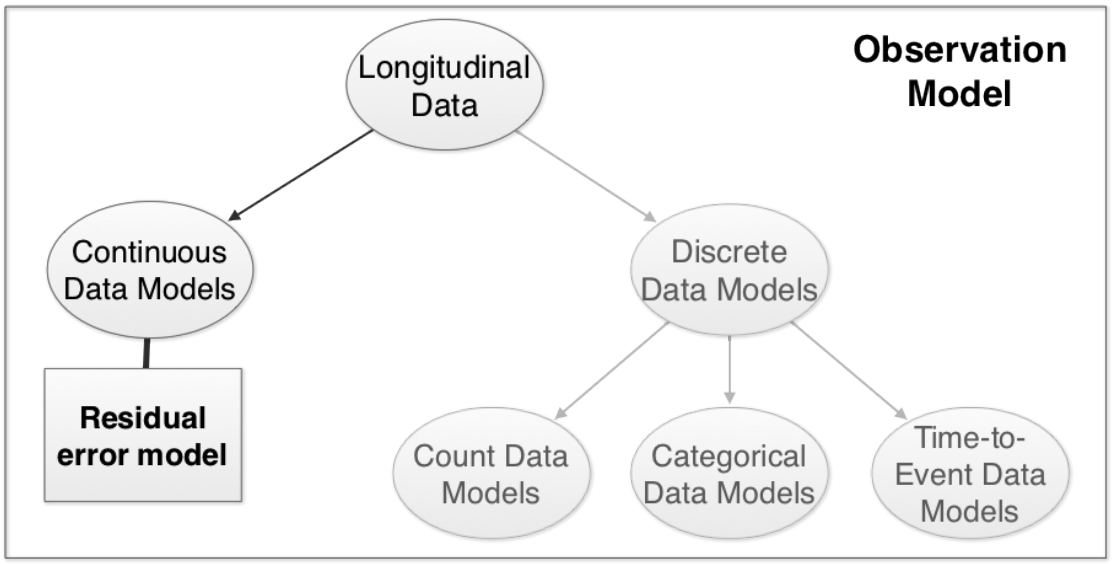


Figure 4.15: Observation Model

the Observation Model is the Residual Error Model, which applies only to continuous data models.

4.7.1 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.4.1 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

$$\underbrace{u(y_{ij})}_{\substack{\text{Transformed}\\ \text{experimental data}}} = \underbrace{U\big(f(x_{ij},\psi_i),\xi_i,\epsilon_{1,ij},\epsilon_{2,ij},\dots\big)}_{\substack{\text{Transformed}\\ \text{model prediction}}} \tag{4.2}$$

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 4.5, is applied to parameters of the residual error model (see section 4.7.2 for details).

4.7.2 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 4.16. For example, if an additive residual error model and a log-normal distribution for a is

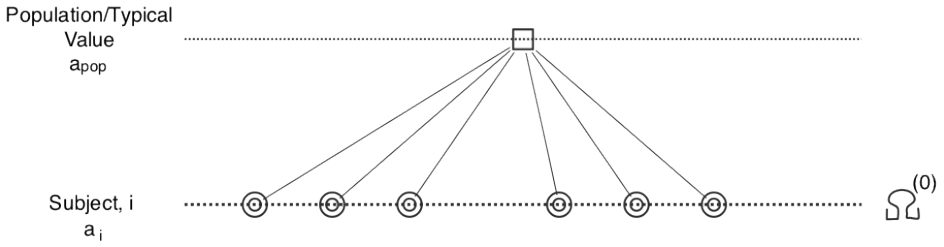


Figure 4.16: Inter-individual variability of the residual error parameter a. The nested hierarchical structure is identical to that of structural model parameters.

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

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

4.7.3 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):

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

- Combined additive and proportional 1:

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

- Combined additive and proportional 2:

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

- 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:

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

- Combined additive and power error model 2:

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

- 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.

CHAPTER 5

Trial design model

5.1 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.

5.1.1 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

[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].

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.

5.2 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'.

5.2.1 Structure

To define the Trial Structure we have reused, almost verbatim, the CDISC Study Design Model [CDISC SDM-XML Technical Committee, 2011], which is an XML representation of a clinical trial. Figure 5.1 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.

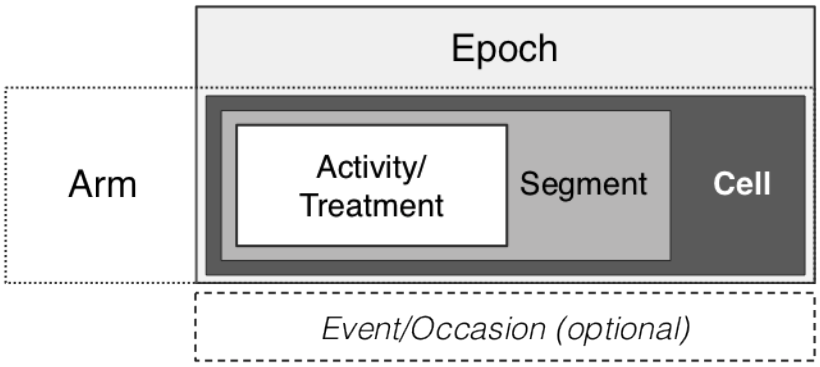


Figure 5.1: Overview of the basic concept of the Trial Structure: arm, epoch, event and a cell with segment and activity/treatment as used in the CDISC Study Design Model. See the next figure for an example.

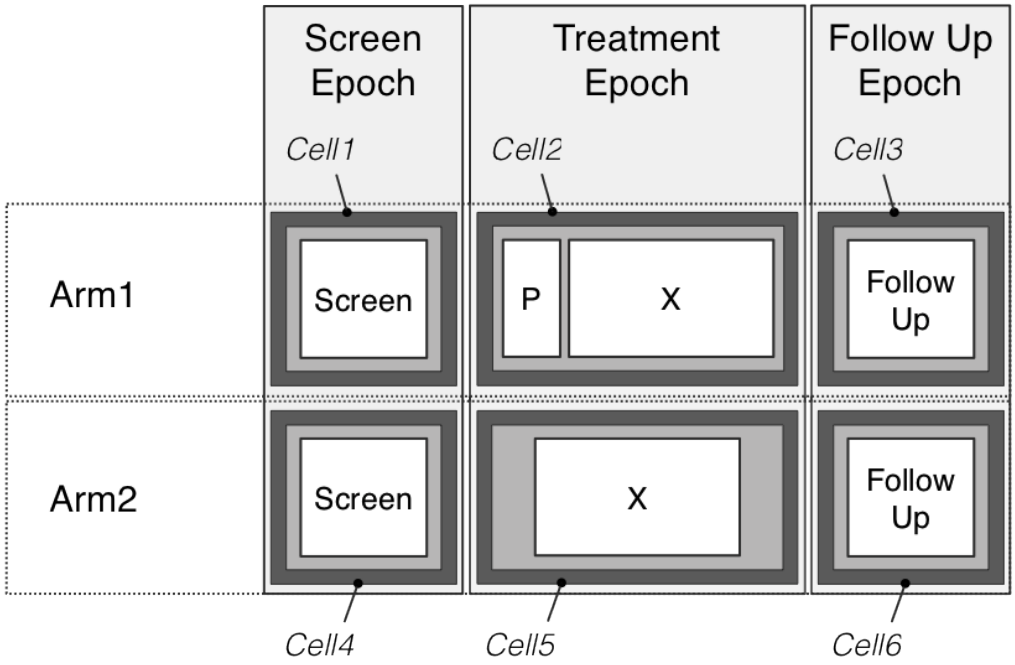


Figure 5.2: An example of a study with two arms and three epochs: Screen, Treatment and Follow Up. A segment can contain more then one Activity/Treatment as can be seen in $Cell2$ with one Pre-Treatment P and one Treatment X. In this particular example no Event/Occasion is specified.

Figure 5.2 shows one such example of a hypothetical trial consisting of two arms and three epochs: $Screen$, $Treatment$ and $FollowUp$. There are accordingly six cells and segments, each consisting of one or two 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 7).

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 4.5 where the full nested hierarchy of the random variability discussed in detail.

5.2.2 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.

5.2.3 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 5.2.1).

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 6

Language Overview

6.1 Introduction

In this chapter we will provide you with the background knowledge you need to understand PharmML. We recommend that you read this chapter before you work through the examples in chapter 7. The chapter will start 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 6.5 and 6.7), and how to define data (section 6.6). 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 6.12).

6.2 Organisation

As can be seen in figure 6.1, PharmML is organised into three main sections: *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 will go into more detail about 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.

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 4.5.

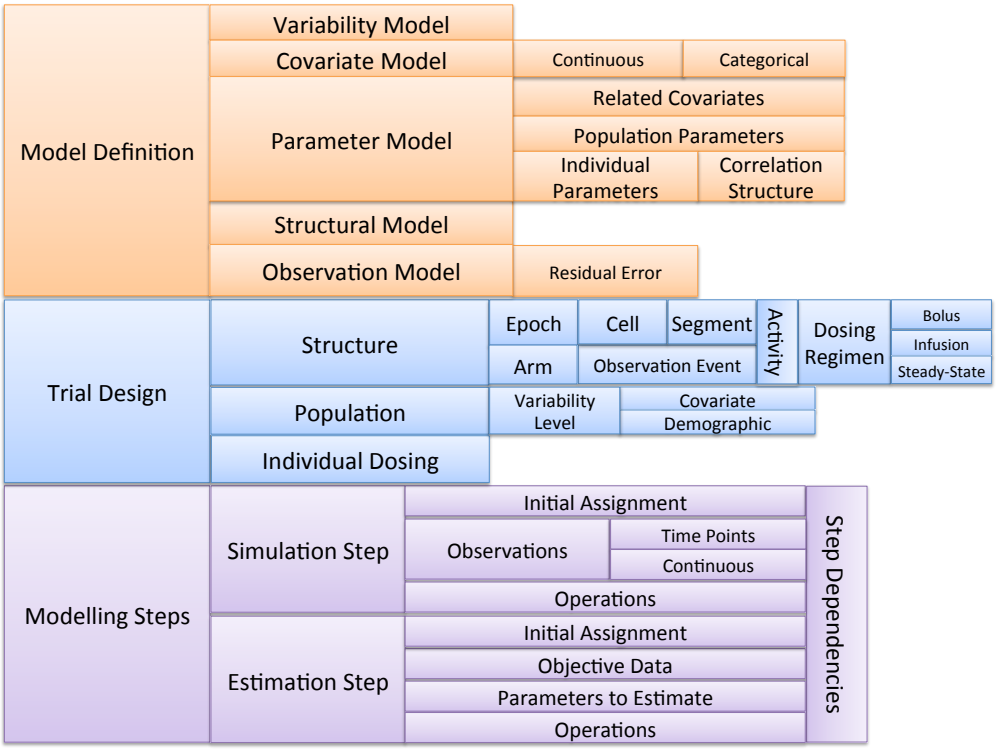


Figure 6.1: An overview of the organisation of PharmML.

Covariate Model

The *Covariate Model* as you would expect from the name describes the covariates used in the *Model Definition*. A covariate can be continuous, in which case it is typically described by a continuous probability distribution, or categorical. The formal description of the covariate model can be found in section 4.6.5.

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 defined more formally in section 4.6, 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), which includes capturing the correlation between them — essentially defining a covariance (or correlation) matrix for each level of variability.

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.

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 4.7.1). 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 will also be defined in the Observation Model, but at the moment are not supported by the current version of PharmML (see chapter 3).

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 groups with each group subject to one or more treatment regimens and observation protocols. Each group 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 describe the structure and population of a trial explicitly in a dedicated section. This differs from some other approaches, bit we feel it makes the clinical trial much clearer to document and easier to encode computationally. More information can be found in chapter 5.

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 to identify components in the *Trial Design* and *Modelling Steps* sections of a PharmML document and the Symbol Identifier 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 SDM-XML Technical Committee, 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.

[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.

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. These compatibilities are documented for each element containing an object reference (i.e. an `oidRef` attribute) in the XML Schema (see chapter 11).

6.3.2 Blocks and symbol scoping

As any other model description language 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 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>
```

[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.

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. 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 `symbIDs` 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. Unlike many languages, such as C or Fortran, symbols can be referred to before they are defined. This may seem odd at first, bit since PharmML is a declarative language (unlike C and Fortran) 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/2013/03/Maths">
            <Binop op="divide">
                <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/2013/03/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.

```
<!-- 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. Why? 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"/>
    <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>
```

At first sight it may seem intuitive to allow variable "a" to be assigned repeatedly in this way, but remember that PharmML is a declarative language and the order is not important. When you remember this then the XML in this listing becomes ambiguous. Which 'initial' assignment should be assigned before the others? Clearly this makes no sense in a declarative language and consequently symbols in PharmML can only be initialised once.

Before we leave symbols and symbol referencing it is worth noting 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 12.3.

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>
```

You will notice 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 Defining derivative variables

The easiest way to understand how one defines a derivative in PharmML is to look at an example such as the listing below:

```
<ct:DerivativeVariable symbId="Ad" symbolType="real">
    <ct:Assign>
        <Equation xmlns="http://www.pharmml.org/2013/03/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:Assign>
            <ct:Real>0</ct:Real>
        </ct:Assign>
    </ct:InitialCondition>
</ct:DerivativeVariable>
```

this corresponds to the equation:

$$\frac{\mathrm{d}Ad}{\mathrm{d}t} = -ka\ Ad$$

$$Ad(t = 0) = 0$$

As you can see the 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 zero in the <InitialCondition> element. 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. This is allowed.

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.

3. The initial condition is applied at $t0$ for the model, which in PharmML is assumed to be $t0 = 0$.

6.6 Datasets

The dataset is a key concept in PharmML and is used to describe the data that describes the trial design and the observations used for estimation. Much of this data is tabular and the dataset has therefore been designed to represent this type of information. The dataset describes data using XML and all data is explicitly typed.

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" valueType="string" columnNum="1"/>
        <ds:Column columnId="arm" valueType="string" columnNum="2"/>
        <ds:Column columnId="reps" 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>
```

As before the dataset has a definition, where the columns of the dataset table are defined. The column number must start at 1 and each column must be numbered in consecutive order (i.e. 1,2,3,4...etc.). The type of each column is specified and this complies with the PharmML type system. 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. NULL values are indicated by the <Null/> element.

This looks like a table in a relational database and indeed this approach is based on the concept of a relation in relational theory. Therefore the ordering of rows is not significant. At present there is no mechanism to define a key on the dataset or columns that cannot be NULL. It is assumed that such restrictions may be applied when the dataset is used. For example it is assumed that when mapping observations in the <EstimationStep> none of the data is NULL and that the combination of the time column and that identifying the individual are unique.

Finally, there is one deviation from standard relational theory in the dataset. This is that a table can be nested multiple times, i.e., a table can contain a column that defines another table and so on. This

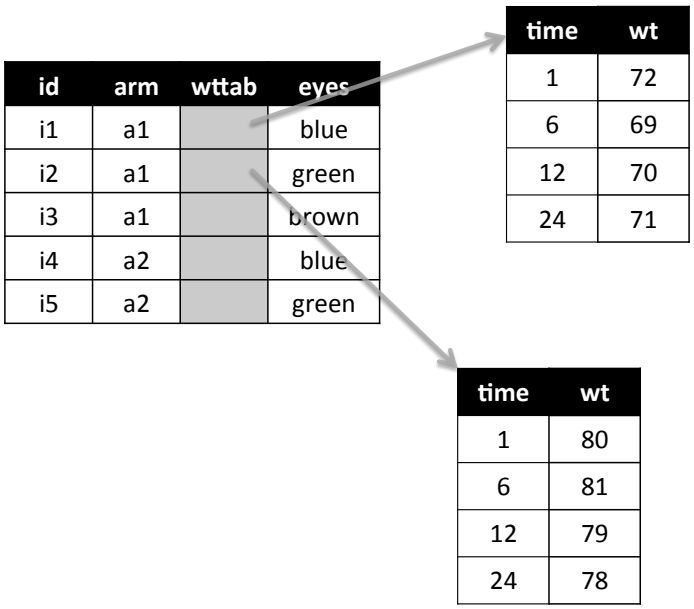


Figure 6.2: An illustration of how a table with a nested table can be conceptualised. Each cell within the `wttab` column contains another table.

concept is illustrated in figure 6.2. We required this because in some situations it is necessary to define one to many relationships within the data. For example when defining a population of individuals in a study whose weight changes during the course of the study (weight is a time-dependent covariate). In relational theory this is achieved by a foreign key relationship, but in the dataset it is simpler and clearer if we take advantage of the hierarchical nature of the XML. You can see this in the example dataset below:

```
<ds:DataSet>
    <ds:Definition>
        <ds:Column columnId="id" valueType="string" columnNum="1"/>
        <ds:Column columnId="arm" valueType="string" columnNum="2"/>
        <ds:Table tableId="wttab" columnNum="3">
            <ds:Definition>
                <ds:Column columnId="time" valueType="real" columnNum="1"/>
                <ds:Column columnId="wt" valueType="real" columnNum="2"/>
            <ds:Definition>
        </ds:Table>
        <ds:Column columnId="eyes" valueType="string" columnNum="4"/>
    </ds:Definition>
    <ds:Table>
        <ds:Row>
            <ct:String>i1</ct:String><ct:String>a1</ct:String>
            <ds:Table>
                <ds:Row>
                    <ct:String>1</ct:String><ct:Real>72</ct:Real>
                </ds:Row>
                <ds:Row>
                    <ct:String>6</ct:String><ct:Real>69</ct:Real>
                </ds:Row>
                <ds:Row>
                    <ct:String>12</ct:String><ct:Real>70</ct:Real>
                </ds:Row>
                <ds:Row>
                    <ct:String>24</ct:String><ct:Real>71</ct:Real>
                </ds:Row>
            </ds:Table>
            <ct:String>blue</ct:String>
        </ds:Row>
```

```
        <ds:Row>
            <ct:String>i2</ct:String><ct:String>a1</ct:String>
            <ds:Table>
                <ds:Row>
                    <ct:String>1</ct:String><ct:Real>80</ct:Real>
                </ds:Row>
                <ds:Row>
                    <ct:String>6</ct:String><ct:Real>81</ct:Real>
                </ds:Row>
                <ds:Row>
                    <ct:String>12</ct:String><ct:Real>79</ct:Real>
                </ds:Row>
                <ds:Row>
                    <ct:String>24</ct:String><ct:Real>78</ct:Real>
                </ds:Row>
            </ds:Table>
            <ct:String>green</ct:String>
        </ds:Row>
    </ds:Table>
</ds:DataSet>
```

In the example the child table is defined by using a `<Table>` element instead of the usual `<Column>` element and given the identifier "wttab". Within the nested table definition another set of columns is specified[3]. Now when encoding the data within the dataset its rows are defined as before, but where a column has been replaced by a nested table the contents of this table are delimited by a `<Table>` element. The dataset in the example defines individual `i1` assigned to arm `a1` with a weight that varies between 69 kg and 72 kg during the study.

6.7 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[4], 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[5]. This was also the conclusion reached for similar standards to PharmML such as SBML [Hucka et al., 2010, Section 3.4], CellML[6] 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 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.

[3] Of course this definition can itself contain another table definition *ad infinitum*.

[4] `http://www.w3.org/TR/MathML3/`

[5] We should emphasise that his is not a criticism of MathML, as this was the problem it was created to solve!

[6] `http://www.cellml.org/specifications/cellml_1.1/#sec_mathematics`

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[7]. 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[8], 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[9].

```
<!-- (9 - 5) * 2 -->
<Equation xmlns="http://www.pharmml.org/2013/03/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>
```

[7] 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.

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

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

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:

```
<?xml version="1.0" encoding="UTF-8"?>
<Equation xmlns="http://www.ddmore.eu/Resources/Maths/1.0"/>
    <!-- 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/Resources/Maths/1.0"/>
    <!-- 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 12.7. Here you will also find a description of each operator's semantics and the permitted types of its operands.

6.8 Representing statistics

As can be seen in chapter 4, 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[10], 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>
```

6.9 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/2013/03/PharmML"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.pharmml.org/2013/03/PharmML ..."
    xmlns:math="http://www.pharmml.org/2013/03/Maths"
    xmlns:ct="http://www.pharmml.org/2013/03/CommonTypes"
    xmlns:ds="http://www.pharmml.org/2013/08/Dataset"
    xmlns:design="http://www.pharmml.org/2013/03/TrialDesign"
    writtenVersion="0.1">
    <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.

6.10 Element identifier

We know that other resources will wish to interact with PharmML and so we have given some thought to how best they should refer to specific pieces of information in a PharmML document. Our solution is to provide every XML element in the document with an optional unique identifier. The listing below shows how the `id` attribute is used:

[10] `http://www.uncertml.org`

```
<FixedEffect id="e10" symbId="beta_V">
    <Covariate>
        <ct:SymbRef  symbIdRef="W"/>
    </Covariate>
</FixedEffect>
<FixedEffect id="e13" symbIdRef="beta_Cl">
    <Covariate>
        <ct:SymbRef symbIdRef="W"/>
    </Covariate>
</FixedEffect>
<RandomVariable id="e16" symbId="eta_V">
    <ct:VariabilityReference id="e17">
        <ct:SymbRef id="e18"  blkIdRef="model" symbIdRef="level"/>
   </ct:VariabilityReference>
   <!-- Snip -->
</RandomVariable>
```

As you can see 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 6.12.1 and 6.12.3) how we take advantage of this mechanism to annotate and extend a PharmML document.

6.11 Ordering modelling steps

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

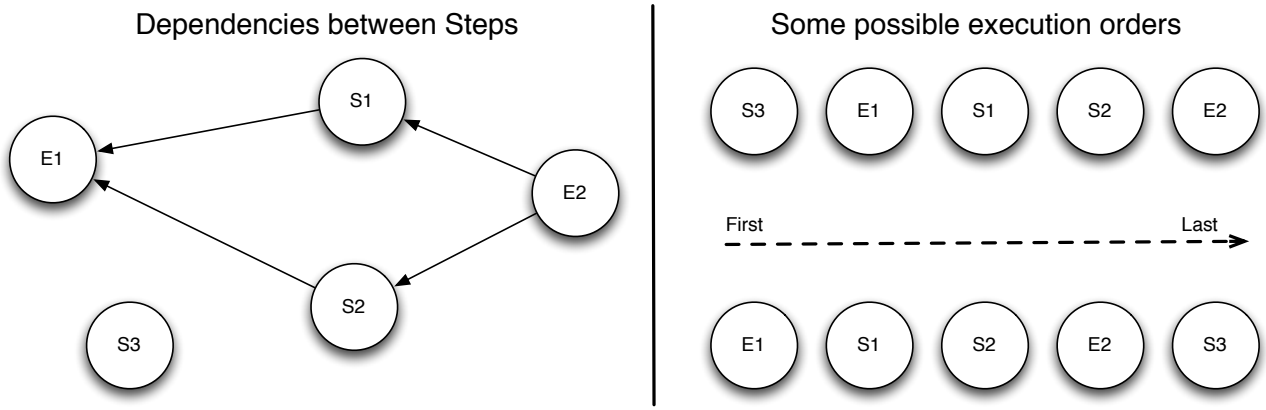


Figure 6.3: An example of the dependencies between modelling steps is shown on the left. The arrow indicates the direction of the dependency so step S1 is dependent on the successful completion of step E1. So in this example step E1 must be executed before step S1 and S2, and those steps must both execute 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 6.3). 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.

6.12 Supporting Resources

6.12.1 Metadata: annotating the PharmML document

As has been stated above (Chapter 3), 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 still very important and so PharmML aims to provide support for external annotation.

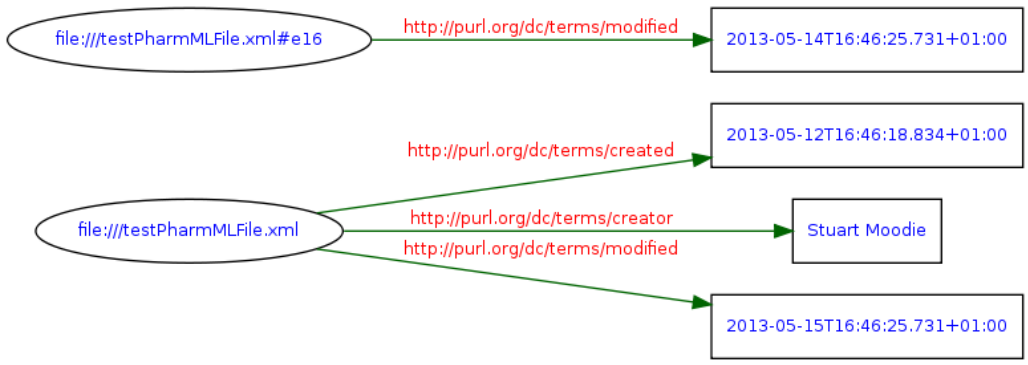


Figure 6.4: This figure illustrates how information in the RDF example is organised. Simply put the subject of the annotation is on the left and is annotated by the object on the right. The meaning of that annotation is described by the *predicate* term labelling the arrow. So reading the top subject from left to right we understand that element *e16* was *modified* at *14/05/13 at 4:46pm*.

We expect such metadata to take the form of ontological annotation and while the detail of what will be annotated is out of scope of this document we can show how this can work using a standard for describing resources (such as XML documents) called Dublin Core[11]. Using this ontology we can describe many things about a PharmML document such as when and by whom it was created or last updated. Typically such annotation is expressed using a standard called RDF[12], which can be updated and queried using associated software libraries. The following example[13] gives you a flavour of what an RDF annotation of a PharmML document would look like if we were referring to the code snippet in section 6.10.

[11] For more information see `dublincore.org`.

[12] See RDF URL

[13] This example is available as part of the examples provided with this specification at `examples/-CombineArchive/annotation.xml`.

```
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:dcterms="http://purl.org/dc/terms/">
  <rdf:Description rdf:about="testPharmMLFile.xml">
    <dcterms:created>2013-05-12T16:46:18.834+01:00</dcterms:created>
    <dcterms:creator>Stuart Moodie</dcterms:creator>
    <dcterms:modified>2013-05-15T16:46:25.731+01:00</dcterms:modified>
  </rdf:Description>
  <rdf:Description rdf:about="testPharmMLFile.xml#e16">
    <dcterms:modified>2013-05-14T16:46:25.731+01:00</dcterms:modified>
  </rdf:Description>
</rdf:RDF>
```

The metadata above is encoded in RDF-XML[14] and annotates two things. The first `<rdf:Description>` element states when the file was both created and modified and also who created it. The second `<rdf:Description>` element points to the XML element in the PharmML document with an `id` value of "e16". In the example in section 6.10 this is the `<RandomVariable>` element and so we can deduce that this description is telling us when this element was modified. Another way of looking at RDF information is shown in figure 6.4, which hopefully makes the relationships described above even clearer. Of course this is a trivial example, but it illustrates how the PharmML document can be annotated using metadata described in a separate RDF document.

6.12.2 Standard Structural Models

In many, if not the majority, of pharmacometric models the structural model is selected from a standard library. Modelling tools such as NONMEM or Monolix provide large platform specific libraries [Beal et al., 2006] [Bertrand and Mentré, 2008]. These have the benefit of being both reusable and optimised to run efficiently on their target platform. With PharmML we want to support both of these features and so we will be establishing a framework that provides the following:

a model ontology an ontology describing all the models held in the standard library.

tool specific mappings using the model ontology this will provide mapping from the models in the standard library to tool specific models.

We expect that the structural model within the PharmML document will be annotated with the correct structural model using the model ontology (following the annotation mechanism described above in section 6.12.1). The model ontology term can be used to identify the appropriate model name for a given modelling tool by using one of the tool-specific mappings. This allows PharmML to provide a tool-independent description of standard structural models. It also allows conversion tools to convert a PharmML model into a tool specific encoding that takes advantage of its built-in structural model library. This process is summarised in figure 6.5.

6.12.3 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.

[14] See `rdf-xml`

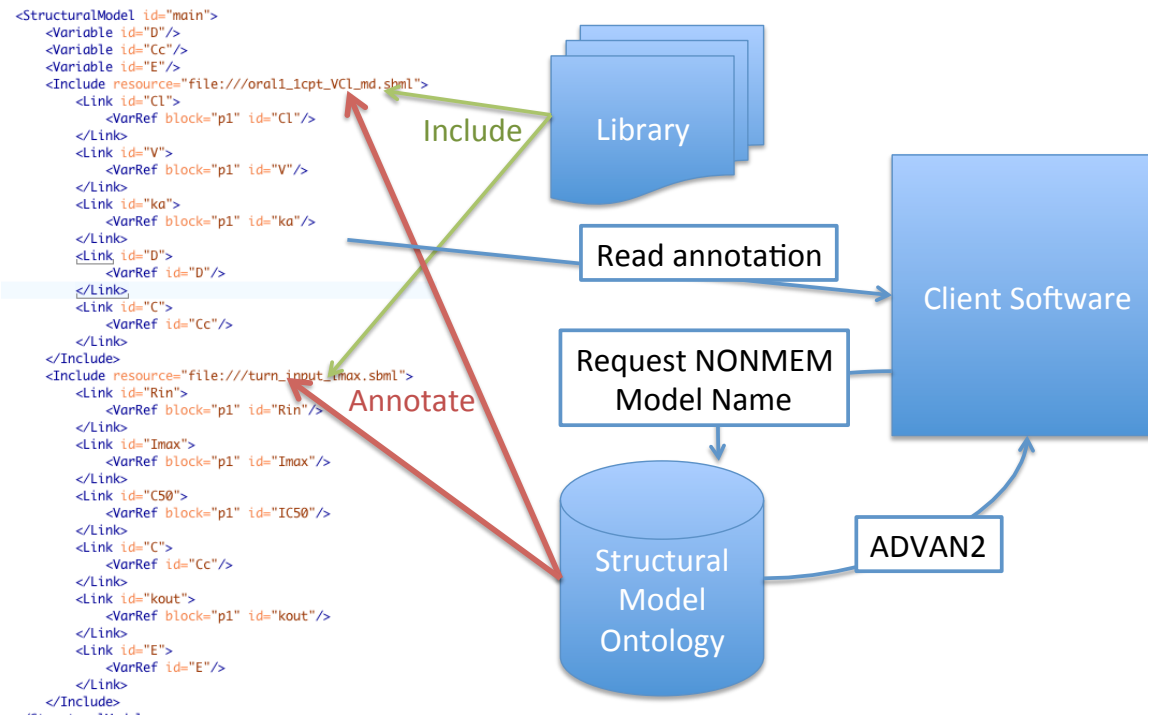


Figure 6.5: A schematic diagram illustrating how the standard library ontology framework is used to generate target specific code when translating PharmML for a specific M&S tool.

Whatever the reason PharmML can be extended. Like the metadata descriptions above (section 6.12.1) this approach relies on the element identifier (see section 6.10). 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.

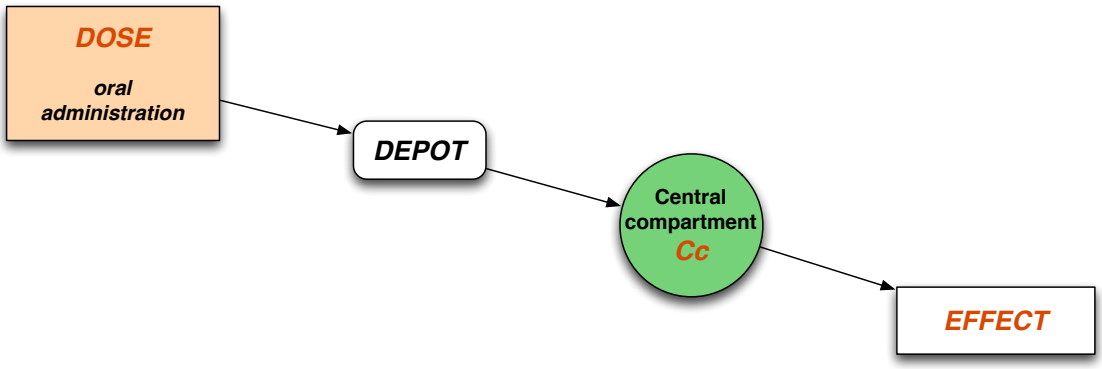


Figure 6.6: This diagram is reproduced from an example in the Monolix user manual [Lixoft, 2012] and it provides a graphical description of a structural model. In the hypothetical example in the text we illustrate how you might extend PharmML to link this graphic with the components of the model it represents.

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 6.6.

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:

```
<?xml version="1.0" encoding="UTF-8"?>
<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[15]. 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.

6.12.4 Organising PharmML resources

We expect that PharmML will in normal usage consist of more than one file. From the discussion about annotating (section 6.12.1) and extending (section 6.12.3) 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.

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 standard[16]. 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

[15] `www.celldesigner.org`
[16] `http://co.mbine.org/documents/archive`

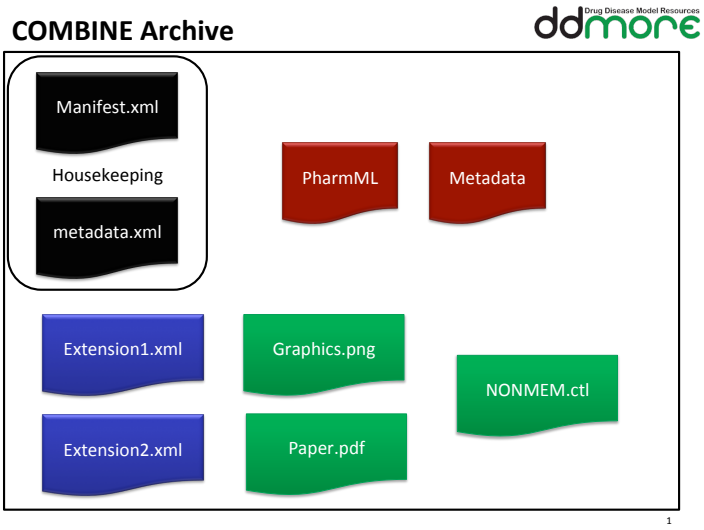


Figure 6.7: An overview of how a COMBINE archive may be used to hold XML documents and files associated with a model encoded in PharmML. In this example the archive holds the model and its metadata (in red) and two application specific extension documents (in blue). This also shows another advantage 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.

this document, but the components of the archive and how it can be used to hold PharmML related resources is illustrated in figure 6.7[17].

6.12.5 Software support for PharmML

PharmML is a complex language. It is designed using an industry standard, XML Schema[18], 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[19] 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).

[17] An example of a CombineArchive file that contains a metadata file annotating a PharmML document can be found in `examples/CombineArchive/archive.zip`.

[18] `http://www.w3.org/XML/Schema`

[19] For more information see the libPharmML specification in the DDMoRe Interface Europe document repository: WP2 deliverables: *D2.2 libPharmMLTechSpec*.

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.

CHAPTER 7

Worked Examples

7.1 How this chapter is organised

In developing PharmML we have found it very useful to explain the language using examples. By taking you through some pharmacometric models that you are familiar with we also hope to help you understand how they correspond to the XML representation in PharmML. Each example is designed to illustrate different aspects of PharmML and our aim is that by the end of this chapter you will understand the language 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.

Figure 7.1 shows an comparison in the structure to be implemented for a typical simulation and estimation task. Boxes underline elements where the structure of PharmML for these two tasks differs.

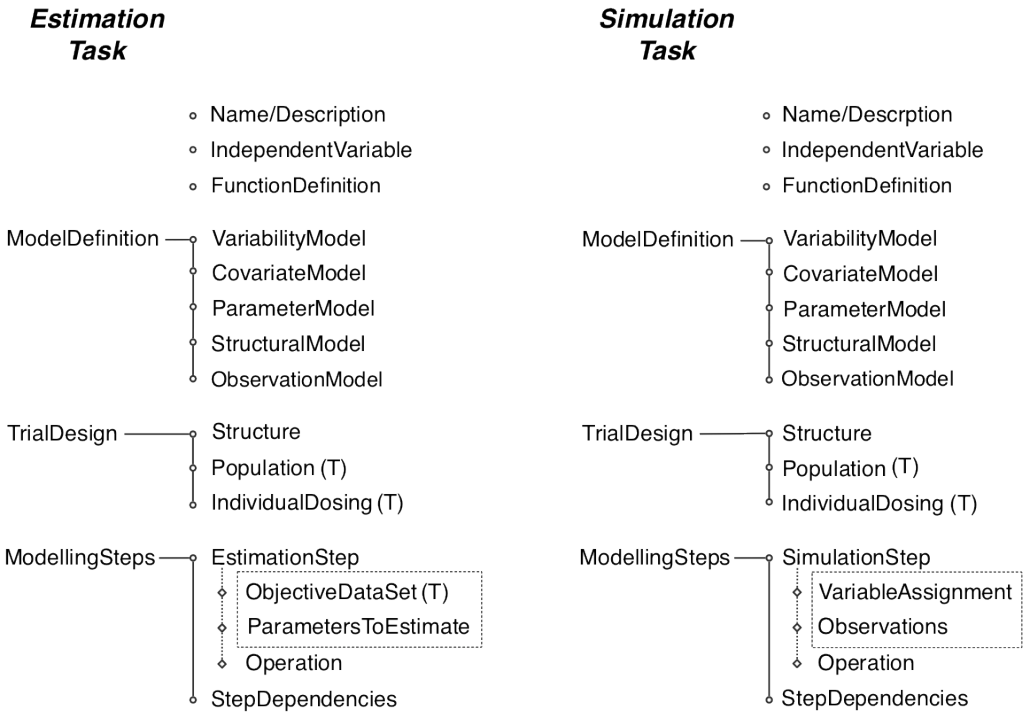


Figure 7.1: PharmML building blocks used in the trial definition for an estimation and simulation task. Boxes underline elements where the structure of PharmML for these two tasks differs. (T) indicates that tabular data structure is used.

7.2 Example 1: Simulation, PK + PD response

7.2.1 Description

The following example 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 7.2 for a typical simulation result for one patient.

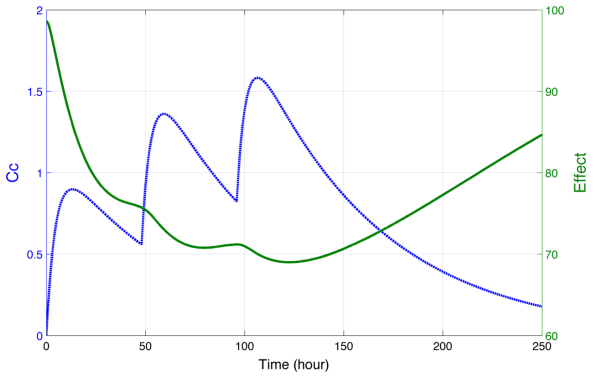


Figure 7.2: Simulated combined model as defined in the example with PK (blue) and PD (green) time courses for one subject. Here three doses were administered every $48h$.

Model Definition

Structural model This is an oral one compartment model and an indirect response model with parameters ka, V, CL, $Imax$, $IC50$, Rin and $kout$.

$$
\begin{aligned}
k &= \frac{CL}{V} \\
\frac{dAd}{dt} &= -ka \times Ad \\
\frac{dAc}{dt} &= ka \times Ad - k \times Ac \\
\frac{dE}{dt} &= Rin \times \left(1 - \frac{Imax \times Cc}{Cc + IC50}\right) - kout \times E \\
Cc &= \frac{Ac}{V}
\end{aligned}
\tag{7.1}
$$

initial conditions:

$$
\begin{aligned}
E(t=0) &= \frac{Rin}{kout} \\
Ad(t=0) &= 0 \\
Ac(t=0) &= 0
\end{aligned}
\tag{7.2}
$$

Covariate model The only covariate is Weight, W, and it is a continuous covariate:

$$W \sim \mathcal{N}(pop_W, \omega_W) \tag{7.3}$$

The following transformation is applied:

$$\log(W/70) \tag{7.4}$$

and the initial values are:

$$pop_W = 70.07, \quad \omega_W = 14.09$$

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:

$$\log(ka_i) = \log(pop_{ka}) + \eta_{ka,i} \tag{7.5}$$
$$\log(V_i) = \log(pop_V) + \beta_{1,V}\log(W_i/70) + \eta_{V,i} \tag{7.6}$$
$$\log(CL_i) = \log(pop_{CL}) + \beta_{1,CL}\log(W_i/70) + \eta_{CL,i}$$

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:

$$pop_{ka} = 1, \quad \omega_{ka} = 0.6 \qquad pop_V = 8, \quad \omega_V = 0.2$$
$$pop_{CL} = 0.13, \quad \omega_{CL} = 0.2 \qquad \beta_{1,V} = 1, \quad \beta_{1,CL} = 0.75$$
$$\rho_{V,CL} = 0.7^{1}$$

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.

$$logit(Imax_i) = logit(pop_{Imax}) + \eta_{Imax,i}$$
$$\log(IC50_i) = \log(pop_{IC50}) + \eta_{IC50,i}$$
$$\log(Rin_i) = \log(pop_{Rin}) + \eta_{Rin,i}$$
$$\log(kout_i) = \log(pop_{kout}) + \eta_{kout,i}$$

[1] Correration coefficient between $\eta_{V,i}$ and $\eta_{CL,i}$

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:

$$\Omega = \begin{pmatrix} \omega_{ka}^2 & 0 & 0 & 0 & 0 & 0 & 0 \\ & \omega_V^2 & \omega_{V,CL} & 0 & 0 & 0 & 0 \\ & & \omega_{CL}^2 & 0 & 0 & 0 & 0 \\ & & & \omega_{Imax}^2 & 0 & 0 & 0 \\ & & & & \omega_{IC50}^2 & 0 & 0 \\ & & & & & \omega_{Rin}^2 & 0 \\ & & & & & & \omega_{kout}^2 \end{pmatrix} \tag{7.7}$$

where

$$\omega_{V,CL} = \omega_V \, \omega_{CL} \, \rho_{V,CL}$$

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	$a = 0.5, \quad b = 0.1$	$a = 4$

Trial Design

Figure 7.3 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

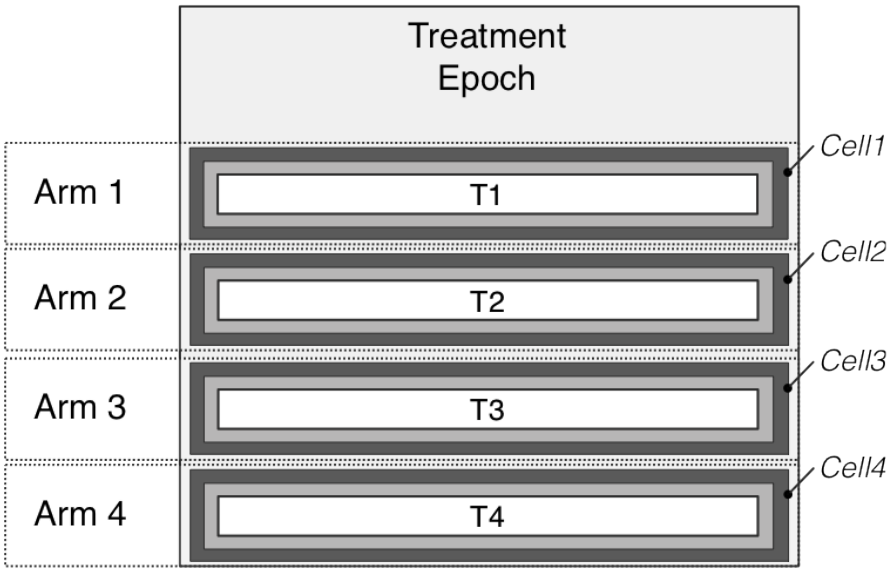


Figure 7.3: Design overview: this study consists of four arms and one epoch. The differences between arms lie in the number of subjects, dose amount and times. See table below for the details.

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

7.2.2 PharmML Document Structure

An overview of the model with the key sections collapsed as shown in this listing

```
<?xml version="1.0" encoding="UTF-8"?>
<PharmML xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns="http://www.pharmml.org/2013/03/PharmML"
    xsi:schemaLocation="http://www.pharmml.org/2013/03/PharmML
    http://www.pharmml.org/2013/03/PharmML"
    xmlns:math="http://www.pharmml.org/2013/03/Maths"
    xmlns:ct="http://www.pharmml.org/2013/03/CommonTypes"
    xmlns:ds="http://www.pharmml.org/2013/08/Dataset"
    xmlns:mdef="http://www.pharmml.org/2013/03/ModelDefinition"
    xmlns:mstep="http://www.pharmml.org/2013/03/ModellingSteps"
    xmlns:mml="http://www.pharmml.org/2013/03/PharmML"
    writtenVersion="0.1">
    <ct:Name>Example 1 - continuous PK/PD</ct:Name>
    <IndependentVariable symbId="t"/>
    <FunctionDefinition xmlns="http://www.pharmml.org/2013/03/CommonTypes"
        symbId="constantErrorModel" symbolType="real">
        <!-- omitted details -->
    </FunctionDefinition>
    <FunctionDefinition xmlns="http://www.pharmml.org/2013/03/CommonTypes"
        symbId="combinedErrorModel" symbolType="real">
        <!-- omitted details -->
    </FunctionDefinition>
    <ModelDefinition xmlns="http://www.pharmml.org/2013/03/ModelDefinition">
        <!-- omitted details -->
    </ModelDefinition>
```

```
    <TrialDesign xmlns="http://www.pharmml.org/2013/03/TrialDesign">
        <!-- omitted details -->
    </TrialDesign>
    <ModellingSteps xmlns="http://www.pharmml.org/2013/03/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` attribute to set the time variable to t (c.f. section 6.9), 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`. These are required by the XML Schema standard and can be ignored as we go through the examples[2].

The above listing introduces a concept and XML element that we did not describe before: this is the `<FunctionDefinition>` element. It defines a function that returns a real type as shown in the following listing

```
<FunctionDefinition xmlns="http://www.pharmml.org/2013/03/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/2013/03/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 7.2.3).

7.2.3 Model Definition

As you can see in the following listing

```
<ModelDefinition xmlns="http://www.pharmml.org/2013/03/ModelDefinition">
    <VariabilityModel blkId="model" type="model">
        <!-- omitted details -->
    </VariabilityModel>
    <VariabilityModel blkId="obsErr" type="error">
        <!-- omitted details -->
    </VariabilityModel>
    <CovariateModel blkId="c1">
        <!-- omitted details -->
    </CovariateModel>
    <ParameterModel blkId="p1">
        <!-- omitted details -->
    </ParameterModel>
```

[2] If you want to learn more about the technical aspects of XML and the XML Schema standard used to define PharmML then we recommend to start here: `http://www.w3.org/TR/xmlschema-0/`. And, by the way, `xmlns` stands for XML namespace.

```
    <StructuralModel blkId="main">
        <!-- omitted details -->
    </StructuralModel>
    <ObservationModel blkId="o1">
        <!-- omitted details -->
    </ObservationModel>
    <ObservationModel blkId="o2">
        <!-- omitted details -->
    </ObservationModel>
</ModelDefinition>
```

the model definition section defines the main components that you will see in the subsequent examples. Apart from the structural model, all of these elements are optional and all follow the scoping rules described in section 6.3.2.

Variability Model

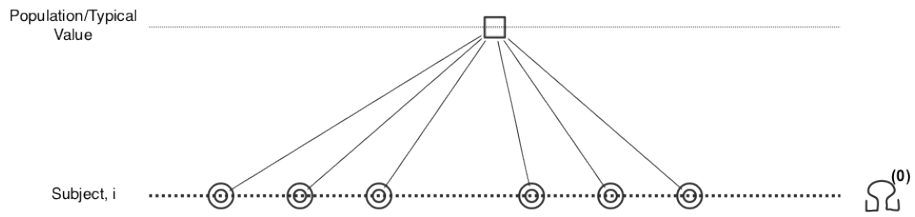


Figure 7.4: There is only one level of variability in this example – inter-individual variability.

From the listing above 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 4.5). In this case there is one level of variability, the inter-individual, Figure 7.4 and the residual error model variability, see the following listing

```
<VariabilityModel blkId="model" type="model">
    <Level symbId="indiv">
        <ct:Name>Individual Variability</ct:Name>
    </Level>
</VariabilityModel>
<VariabilityModel blkId="obsErr" type="error">
    <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, `model` and `error`. The former stands for random variability associated with the parameters, as described in section 4.5. 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.

We will explain this more fully below when we look at examples with more complex variability models (section 7.5.3).

Covariate Model

The `<CovariateModel>` block corresponds to the covariate model defined in section 4.6.5. In this example, as shown in this listing

```
<CovariateModel blkId="c1">
    <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>
                <Equation xmlns="http://www.pharmml.org/2013/03/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 7.3. The element `<Transformation>` beneath the definition of the distribution describes the transformation applied to this covariate whenever it is used. In this case the transformation being applied is defined in equation 7.4.

Parameter Model

All parameters in the current example 1 can be defined using the Gaussian model with linear covariates (see section 4.6). We will start with a simple case in this example, shown in the following listing

```
<ParameterModel blkId="p1">
    <!-- other parameters ... -->
    <!-- ka -->
    <SimpleParameter symbId="pop_ka"/>
    <SimpleParameter symbId="omega_ka"/>
    <RandomVariable symbId="eta_ka">
        <ct:VariabilityReference>
            <ct:SymbRef blkIdRef="model" 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, ω_ka, are defined as <SimpleParameter>'s. Then the random effect eta_ka is defined, and it follows a normal distribution with mean 0 and standard deviation ω_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 4.6), and it follows a log-normal distribution with the mean pop_ka and the standard deviation $omega\_ka$, as shown in equation 7.5. 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">
        <!-- identical to previous example -->
    </RandomVariable>
    <IndividualParameter symbId="V">
        <GaussianModel>
            <Transformation>log</Transformation>
            <LinearCovariate>
                <PopulationParameter>
                    <ct:Assign>
                        <ct:SymbRef symbIdRef="pop_V"/>
                    </ct:Assign>
                </PopulationParameter>
                <Covariate>
                    <ct:SymbRef blkIdRef="c1" symbIdRef="W"/>
                    <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 4.6.5), so this example corresponds to equation 7.6.

You may have noticed that in equation 7.6 the covariate term is $\log(W/70)$ and not W. Why? Well we have applied the covariate transformation defined in (7.4). As we described above (section 7.2.3), whenever a covariate is used here we always apply any transformation defined in the <Transformation> element.

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 4.6.4 for a complete description) we do this using a covariance matrix. In PharmML, if the random effects of both param-

eters 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 4.6.4). This is true in our example, so we only need to define the parts of the covariance matrix that define the correlation between parameters V and CL as shown in this listing

```
<Correlation>
    <ct:VariabilityReference>
        <ct:SymbRef blkIdRef="model" symbIdRef="indiv"/>
    </ct:VariabilityReference>
    <RandomVariable1>
        <ct:SymbRef symbIdRef="eta_V"/>
    </RandomVariable1>
    <RandomVariable2>
        <ct:SymbRef symbIdRef="eta_Cl"/>
    </RandomVariable2>
    <CorrelationCoefficient>
        <ct:SymbRef symbIdRef="rho_V_Cl"/>
    </CorrelationCoefficient>
</Correlation>
```

Notice that with the `symbIdRef` attribute the correlation is associated with the variability level defined at the beginning of the model definition. This allows us to define a covariance matrix for each level of variability in the model and thus to define different parameter correlations at each level as well. This single `<Correlation>` element is all we need to define the covariance matrix mentioned above (section 7.2.1).

You now know most of the structures used to define a parameter in PharmML. As you will see later, these elements can be combined to create more complicated parameter models, but these are elaborations of this framework.

Structural Model

In PharmML the structural model can be described using algebraic equations or an ODE system. In this example the model is defined as a combination of an ODE system and some supporting algebraic equations (c.f. equation 7.1).

The following listing

```
<StructuralModel blkId="main">
    <ct:Variable symbId="k" symbolType="real">
        <ct:Assign>
            <Equation xmlns="http://www.pharmml.org/2013/03/Maths">
                <Binop op="divide">
                    <ct:SymbRef blkIdRef="p1" symbIdRef="Cl"/>
                    <ct:SymbRef blkIdRef="p1" symbIdRef="V"/>
                </Binop>
            </Equation>
        </ct:Assign>
    </ct:Variable>
    <ct:DerivativeVariable symbId="Ad" symbolType="real">
        <ct:Assign>
            <Equation xmlns="http://www.pharmml.org/2013/03/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:Assign>
                <ct:Real>0</ct:Real>
```

```
            </ct:Assign>
        </ct:InitialCondition>
    </ct:DerivativeVariable>
```

illustrates how the variable k is defined. We cover this in more detail in chapter 6, but briefly: k is defined as having a real type and is assigned the result of the expression CL/V, which are parameters defined in the parameter model, $p1$. We can also see how the derivative Ad is defined using the element `<DerivativeVariable>` (see section 6.5 for a more detailed explanation) with t as the independent variable. The full ODE defined is:

$$\frac{\mathrm{d}\,Ad}{\mathrm{d}t} = -ka\,Ad$$

where ka is a parameter defined in the parameter model $p1$. Finally the `<InitialCondition>` element defines the initial value for Ad at time 0, here $Ad(t=0)=0$, which is the default initial time in PharmML.

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="o2">
    <SimpleParameter symbId="a"/>
    <SimpleParameter symbId="b"/>
    <RandomVariable symbId="epsilon_Cc">
        <ct:VariabilityReference>
            <ct:SymbRef blkIdRef="obsErr" symbIdRef="residual"/>
        </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="sigma_Cc"/></stddev>
        </NormalDistribution>
    </RandomVariable>
    <Standard symbId="Cc">
        <Output>
            <ct:SymbRef blkIdRef="main" 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="main" symbIdRef="Cc"/>
                            </math:Equation>
                        </math:FunctionArgument>
                    </math:FunctionCall>
                </math:Equation>
            </ct:Assign>
        </ErrorModel>
        <ResidualError>
            <ct:SymbRef symbIdRef="epsilon_Cc"/>
        </ResidualError>
    </Standard>
</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 4.7).

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 7.2.2). We pass in the appropriate parameter values (see section 4.7.1 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 4.7.3. 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.

7.2.4 Trial Design

Structure

In this fairly simple case, there is only one epoch and four arms, Arm_1, Arm_2 etc., cf. Figure 7.3. The resulting four cells $Cell\_1$, $Cell\_2$ etc. each correspond to one arm and contain one segment. Figure 7.5 (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

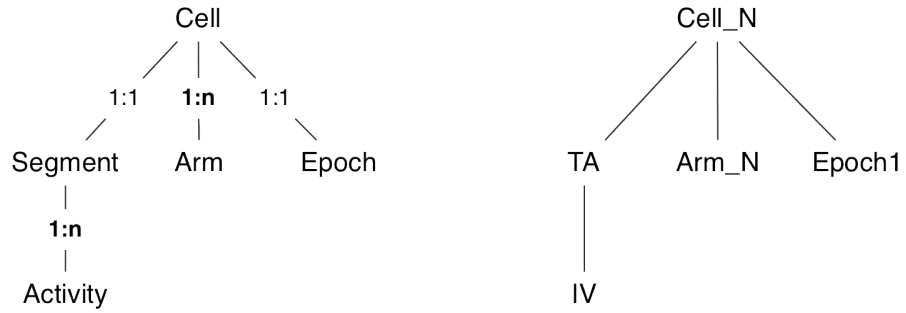


Figure 7.5: (left) General cell hierarchy; (right) How it is applied in example 1. There is only one epoch and four arms, Arm_1, Arm_2 etc. The resulting four cells $Cell\_1$, $Cell\_2$, etc. each correspond to one arm and contain one segment. See the listing bellow for how these features are implemented and structured.

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.

In the following listing

```
<Activity oid="d1">
    <Bolus>
        <DoseAmount inputType="target">
            <ct:SymbRef blkIdRef="main" symbIdRef="Ad"/>
            <ct:Assign>
                <Equation xmlns="http://www.pharmml.org/2013/03/Maths">
                    <Binop op="times">
                        <ct:Real>0.25</ct:Real>
                        <ct:SymbRef blkIdRef="c1" 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[3]. `<DoseAmount>` defines the amount of drug to be administered. The are basically two options here. Either the dose is assigned to the variable defined by an ODE, such is the case here, for which we use `inputType="target"` attribute or the dosing variable D as in algebraic equations. 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 $0 : 24 : 192$.

[3]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.

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> element is used to put all of the components together, see this listing

```
<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>
```

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 5 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.7.3. 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 7.4) 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="model" symbIdRef="indiv"/>
    </ct:VariabilityReference>
    <IndividualTemplate>
        <IndividualMapping>
            <ds:ColumnRef columnIdRef="id"/>
        </IndividualMapping>
        <ArmMapping>
            <ds:ColumnRef columnIdRef="arm"/>
        </ArmMapping>
        <ReplicateMapping>
            <ds:ColumnRef columnIdRef="reps"/>
        </ReplicateMapping>
    </IndividualTemplate>
    <ds:DataSet>
        <ds:Definition>
            <ds:Column columnId="id" valueType="string" columnNum="1"/>
            <ds:Column columnId="arm" valueType="string" columnNum="2"/>
            <ds:Column columnId="reps" 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>
            <!-- arms 2 & 3 omitted -->
            <ds:Row>
                <ct:String>i4</ct:String><ct:String>a4</ct:String><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*.
In the estimation case another block `<IndividualDosing>` would have to be defined as next. This will be explained in the following example 7.4.

7.2.5 Modelling Steps

Simulation settings and dependencies

In the following listing

```
<ModellingSteps xmlns="http://www.pharmml.org/2013/03/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.

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>This is the: c1.omega_W = 1409/100</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 12.3 on page 234).

Observations

Typically, what drives a simulation task are its outputs. You only need to simulate the parts of your system that produce the required outputs and for as long as you wish to observe those outputs. In PharmML we use the `<Observations>` element to do this job, as you can see in this listing

```
<Observations>
    <Timepoints>
        <ct:Vector>
            <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>
            <!-- SNIP -->
            <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:Vector>
    </Timepoints>
    <Continuous>
        <ct:SymbRef blkIdRef="main" symbIdRef="Cc"/>
        <ct:SymbRef blkIdRef="o1" symbIdRef="Cc"/>
    </Continuous>
</Observations>
```

In this example we define 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. You can define one or more output variables here using the `<Continuous>` element. It is noteworthy here that by choosing the Cc defined in both the structural model ("`main`") and observation model ("`o1`") blocks we can show the output of the structural model with and without the residual error applied.

7.3 Example 2: Simulation with steady state dosing

7.3.1 Description

The following example is taken from [Bonate, 2011], p.535, and represents another simple example for a PK simulation. 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 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.

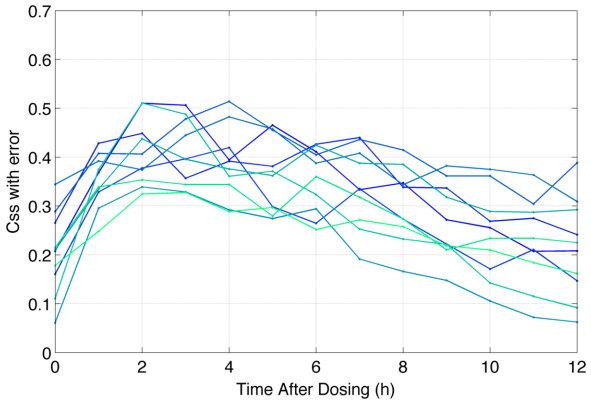


Figure 7.6: Simulated PK model as defined in the example for 10 subjects.

Variability model

The variability structure is identical to that in the previous example – there is only one level of subject related variability, see Figure 7.4.

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

$$pop_{\theta_1} = 25, \quad \omega_{\theta_1} = 5 \quad pop_V = 250, \quad \omega_V = 100.$$

Covariate model

Body weight is the only covariate used in this model. It is used in the model for the individual clearance only.

Structural model

$$
\begin{aligned}
k &= \frac{CL}{V} \\
C_{SS}(t) &= \frac{D}{V}\frac{K_a}{K_a - k}\left(\frac{e^{-k(t-t_D)}}{1-e^{-k\tau}} - \frac{e^{-K_a(t-t_D)}}{1-e^{-K_a\tau}}\right)
\end{aligned}
$$

	Weight
Type	Continues
Transformation	$(W/70)^{\theta_2}$
Distribution	Normal
Mean, pop_W	80
Standard deviation, ω_W	9.6

Table 7.1: Covariates overview.

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	$b = 0.1$

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

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

7.3.2 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/2013/03/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.

7.3.3 Trial design model

Structure

Figure 7.7 shows the *Structure* of this simple example consisting of 1 arm and one epoch, meaning one treatment type for everybody.

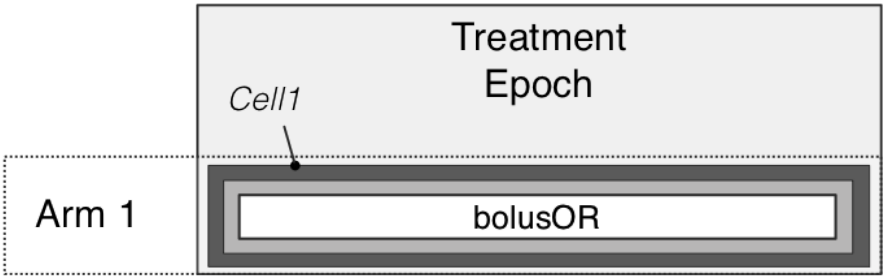


Figure 7.7: Design overview: this study consists of one arm and one epoch.

Segment	Activity	Treatment	DoseTime	DoseSize	Target Variable
TA	bolusOR	OR bolus	0	100	D

Table 7.2: Segment/activity overview.

Epoch	Start time	End time
Treatment Epoch	0	12

Table 7.3: 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 inputType="dose">
            <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>
```

7.4 Example 3: Estimation, Warfarin PK

7.4.1 Description

This model describes the PK of the drug Warfarin and corresponds to the DDMoRe WP3 use case Warfarin_PK_PRED[4].

Structural model

The model is a one compartment model with first-order absorption with lag time and first-order elimination.

D Dosing variable.

$\boldsymbol{t_D}$ Time of the dose.

C Concentration of drug in the compartment.

$$k = \frac{CL}{V}$$

$$C(t) = \begin{cases} 0 & \text{if} \quad t - t_D < T_{lag} \\ \frac{D}{V}\frac{k_a}{k_a - k}\left[e^{-k\left(t-t_D-T_{lag}\right)} - e^{-k_a\left(t-t_D-T_{lag}\right)}\right] & \text{otherwise} \end{cases}$$

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 7.4: Covariates overview.

[4] Available via Interface Europe: https://cp1.interfaceurope.eu/LotusQuickr/ddmore/PageLibraryC125786900388659.nsf/h_Toc/92be13faec1b58390525670800167238/?OpenDocument#{type=0&unid=5801C48FC6C39BB141257B2A007D6F31}

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 :

$$\Omega = \begin{pmatrix} \omega^2_{Tlag} & 0 & 0 & 0\\ & \omega^2_{ka} & 0 & 0\\ & & \omega^2_V & 0\\ & & & \omega^2_{CL}\end{pmatrix}$$

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	$a = 0.1, \quad b = 0.1$

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

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

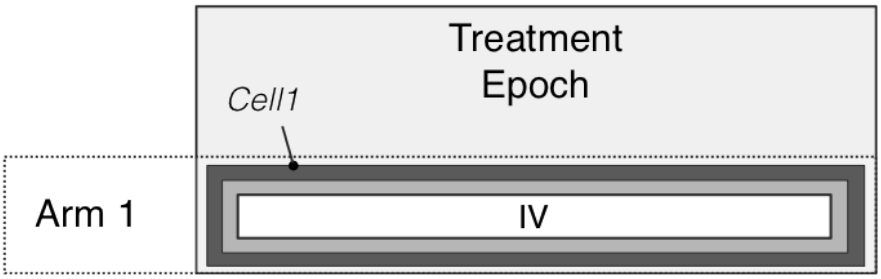


Figure 7.8: Design overview: this study consists of one arm and one epoch.

Figure 7.8 shows the *Structure* of this simple example consisting of 1 arm and one epoch, meaning one treatment type for everybody.

7.4.2 Overview

This our first estimation case. From figure 7.1 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 6.7). 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 7.6.2 in example 7.6 how this is done.

7.4.3 Trial Design

Structure

As explained in the chapter 5 on trial design, we base the following structure on the CDISC standard Study Design Model [CDISC SDM-XML Technical Committee, 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 inputType="dose">
                <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="dose". Additionally the dosing time variable tD is referenced here and initialised.

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>
    <IndividualTemplate>
        <IndividualMapping>
            <ds:ColumnRef columnIdRef="ID"/>
        </IndividualMapping>
        <ArmMapping>
            <ds:ColumnRef columnIdRef="ARM"/>
        </ArmMapping>
        <CovariateMapping>
            <ds:ColumnRef columnIdRef="WEIGHT"/>
            <ct:SymbRef blkIdRef="cm1" symbIdRef="W"/>
        </CovariateMapping>
    </IndividualTemplate>
    <ds:DataSet>
        <ds:Definition>
            <ds:Column columnId="ID" valueType="string" columnNum="1"/>
            <ds:Column columnId="ARM" valueType="string" columnNum="2"/>
            <ds:Column columnId="WEIGHT" valueType="real" columnNum="3"/>
        </ds:Definition>
        <ds:Table>
            <ds:Row><ct:String>i1</ct:String><ct:String>a1</ct:String>
                <ct:Real>70.1</ct:Real></ds:Row>
            <ds:Row><ct:String>i2</ct:String><ct:String>a1</ct:String>
                <ct:Real>60.0</ct:Real></ds:Row>
            <ds:Row><ct:String>i3</ct:String><ct:String>a1</ct:String>
                <ct:Real>93.2</ct:Real></ds:Row>
            <ds:Row><ct:String>i4</ct:String><ct:String>a1</ct:String>
                <ct:Real>85.7</ct:Real></ds:Row>
            <ds:Row><ct:String>i5</ct:String><ct:String>a1</ct:String>
                <ct:Real>78.3</ct:Real></ds:Row>
            <!-- SNIP -->
            <ds:Row><ct:String>i33</ct:String><ct:String>a1</ct:String>
                <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 $a1$.

7.4.4 Modelling Steps

Objective data

The biggest advantage of the current specification is that we do not have to define the design in the data file. After the structure of the trial is defined as above 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>
    <IndividualMapping>
        <ds:ColumnRef columnIdRef="ID"/>
    </IndividualMapping>
    <VariableMapping>
        <ds:ColumnRef columnIdRef="time"/>
        <ct:SymbRef symbIdRef="t"/>
```

```
    </VariableMapping>
    <VariableMapping>
        <ds:ColumnRef columnIdRef="dv"/>
        <ct:SymbRef blkIdRef="om1" symbIdRef="C"/>
    </VariableMapping>
    <ds:DataSet>
        <ds:Definition>
            <ds:Column columnId="ID" valueType="string" columnNum="1"/>
            <ds:Column columnId="time" valueType="real" columnNum="2"/>
            <ds:Column columnId="dv" valueType="real" columnNum="3"/>
        </ds:Definition>
        <ds:Table>
            <!-- SUBJECT 1 -->
            <ds:Row><ct:String>i1</ct:String><ct:Real>0.5</ct:Real><ct:Real>0</ct:Real></ds:Row>
            <ds:Row><ct:String>i1</ct:String><ct:Real>1</ct:Real><ct:Real>1.9</ct:Real></ds:Row>
            <ds:Row><ct:String>i1</ct:String><ct:Real>2</ct:Real><ct:Real>3.3</ct:Real></ds:Row>
            <ds:Row><ct:String>i1</ct:String><ct:Real>3</ct:Real><ct:Real>6.6</ct:Real></ds:Row>
            <ds:Row><ct:String>i1</ct:String><ct:Real>6</ct:Real><ct:Real>9.1</ct:Real></ds:Row>
            <ds:Row><ct:String>i1</ct:String><ct:Real>9</ct:Real><ct:Real>10.8</ct:Real></ds:Row>
            <!-- SUBJECT 2 -->
            <!-- SNIP -->
        </ds:Table>
    </ds:DataSet>
</ObjectiveDataSet>
```

Before that we have to make sure that these values are correctly mapped to variable used in the model which is implemented in the `<VariableMapping>` element. Accordingly, we define the identifier ID and define the variable mapping. Here the $time$ as in the data is mapped to model time t and the measured concentration is mapped to the variable C as in the observation model.

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

```
<ParametersToEstimate>
    <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 12) is that every parameter has to be initialised.

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"></ct:OidRef>
    </mstep:Step>
</mstep:StepDependencies>
```

7.5 Example 4: Estimation with IOV

7.5.1 Description

In this example we will look at more complex trial design and a correspondingly complex variability model. 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 4.5). 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.

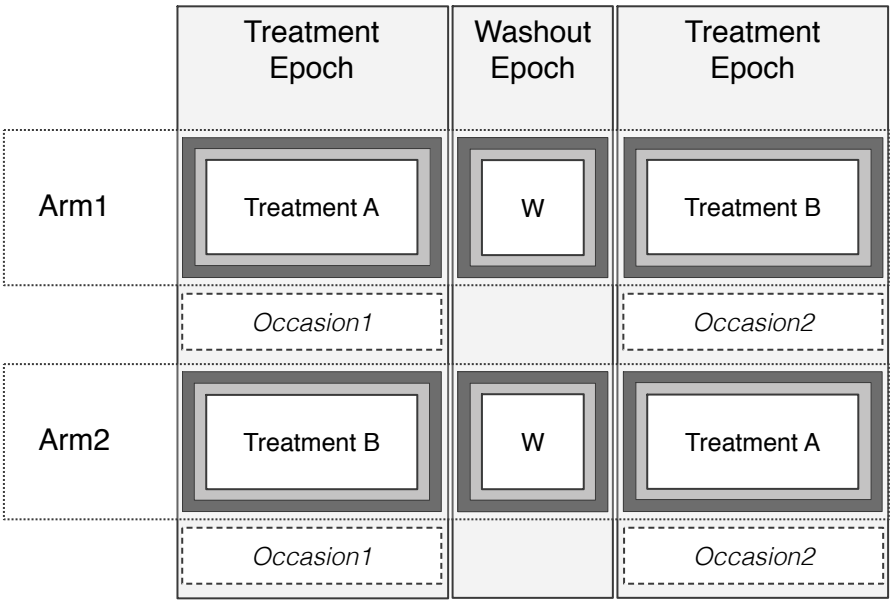


Figure 7.9: Schematic representation of a crossover design with washout. The reader is referred to Figure 5.2 for the colour code used to identify the elements of a trial. See tables 7.7 and 7.8 for the detailed definition of segments, cells, arms, epochs and occasions in this example.

Trial Design

The model features a basic crossover design (see Figure 7.9) 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 treatments, fi treated as occasions, provide a second level of variability – IOV (see section 4.5). This is summarised in Figure 7.10 (see also the listing in section 7.5.3, showing relevant code within the element `<VariabilityModel>`).

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 : 12 : 72]	[0 : 12 : 72

Table 7.5: Arms overview with dosing specification.

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.

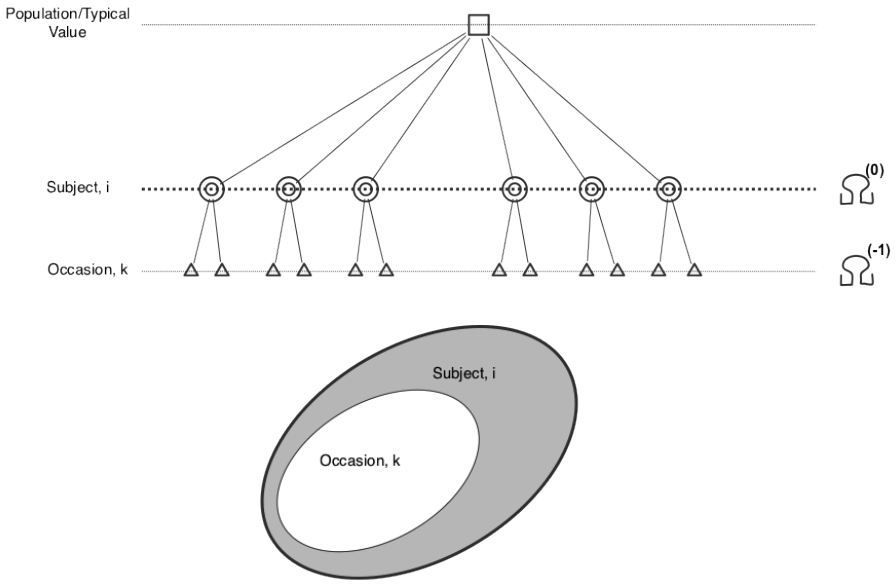


Figure 7.10: Two levels of variability – inter-individual and inter-occasion within individual variability.

Covariate Model

As discussed about all but the 'Sex' covariate is used to capture the variability in the model, see Table 7.6.

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[5]

[5]To improve clarity we have colour coded the contributions of the different levels of variability and the different covariates.

	Sex	Treat	TreatSeq	Occasion
Type	Categorical	Categorical	Categorical	Categorical
Category Count	2	2	2	2
Categories	F, M	A, B	A–B,B–A	1, 2
Reference	F	A	A–B	1

Table 7.6: Covariates overview.

$$\log(ka_i) = \log(ka_{pop}) + \beta_{ka,TreatSeq} 1_{TreatSeq_i=A-B} + \eta_{ka,i} \tag{7.8}$$

$$\begin{aligned}\log(V_{ik}) &= \log(V_{pop}) + \boldsymbol{\beta}_V 1_{S_i=F} + \beta_{V,OCC} 1_{OCC_{ik}=1} \\ &+ \beta_{V,Treat} 1_{Treat_{ik}=A} + \beta_{V,TreatSeq} 1_{TreatSeq_i=A-B} \\ &+ \eta_{V,i}^{(0)} + \eta_{V,ik}^{(-1)}\end{aligned} \tag{7.9}$$

$$\begin{aligned}\log(CL_{ik}) &= \log(CL_{pop}) + \boldsymbol{\beta}_{CL} 1_{S_i=F} + \beta_{CL,OCC} 1_{OCC_{ik}=1} \\ &+ \eta_{CL,i}^{(0)} + \eta_{Cl,ik}^{(-1)}\end{aligned}$$

where

$$\eta_{ka,i}^{(0)} \sim \mathcal{N}(0, \omega_{ka}), \quad \eta_{V,i}^{(0)} \sim \mathcal{N}(0, \omega_V), \quad \eta_{CL,i}^{(0)} \sim \mathcal{N}(0, \omega_{CL}),$$
$$\eta_{V,ik}^{(-1)} \sim \mathcal{N}(0, \gamma_V), \quad \eta_{CL,ik}^{(-1)} \sim \mathcal{N}(0, \gamma_{CL})$$

The full variance-covariance matrix for our model is :

$$\Omega^{(0)} = \begin{pmatrix} \omega_{ka}^2 & 0 & 0 \\ & \omega_V^2 & 0 \\ & & \omega_{CL}^2 \end{pmatrix} \tag{7.10}$$

$$\Omega^{(-1)} = \begin{pmatrix} 0 & 0 & 0 \\ & \gamma_V^2 & 0 \\ & & \gamma_{CL}^2 \end{pmatrix} \tag{7.11}$$

Structural model

The model is first order absorption with linear elimination, with multiple dosing. This is the equivalent to oral1_1cpt_kaVCl (model 8) from [Bertrand and Mentré, 2008, Appendix I].

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	$a = 0.1, \quad b = 0.1$

Modelling Steps

Compared to the last example, we have define here two tasks:

- Estimation of population paramaters.
- Estimation of the individual parameters.

7.5.2 Trial Design

We have summaries the dosing regimen and organisation of the trial design below, see also Figure 7.9.

Segment	Activity	Treatment	DoseTime	DoseSize	Target Variable
TA	OR1	OR bolus	0 : 12 : 72	150	Ac
TA	OR2	OR bolus	0 : 24 : 72	100	Ac

Table 7.7: Segment/activity overview.

Epoch	Occasion	Start time	End time
Treatment Epoch	OCC1	0	180
Washout	–	0	10
Treatment Epoch	OCC2	0	180

Table 7.8: Epoch and occasion definition.

Structure

The implementation of the treatments, in PharmML we use the `<Activity>` element, is different compared to the previous example. See Table 7.7 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 inputType="dose">
            <ct:SymbRef blkIdRef="main" symbIdRef="D"/>
            <ct:Assign>
                <ct:Real>150</ct:Real>
            </ct:Assign>
        </DoseAmount>
        <DosingTimes>
            <ct:SymbRef blkIdRef="main" 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 7.8 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 5.2.1, in <Structure> block we encode the variability which is located below the subject (see the hierarchy of the random variability discussed in section 4.5). 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="model" symbIdRef="iov"/>
    </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:

```
<ObservationsEvent oid="occasions2">
    <ArmRef oidRef="a1"/>
    <ArmRef oidRef="a2"/>
    <ct:VariabilityReference>
        <ct:SymbRef blkIdRef="model" symbIdRef="iov"/>
    </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>
```

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 7.4.3 for how to encode epochs in the <Structure> definition).

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="model" symbIdRef="indiv"/>
    </ct:VariabilityReference>
```

Here we deal only with the IIV so we are done with this aspect.

The next part of the <Population> block was discussed previously, with one exception. Beside the standard assignment of subjects to an Arm and providing information regarding Sex, we need to encode the information about $Treat$, i.e. treatment type considered here as covariate, which varies by definition in this cross-over design as the study progress from $Epoch1$ to $Epoch3$. To encode this we use the *nested table* concept as described in section 6.6. Here the child table is defined by using a <Table> element instead of the usual <Column> element and given the identifier 'treat-tab'. Within the nested table definition another set of relevant columns is specified, $epoch$ and $treat$. Next these nested tables are populated with data as can be seen in the following listing

```
<IndividualTemplate>
    <IndividualMapping>
        <ds:ColumnRef columnIdRef="id"/>
    </IndividualMapping>
    <ArmMapping>
        <ds:ColumnRef columnIdRef="arm"/>
    </ArmMapping>
    <CovariateMapping>
        <ds:ColumnRef columnIdRef="sex"/>
        <ct:SymbRef blkIdRef="c1" symbIdRef="Sex"/>
    </CovariateMapping>
    <IVDependentMapping>
        <ds:ColumnRef columnIdRef="treat-tab"/>
        <EpochMapping>
            <ds:ColumnRef columnIdRef="epoch"/>
        </EpochMapping>
        <CovariateMapping>
            <ds:ColumnRef columnIdRef="treat"></ds:ColumnRef>
            <ct:SymbRef blkIdRef="c1" symbIdRef="Treat"/>
        </CovariateMapping>
    </IVDependentMapping>
</IndividualTemplate>
<ds:DataSet>
    <ds:Definition>
        <ds:Column columnId="id" valueType="id" columnNum="1"/>
        <ds:Column columnId="arm" valueType="id" columnNum="2"/>
        <ds:Column columnId="sex" valueType="id" columnNum="3"/>
        <ds:Table tableId="treat-tab" columnNum="4">
            <ds:Definition>
                <ds:Column columnId="epoch" valueType="id" columnNum="1"/>
                <ds:Column columnId="treat" valueType="id" columnNum="2"/>
            </ds:Definition>
        </ds:Table>
    </ds:Definition>
    <ds:Table>
        <ds:Row>
            <ct:Id>i1</ct:Id>
            <ct:Id>a1</ct:Id>
            <ct:Id>M</ct:Id>
            <ds:Table>
                <ds:Row><ct:Id>ep1</ct:Id><ct:Id>A</ct:Id></ds:Row>
                <ds:Row><ct:Id>ep3</ct:Id><ct:Id>B</ct:Id></ds:Row>
            </ds:Table>
        </ds:Row>
        <!-- SNIP -->
```

here for $Arm1$. Listing

```
<!-- SNIP -->
<ds:Row>
    <ct:Id>i6</ct:Id>
    <ct:Id>a2</ct:Id>
```

```
    <ct:Id>M</ct:Id>
    <ds:Table>
        <ds:Row><ct:Id>ep1</ct:Id><ct:Id>B</ct:Id></ds:Row>
        <ds:Row><ct:Id>ep3</ct:Id><ct:Id>A</ct:Id></ds:Row>
    </ds:Table>
</ds:Row>
<!-- SNIP -->
```

shows one data record for *Arm2*.

7.5.3 Variability Model

In this example the variability model is more complex than before, with IIV and IOV levels of variability, see Figure 7.10. As you will see, in PharmML the complexity comes later – in the parameter model. At this point in the PharmML document all we need to do is define the variability levels to be used in the rest of the document. You can see in the following listing

```
<VariabilityModel blkId="model" type="model">
    <Level symbId="indiv"/>
    <Level symbId="iov1">
        <ParentLevel>
            <ct:SymbRef symbIdRef="indiv"/>
        </ParentLevel>
    </Level>
</VariabilityModel>
```

that this is done simply by listing the variability levels using the `<VariabilityLevel>` element. There are three important points to note here:

1. There is parent-child relationship between the levels of variability. The *Subject* level, in the PharmML it is referenced with the attribute `symbId="indiv"` is higher in the hierarchy and directly above the *Occasion* level, referenced with the attribute `symbId="iov1"` which is exactly what is done using the `<ParentLevel>` in the listing above.

2. 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.

3. 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[6].

So in this example the PharmML document tells us that there are two variability levels and that the lowest level of variability is called "`iov1`". This may seem odd, but to simulate or estimate the model we do not need to know which level of variability is considered IIV and which IOV. We only need to know their level relative to each other. Of course it may be desirable to know this when exchanging a model, and we feel that this information can be provided by annotation of the PharmML document.

7.5.4 Covariate Model

The covariate model describes categorical covariates, listed in Table 7.6, 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

[6] N.B., The numerical levels described in the variability model (section 4.5) are not used.

```
<CovariateModel blkId="c1">
    <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>A-B</ct:Name>
            </Category>
            <Category catId="BA">
                <ct:Name>B-A</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>
```

7.5.5 Parameter Model

In example 1 (section 7.2) 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="model" 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:SymbRef></ct:Assign>
            </PopulationParameter>
            <Covariate>
                <ct:SymbRef blkIdRef="c1" 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 (7.8). 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 $\beta_{ka,TreatSeq}1_{TreatSeq_i=A-B}$ in (7.8). 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_TreatSet"/>
<RandomVariable symbId="eta_V">
    <ct:VariabilityReference>
        <ct:SymbRef blkIdRef="model" 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="model" symbIdRef="iov1"/>
    </ct:VariabilityReference>
    <NormalDistribution xmlns="http://www.uncertml.org/3.0" definition="">
        <mean><rVal>0</rVal></mean>
        <stddev><var varId="omega_ka"/></stddev>
    </NormalDistribution>
</RandomVariable>
```

and this listing

```
<IndividualParameter symbId="V">
    <GaussianModel>
        <Transformation>log</Transformation>
        <LinearCovariate>
            <PopulationParameter>
                <ct:Assign><ct:SymbRef symbIdRef="pop_ka"></ct:SymbRef></ct:Assign>
            </PopulationParameter>
            <Covariate>
                <ct:SymbRef blkIdRef="c1" symbIdRef="sex"/>
                <FixedEffect>
                    <ct:SymbRef symbIdRef="beta_V"/>
                    <Category catId="F"/>
                </FixedEffect>
            </Covariate>
            <Covariate>
                <ct:SymbRef blkIdRef="c1" symbIdRef="Occasion"/>
                <FixedEffect>
                    <ct:SymbRef symbIdRef="beta_V_occ1"/>
                    <Category catId="occ1"/>
                </FixedEffect>
            </Covariate>
            <Covariate>
                <ct:SymbRef blkIdRef="c1" symbIdRef="Treat"/>
```

```
                <FixedEffect>
                    <ct:SymbRef symbIdRef="beta_V_treat"/>
                    <Category catId="A"/>
                </FixedEffect>
            </Covariate>
            <Covariate>
                <ct:SymbRef blkIdRef="c1" symbIdRef="TreatSeq"/>
                <FixedEffect>
                    <ct:SymbRef symbIdRef="beta_V_treatseq"/>
                    <Category catId="AB"/>
                </FixedEffect>
            </Covariate>
        </LinearCovariate>
        <RandomEffects>
            <ct:SymbRef symbIdRef="eta_V"/>
            <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 $\eta^{(0)}_{V,i}$ and $\eta^{(-1)}_{V,ik}$ in (7.9). 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 (7.10) and (7.11) are implicitly defined because all the random effects follow a normal distribution and we can deduce the diagonal of each matrix at each level of variability from the definition of each random effect.

7.5.6 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.

7.6 Example 5: Estimation with individual dosing

7.6.1 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. 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 patient specific administration scenarios.

7.6.2 Trial design

We will start with the definition of <Structure>, <Population>. The next language element, <IndividualDosing>, is, as mentioned above, new but it's easy to understand.

Structure

Figure 7.11 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 5 the design element <Cell> comprises the essential elements specifying the information about the arm, epoch and segment/activities. contains treatment definition, here an IV bolus administration, defined in the <Activity> element. Figure 7.12 shows the general relationship of these elements (left) and how it applies to the current example (right). See the following listing

```
<!-- BLOCK II: TRIAL DEFINITION -->
<TrialDesign xmlns="http://www.pharmml.org/2013/03/TrialDesign">

    <!-- STRUCTURE -->
    <Structure>
        <Epoch oid="epoch1">
            <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"/>
        </Segment>
        <Activity oid="bolusIV">
            <Bolus>
                <DoseAmount inputType="target">
                    <ct:SymbRef blkIdRef="sm1" symbIdRef="C"/>
                </DoseAmount>
            </Bolus>
        </Activity>
    </Structure>
```

for the PharmML implementation.

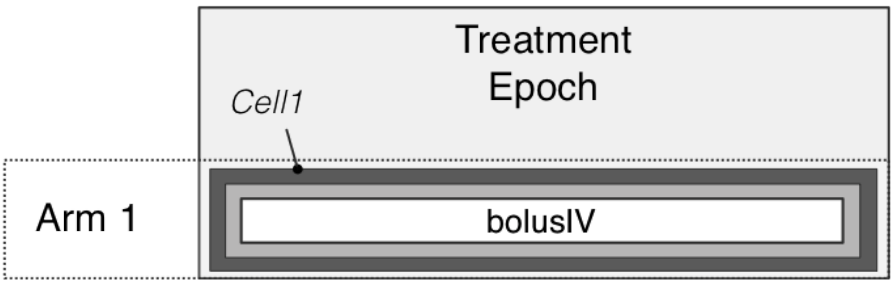


Figure 7.11: Design overview: single arm design.

Segment	Activity	Treatment	DoseTime	DoseSize	Target Variable
TA	bolusIV	IV bolus	individual	1	C

Table 7.9: Segment/activity overview.

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

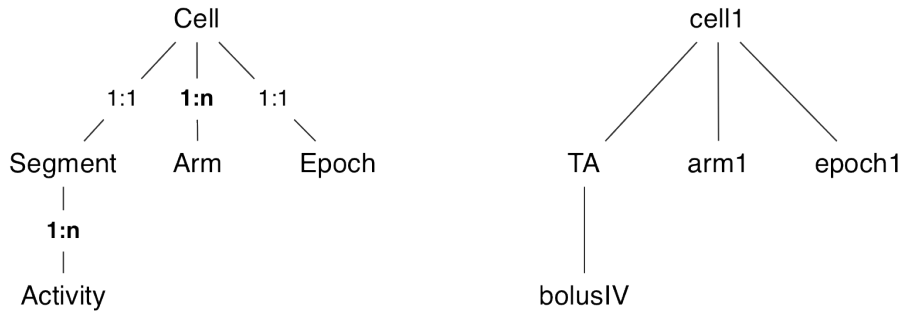


Figure 7.12: General cell hierarchy (left); The root of the trial design structure hierarchy is the 'Cell' which can contain one 'Segment', one 'Epoch' and multiple 'Arms'. The 'Segment' element can have multiple child elements, the 'Activities', e.g. treatments or a washout. (right) An example of how it is applied in [Ribba et al., 2012].

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 -->
<Population>
    <IndividualTemplate>
        <IndividualMapping>
            <ColumnRef xmlns="http://www.pharmml.org/2013/08/Dataset" columnIdRef="ID"/>
        </IndividualMapping>
        <ArmMapping>
            <ColumnRef xmlns="http://www.pharmml.org/2013/08/Dataset" columnIdRef="ARM"/>
        </ArmMapping>
        <ReplicateMapping>
            <ColumnRef xmlns="http://www.pharmml.org/2013/08/Dataset" columnIdRef="REP"/>
        </ReplicateMapping>
    </IndividualTemplate>
    <DataSet xmlns="http://www.pharmml.org/2013/08/Dataset">
        <Definition>
            <Column columnId="ID" valueType="string" columnNum="1"/>
            <Column columnId="ARM" valueType="string" columnNum="2"/>
            <Column columnId="REP" valueType="int" columnNum="3"/>
        </Definition>
        <Table>
            <Row>
                <ct:String>i</ct:String>
                <ct:String>a1</ct:String>
                <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.

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 7.10.

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

ID	TIME	DV	DOSE	ID	TIME	DV	DOSE	ID	TIME	DV	DOSE
1	0	.	.	1	116.23	72.04	.	20	13.4	.	1
1	3.43	45.7	.	1	121.87	90.16	.	20	17.13	42.62	.
1	5.3	48.03	.	...	...	...	...	21	0	.	.
1	42.13	71.34	.	...	...	...	...	21	1.5	.	1
1	52.63	79.3	.	20	0	48.61	.	21	3.17	.	1
1	54.57	.	1	20	4	.	1	21	4.85	.	1
1	57.53	72.3	.	20	5.88	.	1	21	6.52	.	1
1	59.77	.	1	20	6.7	46.64	.	21	8.19	.	1
1	63.3	72.07	.	20	7.76	.	1	21	9.77	72.35	.
1	68.97	70.24	.	20	9.27	44.97	.	21	9.87	.	1
1	76.53	66.81	.	20	9.64	.	1	21	14.23	66.96	.
1	94.53	60.48	.	20	11.52	.	1	21	18.13	56.79	.
1	106.1	62	.	20	13.23	42.96	.	21	23.9	60.06	.

Table 7.10: Data used in [Ribba et al., 2012], an excerpt from the experimental data set in NONMEM format. The columns are: the identifier, ID, time for measurements and dosing events, dependent variable, DV, which stand 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.

```
<IndividualDosing>
    <ActivityRef oidRef="bolusIV"/>
    <IndividualRef columnIdRef="ID"/>
    <DataSet xmlns="http://www.pharmml.org/2013/08/Dataset">
        <Definition>
            <Column columnId="ID" valueType="string" columnNum="1"/>
            <Column columnId="TIME" valueType="real" columnNum="2"/>
            <Column columnId="DOSE" valueType="real" columnNum="3"/>
        </Definition>
        <Table>
            <!-- subject 1 -->
            <Row><ct:String>i1</ct:String><ct:Real>54.57</ct:Real><ct:Real>1</ct:Real></Row>
            <Row><ct:String>i1</ct:String><ct:Real>59.77</ct:Real><ct:Real>1</ct:Real></Row>
            <!-- SNIP -->
            <!-- subject 21 -->
            <Row><ct:String>i21</ct:String><ct:Real>1.5</ct:Real><ct:Real>1</ct:Real></Row>
            <Row><ct:String>i21</ct:String><ct:Real>3.17</ct:Real><ct:Real>1</ct:Real></Row>
            <Row><ct:String>i21</ct:String><ct:Real>4.85</ct:Real><ct:Real>1</ct:Real></Row>
            <Row><ct:String>i21</ct:String><ct:Real>6.52</ct:Real><ct:Real>1</ct:Real></Row>
            <Row><ct:String>i21</ct:String><ct:Real>8.19</ct:Real><ct:Real>1</ct:Real></Row>
            <Row><ct:String>i21</ct:String><ct:Real>9.87</ct:Real><ct:Real>1</ct:Real></Row>
        </Table>
    </DataSet>
</IndividualDosing>
</TrialDesign>
```

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.

7.6.3 Structural model definition

The following ODE system is defined:

$$\begin{aligned}
\frac{dC}{dt} &= -\boldsymbol{KDE} \times C \\
\frac{dP}{dt} &= \lambda_P \times P\left(1 - \frac{P^\star}{K}\right) + k_{QPP} \times Q_P - k_{PQ} \times P - \gamma \times C \times \boldsymbol{KDE} \times P \\
\frac{dQ}{dt} &= k_{PQ} \times P - \gamma \times C \times KDE \times Q \\
\frac{dQ_P}{dt} &= \gamma \times C \times \boldsymbol{KDE} \times Q - k_{QPP} \times Q_P - \delta_{QP} \times Q_P
\end{aligned}$$

$$P^\star = P + Q + Q_P$$

with initial conditions

$$C(t=0) = 1; \quad P(t=0) = P0; \quad Q(t=0) = Q0; \quad Q_P(t=0) = 0.$$

Defining 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

```
<!-- P0 -->
<SimpleParameter symbId="pop_P0"/>
<SimpleParameter symbId="omega_P0"/>
<RandomVariable symbId="eta_P0">
    <ct:VariabilityReference>
        <ct:SymbRef blkIdRef="sm1" 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_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>
<!-- QP0 -->
<SimpleParameter symbId="QP0">
    <ct:Assign>
        <ct:Real>0</ct:Real>
    </ct:Assign>
</SimpleParameter>
```

where the definition of the distribution for the initial condition P_0 is shown.

7.6.4 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.

CHAPTER 8

Unresolved issues

Issue: 1 Can symbol resolution rules be simplified? Currently the rules for symbol resolution (see section 12.3.2) define the class of symbol that can be referred to. For example in the GeneralCovariate parameter definition, we restrict any equation defined there to only reference parameters and covariates — random variables are prohibited. This ensures that this part of the PharmML document is used correctly, but it is it too restrictive?

Issue: 2 More work to define operations and algorithms in PharmML. There is not specification for what estimation operations and algorithms should be supported by PharmML. Ideally algorithm definitions will be supported by external resources such as KiSAO (`http://biomodels.net/kisao/`), but there is no support there yet.

Issue: 3 The way we map a dataset column to an independent variable is not consistent. In the Estimation Step we map to the independent variable symbol (t) using a `<SymbRef>` element and in the Trial Design we use the `<IndependentVariableMapping>` element. It will simplify the rules if w are consistent.

Issue: 4 Units We had planned to introduce units into this release using the mechanism adopted by SBML. However, their approach does not enable the encoding of temperature in either Fahrenheit or Celsius (because these conversions require the addition of a constant). SBML only alow temperature in Kelvin as a result. Do we wish to follow their approach or try and find a different solution?

Issue: 5 Interpolation When estimating from experimental data it is often necessary to use time-points between those for which we have experimental data. Software interpolate between the known data-points to obtain a value, but of course there is more than one way to do this. The approach taken is tool specific, so the question is do we wish to specify what interpolation method is used in the estimation step?

Issue: 6 Use of the `columnNum` in the dataset definition. In version 0.1.0 of PharmML the dataset read from a tabular ascii file, such as a tab delimited file. Because of this we used the `columnNum` to map the column in the data-file to that in the dataset. Now that the data is defined in XML this use of the `columnNum` is not used and the columnNum had been reinterpreted to define the order of the column in the dataset definition. This is superfluous and the column order could be easily defined by the order in the XML document - as it is for the contents of the `<Row>` element.

Part II

Technical Reference

CHAPTER 9

Purpose and Organisation

9.1 Introduction

The technical reference, as you might expect, provides the detail of the PharmML specification and aims to provide a definition of the language in a form that is useful to those developing software tools that support the language, and reviewers who wish to understand its fine details.

The reference is organised to first provide background about the design guidelines and conventions used and then give an annotated description of the XML Schema definition (both in chapter 10). Then an enumeration of the language's rules is given (in chapter 12)

9.2 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. For convenience we provide a documented form of the schema definition in chapter 10, but if there are any discrepancies between these, then the version in the file should be regarded as definitive.

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 12. 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.

CHAPTER 10

The XML Schema Definition

10.1 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.

10.1.1 XML Schema Compliance

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

10.1.2 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.

10.1.3 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

[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/
[4] http://www.oracle.com/technetwork/java/design-patterns-142138.html

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.

10.1.4 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. However, we have followed the W3C conventions[6] and used the following form:

```
http://pharmml.org/Year/Month/Resource
```

Here the year and month define when the URL was created and the *Resource* provides a short, but descriptive name of the purpose of the schema.

10.1.5 Elements

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

10.1.6 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[7]. For these reasons the use of default attribute values is forbidden.

Attributes should never be qualified by a namespace[8]. This corresponds to the XML Schema declaration `attributeFormDefault="unqualified"`.

10.1.7 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[9]. 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.

[5] `http://www.ibm.com/developerworks/library/x-namcar/index.html`
[6] `http://www.w3.org/Provider/Style/URI`
[7] ref
[8] `http://www.xml.com/pub/a/2002/11/20/schemas.html?page=3`
[9] `http://http://www.ibm.com/developerworks/xml/library/x-eleatt/index.html`

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

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.

10.1.8 Keys and Key References

We use the XML Schema key and keyref mechanisms only. ID and IDREF types should not be used[10].

10.1.9 Versioning Strategy

PharmML will change over time and indeed we have described the versioning strategy for PharmML earlier in this document (section 1.6.1 on page 6). 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[11]. 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.
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.

10.2 XML Schema Organisation

PharmML is a large language and the XML Schema definition is correspondingly large too. Fortunately, the language is naturally organised into three sections (see chapter 6): Model Definition, Trial Design, and Modelling Steps, which provides us with a convenient way to modularise the XML

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

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 following xsd files.

File Name	Namespace URI	Description
pharmml.xsd	http://www.pharmml.org/2013/03/PharmML	The overall PharmML definition that includes all the other components.
modelDefinition.xsd	http://www.pharmml.org/2013/03/ModelDefinition	Defines the model definition section.
trialDesign.xsd	http://www.pharmml.org/2013/03/TrialDesign	Defines the trial design section.
modellingSteps.xsd	http://www.pharmml.org/2013/03/ModellingSteps	Defines the modelling steps section.
commonTypes.xsd	http://www.pharmml.org/2013/03/CommonTypes	Defines the type definitions and structures common to the above schema definitions.
dataset.xsd	http://www.pharmml.org/2013/08/Dataset	Defined the dataset and related structures that is used in the trial design and modelling steps to represent tabular data.
maths.xsd	http://www.pharmml.org/2013/03/Maths	Defines the representation of mathematical expressions.
UncertML30.xsd	http://www.uncertml.org/3.0	Defines the probability distributions provided by PharmML.

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 11

Schema Description

In this chapter the detailed documentation of the XML Schemas used to define PharmML is included. All the schemas except the UncertML schema is included. UncertML is maintained separately from PharmML and for more information you should go to `http://www.uncertml.org`.

11.1 PharmML

1 Namespace: "http://www.pharmml.org/2013/03/PharmML"

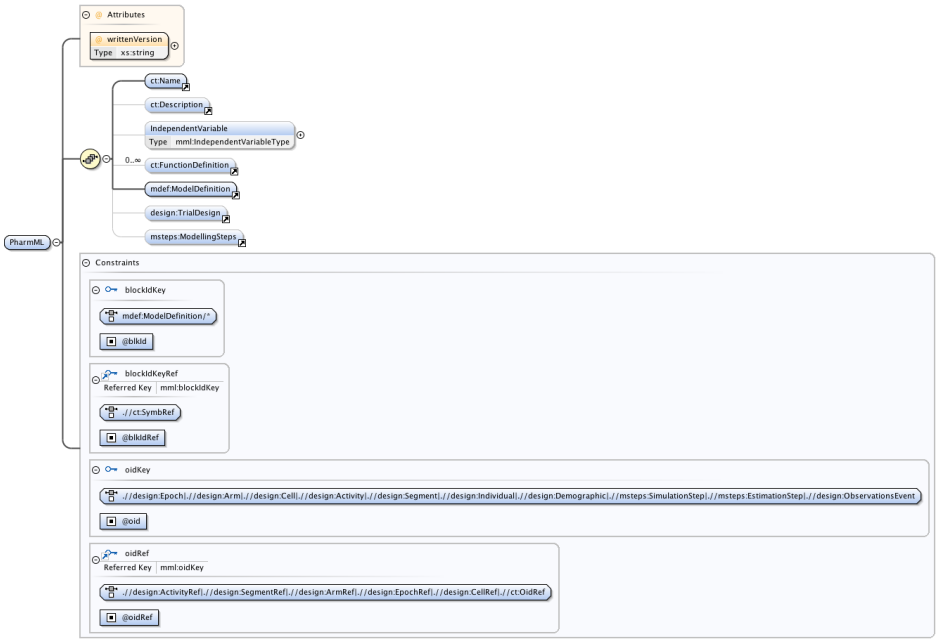
1.1 Schema(s)

1.1.1 Main schema `pharmml.xsd`

Namespace	http://www.pharmml.org/2013/03/PharmML

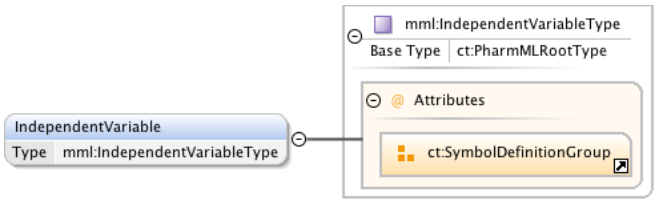
1.2 Element(s)

1.2.1 Element `mml:PharmML`

Namespace	http://www.pharmml.org/2013/03/PharmML
Annotations	The root element of the PharmML document.
Diagram	

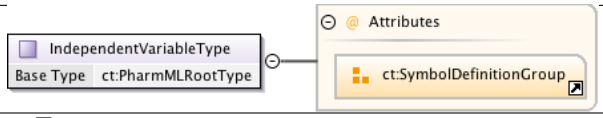
Attributes	QName	Type	Use			
	writtenVersion	xs:string	required			
		The version of PharmML that this document was compliant with when this document was written.				

1.2.2 Element `mml:PharmML /mml:IndependentVariable`

Namespace	http://www.pharmml.org/2013/03/PharmML
Diagram	
Type	mml:IndependentVariableType

1.3 Complex Type(s)

1.3.1 Complex Type `mml:IndependentVariableType`

Namespace	http://www.pharmml.org/2013/03/PharmML
Annotations	Type used to specify the indepdent variable of the model.
Diagram	
Type	extension of PharmMLRootType

2 Namespace: ""

2.1 Attribute(s)

2.1.1 Attribute `mml:PharmML /@writtenVersion`

Namespace	No namespace
Annotations	The version of PharmML that this document was compliant with when this document was written.
Type	xs:string

11.2 Model Definition

1 Namespace: "http://www.pharmml.org/2013/03/ModelDefinition"

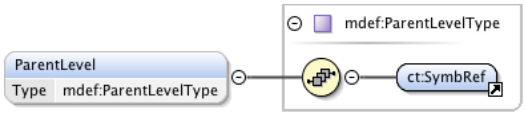
1.1 Schema(s)

1.1.1 Main schema `modelDefinition.xsd`

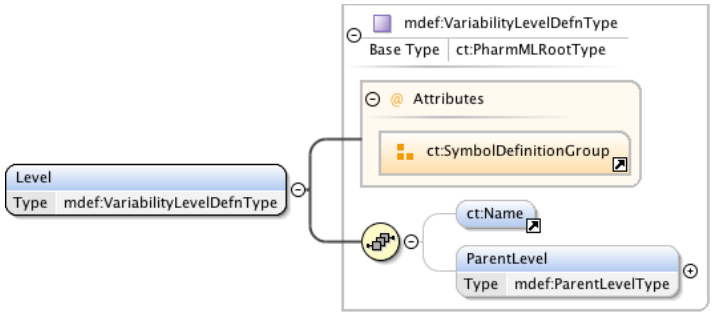
Namespace	http://www.pharmml.org/2013/03/ModelDefinition

1.2 Element(s)

1.2.1 Element `mdef:VariabilityLevelDefnType /mdef:ParentLevel`

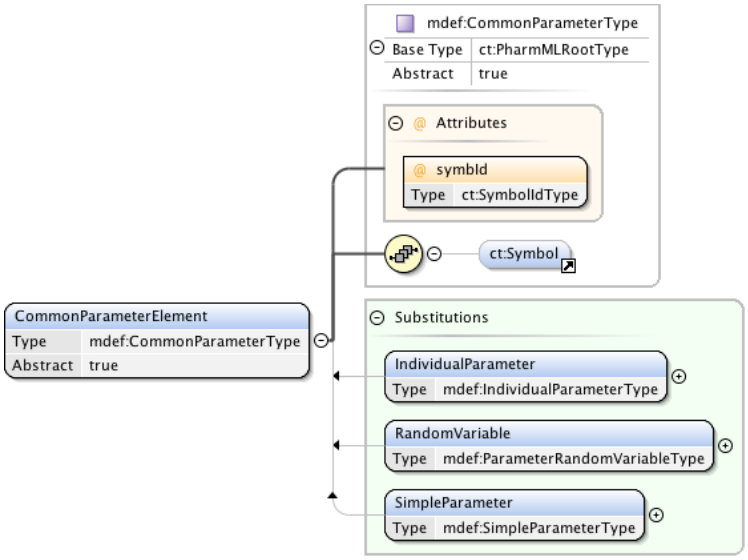
Namespace	http://www.pharmml.org/2013/03/ModelDefinition
Annotations	Parent variability level.
Diagram	
Type	mdef:ParentLevelType

1.2.2 Element `mdef:VariabilityDefnBlock /mdef:Level`

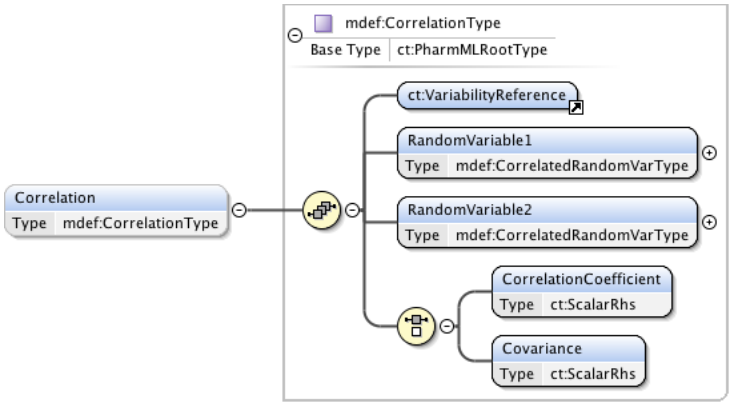
Namespace	http://www.pharmml.org/2013/03/ModelDefinition
Annotations	A variability level within the model.
Diagram	
Type	mdef:VariabilityLevelDefnType

1.2.3 Element `mdef:CommonParameterElement`

Namespace	http://www.pharmml.org/2013/03/ModelDefinition
Annotations	Abstract element defining parameters used in the model.

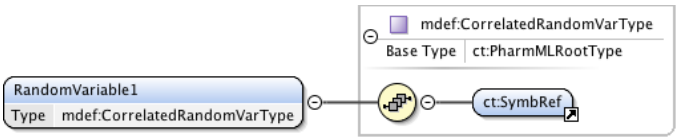
Diagram	
Type	mdef:CommonParameterType
Substitution Group	• mdef:SimpleParameter • mdef:IndividualParameter • mdef:RandomVariable

1.2.4 Element `mdef:CommonParameterModelType /mdef:Correlation`

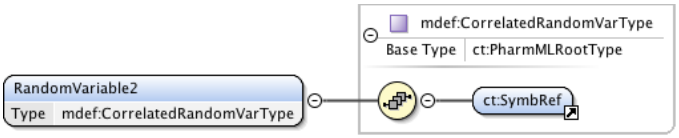
Namespace	http://www.pharmml.org/2013/03/ModelDefinition
Annotations	Defines the correlation between the random effects.
Diagram	
Type	mdef:CorrelationType

1.2.5 Element `mdef:CorrelationType /mdef:RandomVariable1`

Namespace	http://www.pharmml.org/2013/03/ModelDefinition
Annotations	The first correlated parameter.

Diagram	
Type	mdef:CorrelatedRandomVarType

1.2.6 Element `mdef:CorrelationType /mdef:RandomVariable2`

Namespace	http://www.pharmml.org/2013/03/ModelDefinition
Annotations	The second correlated parameter.
Diagram	
Type	mdef:CorrelatedRandomVarType

1.2.7 Element `mdef:CorrelationType /mdef:CorrelationCoefficient`

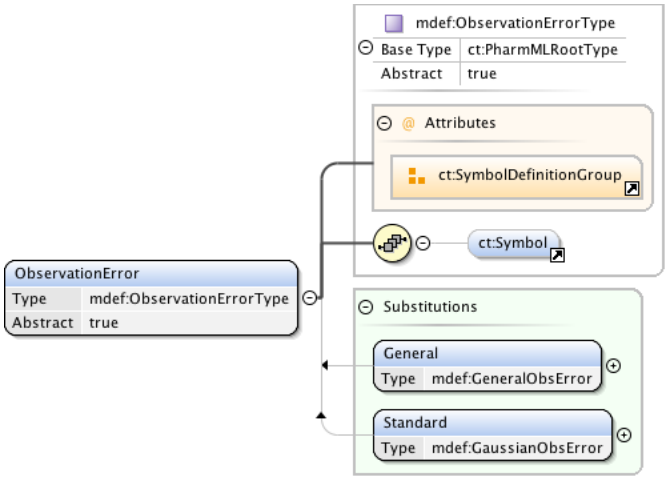
Namespace	http://www.pharmml.org/2013/03/ModelDefinition
Annotations	The correlation co-efficient variable.
Diagram	
Type	ScalarRhs

1.2.8 Element `mdef:CorrelationType /mdef:Covariance`

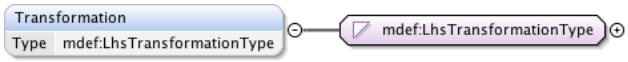
Namespace	http://www.pharmml.org/2013/03/ModelDefinition
Annotations	The covariance for both parameters.
Diagram	
Type	ScalarRhs

1.2.9 Element `mdef:ObservationError`

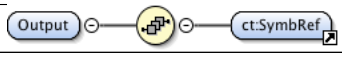
Namespace	http://www.pharmml.org/2013/03/ModelDefinition
Annotations	Abstract element defining an observation error.

Diagram	
Type	mdef:ObservationErrorType
Substitution Group	• mdef:Standard • mdef:General

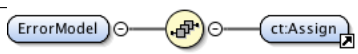
1.2.10 Element `mdef:GaussianObsError /mdef:Transformation`

Namespace	http://www.pharmml.org/2013/03/ModelDefinition		
Annotations	Defines the transformation (u) applied to both sides of equation.		
Diagram			
Type	mdef:LhsTransformationType		
Facets	enumeration	`log`	Natural log transformation.
	enumeration	`logit`	Logit transformation.
	enumeration	`probit`	Probit transformation.

1.2.11 Element `mdef:GaussianObsError /mdef:Output`

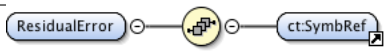
Namespace	http://www.pharmml.org/2013/03/ModelDefinition
Annotations	The output variable from the structural model.
Diagram	

1.2.12 Element `mdef:GaussianObsError /mdef:ErrorModel`

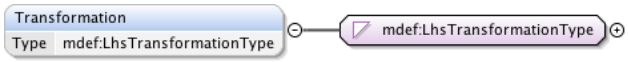
Namespace	http://www.pharmml.org/2013/03/ModelDefinition
Annotations	The error model (g) to apply to the residual error.
Diagram	

1.2.13 Element `mdef:GaussianObsError /mdef:ResidualError`

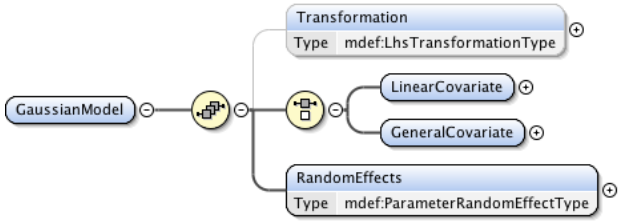
Namespace	http://www.pharmml.org/2013/03/ModelDefinition

Annotations	The residual error (eps).
Diagram	

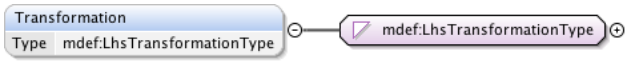
1.2.14 Element `mdef:GeneralObsError /mdef:Transformation`

Namespace	http://www.pharmml.org/2013/03/ModelDefinition		
Annotations	Defines a transformation applied to the left-hand-side of the residual error equation.		
Diagram			
Type	mdef:LhsTransformationType		
Facets	enumeration	`log`	Natural log transformation.
	enumeration	`logit`	Logit transformation.
	enumeration	`probit`	Probit transformation.

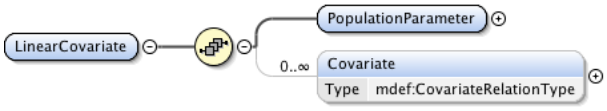
1.2.15 Element `mdef:IndividualParameterType /mdef:GaussianModel`

Namespace	http://www.pharmml.org/2013/03/ModelDefinition		
Annotations	Defines a Gaussian model, with either linear or non-linear covariates.		
Diagram			

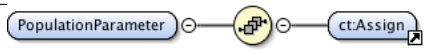
1.2.16 Element `mdef:IndividualParameterType /mdef:GaussianModel /mdef:Transformation`

Namespace	http://www.pharmml.org/2013/03/ModelDefinition		
Annotations	The transformation (h) applied to both sides of the equation.		
Diagram			
Type	mdef:LhsTransformationType		
Facets	enumeration	`log`	Natural log transformation.
	enumeration	`logit`	Logit transformation.
	enumeration	`probit`	Probit transformation.

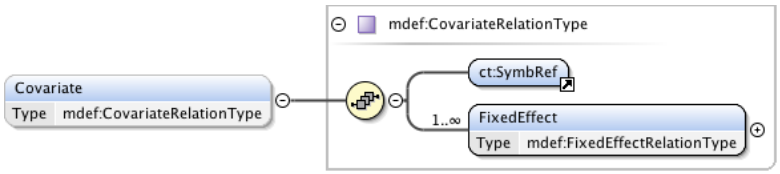
1.2.17 Element `mdef:IndividualParameterType /mdef:GaussianModel /mdef:LinearCovariate`

Namespace	http://www.pharmml.org/2013/03/ModelDefinition
Annotations	Defines the linear covariate model: h(psi_pop) + beta c_i
Diagram	

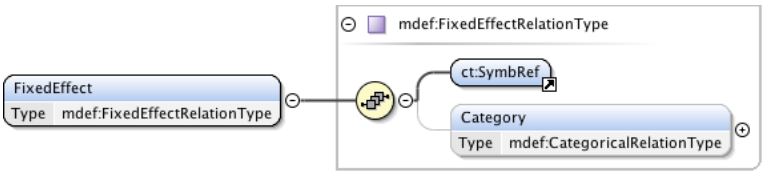
1.2.18 Element mdef:IndividualParameterType /mdef:GaussianModel /mdef:LinearCovariate /mdef:PopulationParameter

Namespace	http://www.pharmml.org/2013/03/ModelDefinition
Annotations	The population parameter: psi_pop.
Diagram	

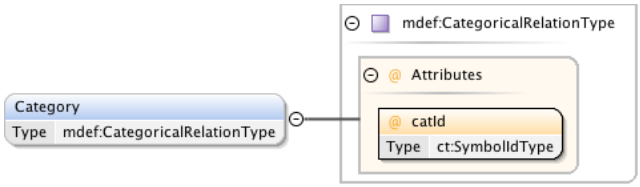
1.2.19 Element mdef:IndividualParameterType /mdef:GaussianModel /mdef:LinearCovariate /mdef:Covariate

Namespace	http://www.pharmml.org/2013/03/ModelDefinition
Annotations	Defines the linear covates: beta c_i
Diagram	
Type	mdef:CovariateRelationType

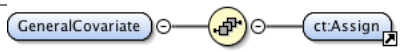
1.2.20 Element mdef:CovariateRelationType /mdef:FixedEffect

Namespace	http://www.pharmml.org/2013/03/ModelDefinition
Annotations	The fixed effect relating the parameter and covariate.
Diagram	
Type	mdef:FixedEffectRelationType

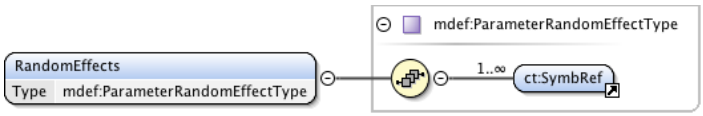
1.2.21 Element mdef:FixedEffectRelationType /mdef:Category

Namespace	http://www.pharmml.org/2013/03/ModelDefinition
Annotations	Specifies the category value of the covariate that must apply when this fixed effect is to be used in the parameter equation. This is equivalent to specifying the following: 1_cov=cat . beta.
Diagram	
Type	mdef:CategoricalRelationType
Attributes	**QName** / **Type** / **Use**: **catId** — SymbolIdType — required Specifies the category value of the covariate to which this relationship applies. For example if a covariate is sex then the Female category may be specified as catId="F".

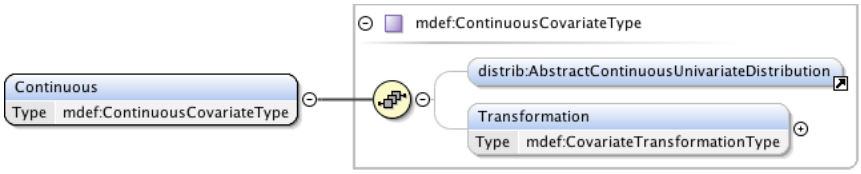
1.2.22 Element `mdef:IndividualParameterType /mdef:GaussianModel /mdef:GeneralCovariate`

Namespace	http://www.pharmml.org/2013/03/ModelDefinition
Annotations	A general covariate model definition. This can be used to define a non-linear covariate model. This equates to H in the above definitions.
Diagram	

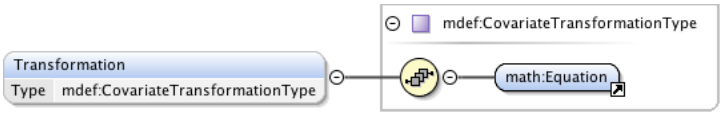
1.2.23 Element `mdef:IndividualParameterType /mdef:GaussianModel /mdef:RandomEffects`

Namespace	http://www.pharmml.org/2013/03/ModelDefinition
Annotations	The random effects (eta) used in the gaussian parameter model.
Diagram	
Type	mdef:ParameterRandomEffectType

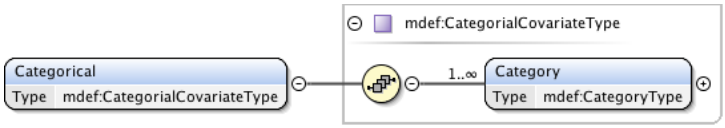
1.2.24 Element `mdef:CovariateDefinitionType /mdef:Continuous`

Namespace	http://www.pharmml.org/2013/03/ModelDefinition
Annotations	Specifies a continuous covariate.
Diagram	
Type	mdef:ContinuousCovariateType

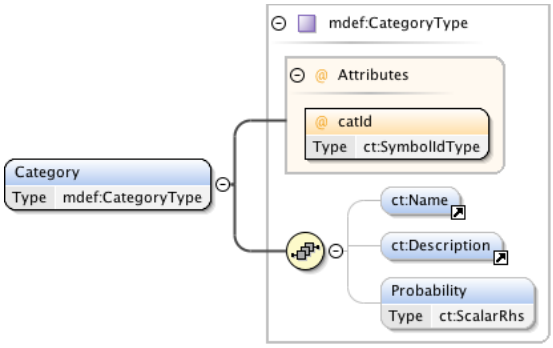
1.2.25 Element `mdef:ContinuousCovariateType /mdef:Transformation`

Namespace	http://www.pharmml.org/2013/03/ModelDefinition
Annotations	The transformation to be applied when the covariate is ued.
Diagram	
Type	mdef:CovariateTransformationType

1.2.26 Element `mdef:CovariateDefinitionType /mdef:Categorical`

Namespace	http://www.pharmml.org/2013/03/ModelDefinition
Annotations	Specifies a categprical covariate.
Diagram	
Type	mdef:CategorialCovariateType

1.2.27 Element `mdef:CategorialCovariateType /mdef:Category`

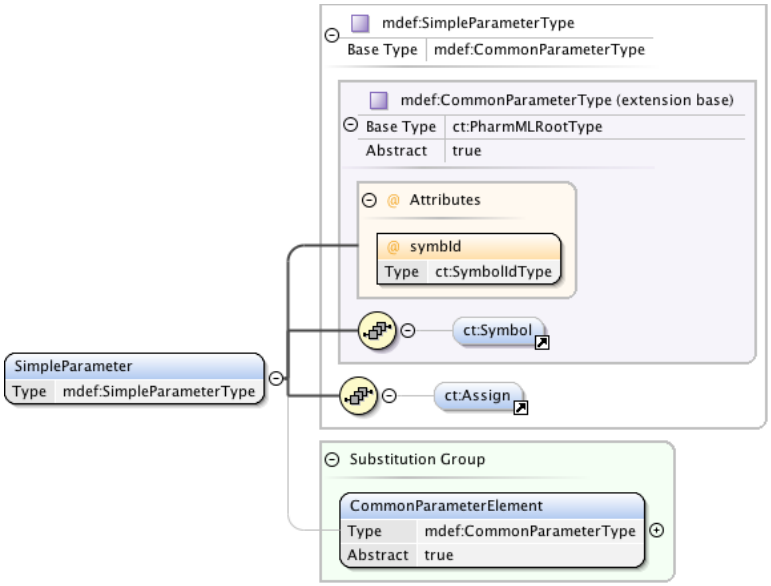
Namespace	http://www.pharmml.org/2013/03/ModelDefinition
Annotations	A category of the categorical covariate.
Diagram	
Type	mdef:CategoryType

Attributes	QName	Type	Use			
	catId	SymbolIdType	required			
		The identifier of the category.				

1.2.28 Element `mdef:CategoryType /mdef:Probability`

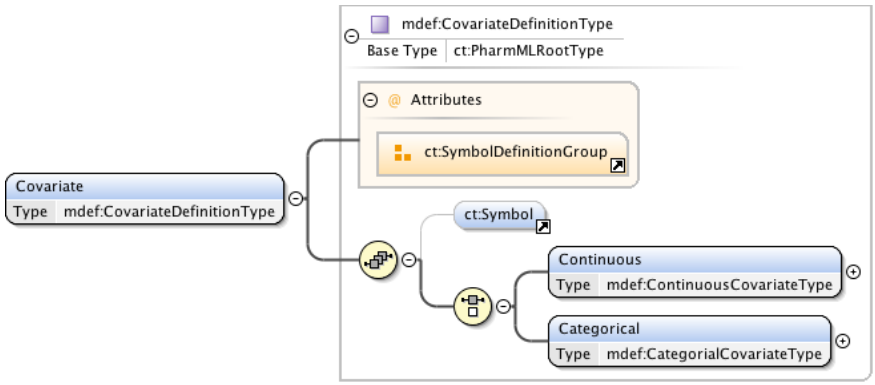
Namespace	http://www.pharmml.org/2013/03/ModelDefinition
Annotations	The definition of the probability associated with this category.
Diagram	
Type	ScalarRhs

1.2.29 Element `mdef:SimpleParameter`

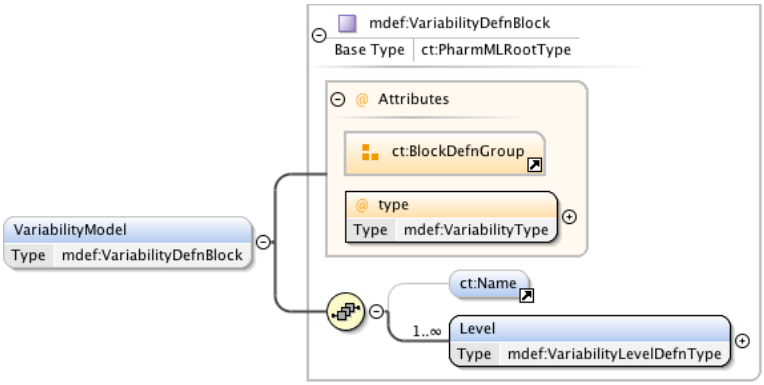
Namespace	http://www.pharmml.org/2013/03/ModelDefinition
Annotations	Defines a simple parameter.
Diagram	
Type	mdef:SimpleParameterType

Substitution Group Affiliation	• mdef:CommonParameterElement

1.2.30 Element `mdef:CovariateModelType /mdef:Covariate`

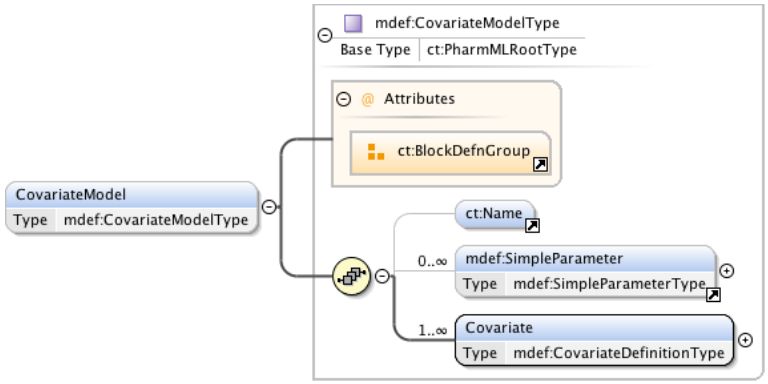
Namespace	http://www.pharmml.org/2013/03/ModelDefinition
Annotations	Defines a covariate.
Diagram	
Type	mdef:CovariateDefinitionType

1.2.31 Element `mdef:ModelDefinitionType /mdef:VariabilityModel`

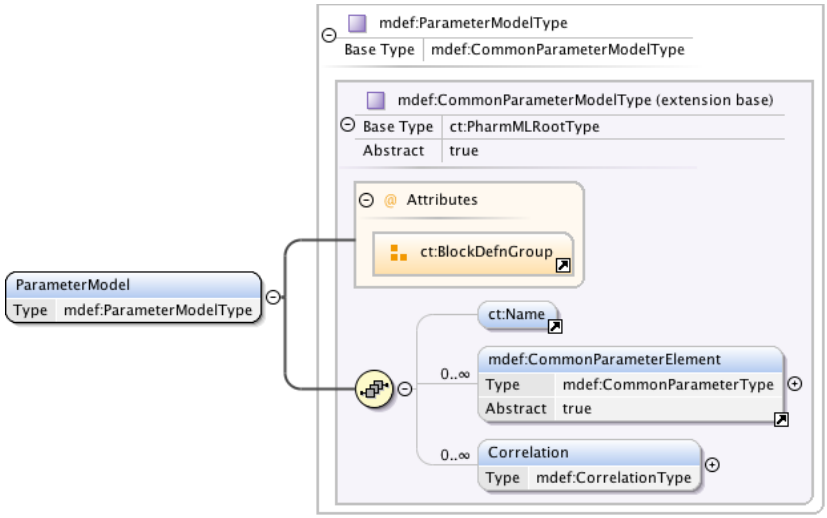
Namespace	http://www.pharmml.org/2013/03/ModelDefinition
Annotations	A variability level.
Diagram	
Type	mdef:VariabilityDefnBlock

1.2.32 Element `mdef:ModelDefinitionType /mdef:CovariateModel`

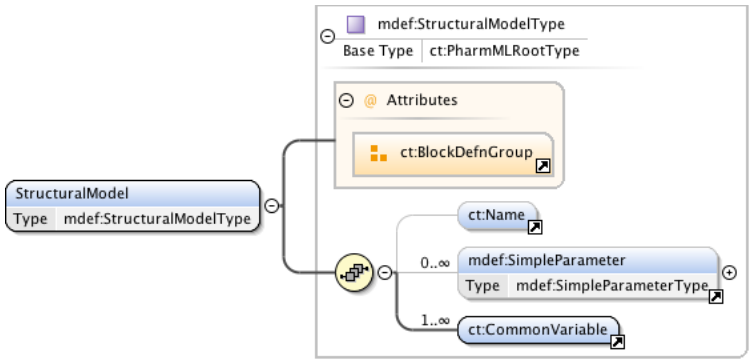
Namespace	http://www.pharmml.org/2013/03/ModelDefinition
Annotations	A covariate model.

Diagram	
Type	mdef:CovariateModelType

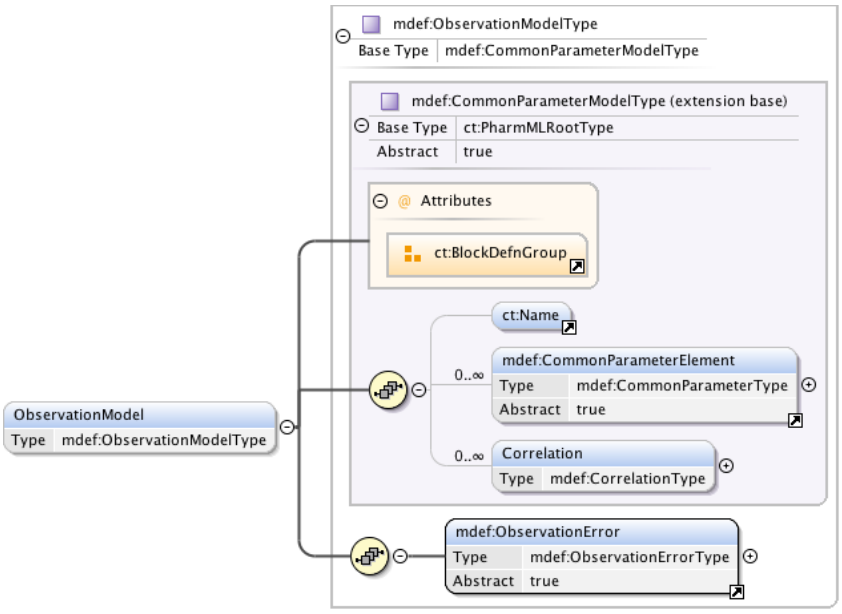
1.2.33 Element `mdef:ModelDefinitionType /mdef:ParameterModel`

Namespace	http://www.pharmml.org/2013/03/ModelDefinition
Annotations	A parameter model.
Diagram	
Type	mdef:ParameterModelType

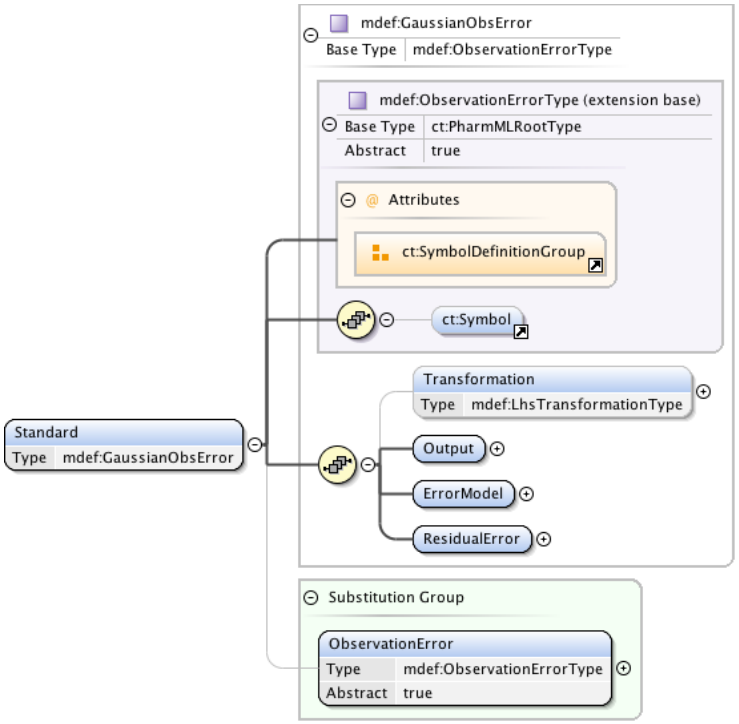
1.2.34 Element `mdef:StructuralModel`

Namespace	http://www.pharmml.org/2013/03/ModelDefinition
Annotations	Defines a structural model.
Diagram	
Type	mdef:StructuralModelType

1.2.35 Element `mdef:ModelDefinitionType /mdef:ObservationModel`

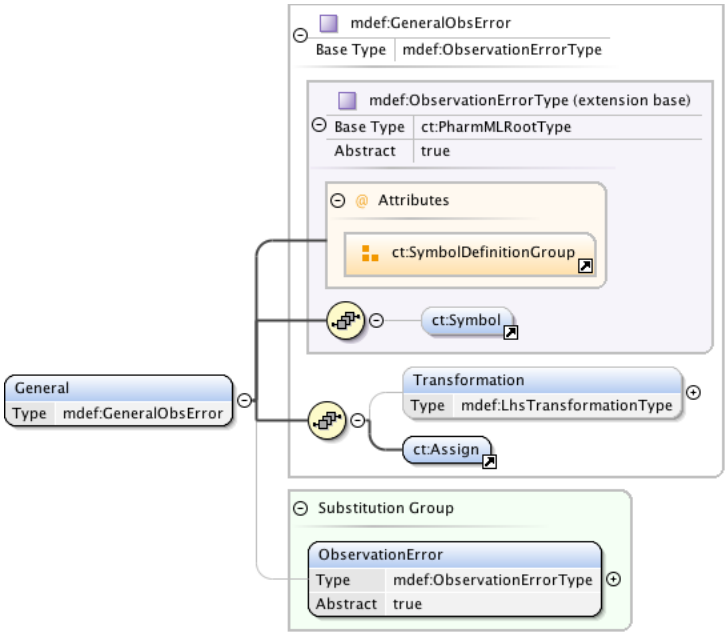
Namespace	http://www.pharmml.org/2013/03/ModelDefinition
Annotations	An observations model.
Diagram	
Type	mdef:ObservationModelType

1.2.36 Element `mdef:Standard`

Namespace	http://www.pharmml.org/2013/03/ModelDefinition
Annotations	Defines standard error model.
Diagram	
Type	mdef:GaussianObsError

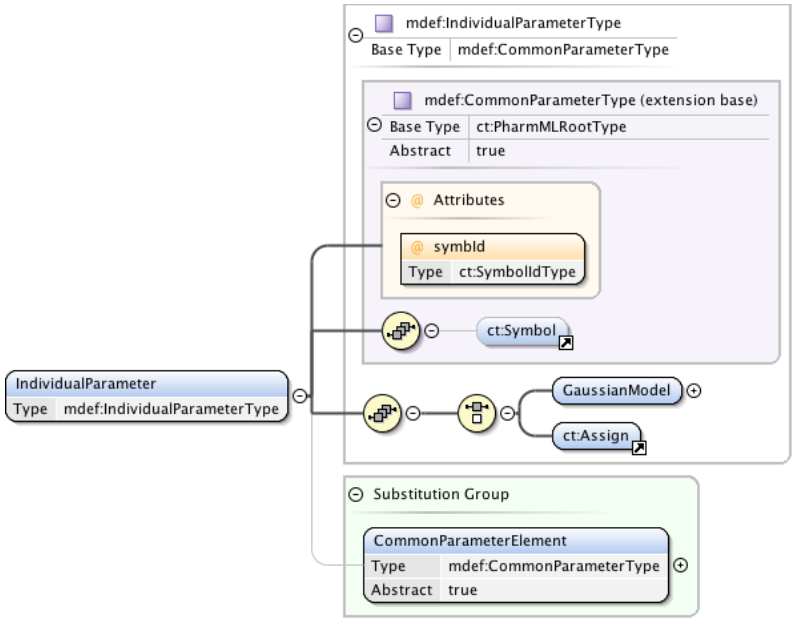
Substitution Group Affiliation	• mdef:ObservationError

1.2.37 Element `mdef:General`

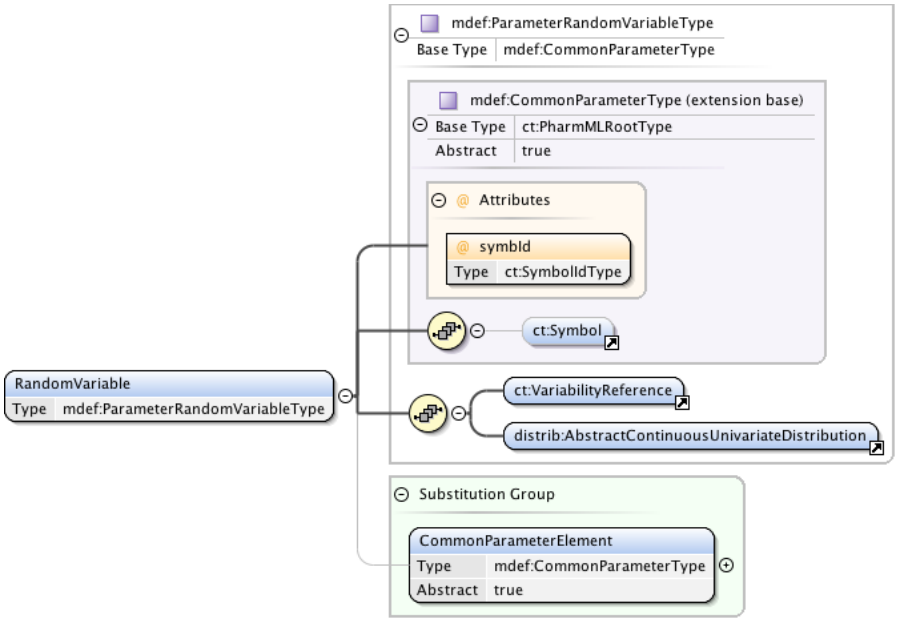
Namespace	http://www.pharmml.org/2013/03/ModelDefinition
Annotations	Defines general error model.
Diagram	
Type	mdef:GeneralObsError
Substitution Group Affiliation	• mdef:ObservationError

1.2.38 Element `mdef:IndividualParameter`

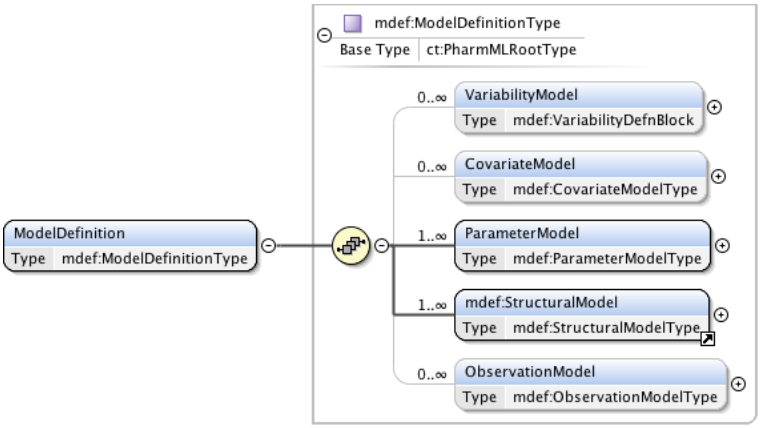
Namespace	http://www.pharmml.org/2013/03/ModelDefinition
Annotations	Defines an individual parameter.

Diagram	
Type	mdef:IndividualParameterType
Substitution Group Affiliation	• mdef:CommonParameterElement

1.2.39 Element `mdef:RandomVariable`

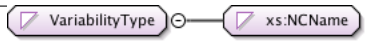
Namespace	http://www.pharmml.org/2013/03/ModelDefinition
Annotations	Defines a random variable.
Diagram	
Type	mdef:ParameterRandomVariableType
Substitution Group Affiliation	• mdef:CommonParameterElement

1.2.40 Element `mdef:ModelDefinition`

Namespace	http://www.pharmml.org/2013/03/ModelDefinition
Annotations	This is the top element defining the defintion of the pharmacometric model. This contains the variability model, covariate model, parameter model, structural model and observations model.
Diagram	
Type	mdef:ModelDefinitionType

1.3 Simple Type(s)

1.3.1 Simple Type `mdef:VariabilityType`

Namespace	http://www.pharmml.org/2013/03/ModelDefinition
Annotations	The type defining the type of variability of the variability model.
Diagram	
Type	restriction of xs:NCName
Facets	enumeration `error` — Residual error variability. enumeration `model` — Model variability.

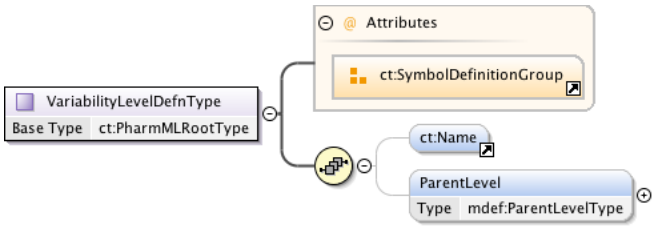
1.3.2 Simple Type `mdef:LhsTransformationType`

Namespace	http://www.pharmml.org/2013/03/ModelDefinition
Annotations	A type defining possible transformation functions that may be applied.
Diagram	
Type	restriction of xs:token
Facets	enumeration `log` — Natural log transformation. enumeration `logit` — Logit transformation. enumeration `probit` — Probit transformation.

1.4 Complex Type(s)

1.4.1 Complex Type `mdef:VariabilityLevelDefnType`

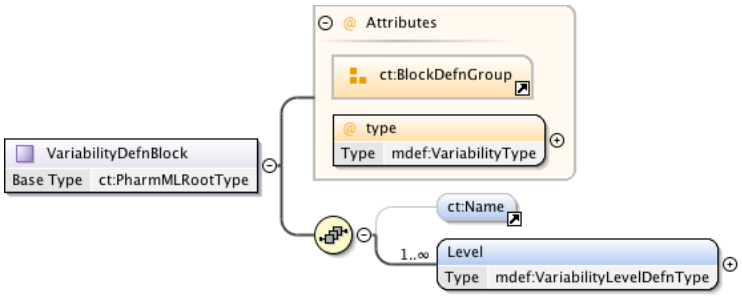
Namespace	http://www.pharmml.org/2013/03/ModelDefinition
Annotations	Defines the variability level.

Diagram	
Type	extension of PharmMLRootType

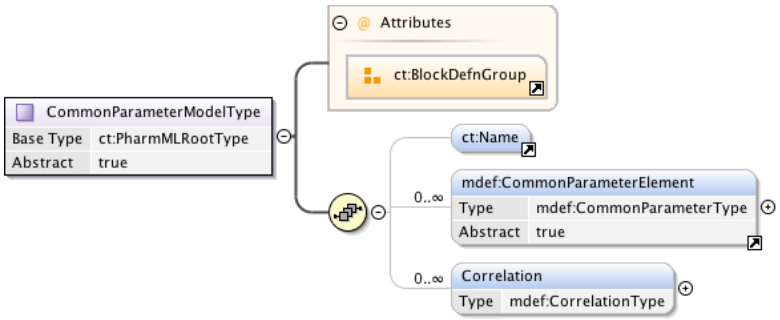
1.4.2 Complex Type `mdef:ParentLevelType`

Namespace	http://www.pharmml.org/2013/03/ModelDefinition
Annotations	Parent level type.
Diagram	

1.4.3 Complex Type `mdef:VariabilityDefnBlock`

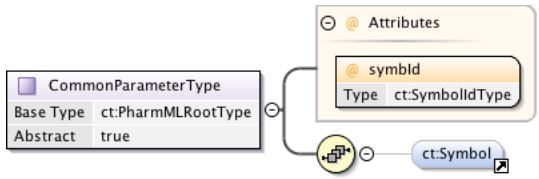
Namespace	http://www.pharmml.org/2013/03/ModelDefinition
Annotations	Type defining a block defining a variability model.
Diagram	
Type	extension of PharmMLRootType

1.4.4 Complex Type `mdef:CommonParameterModelType`

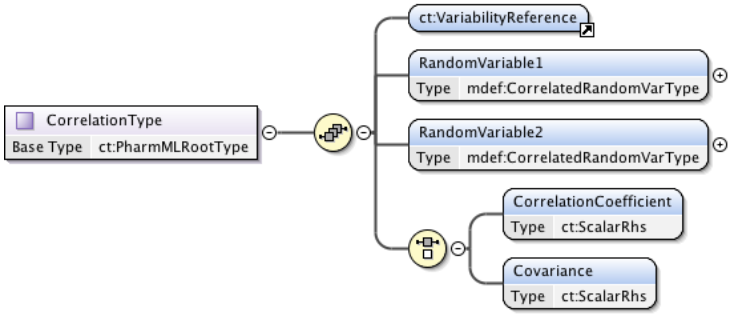
Namespace	http://www.pharmml.org/2013/03/ModelDefinition
Annotations	Abstract type defining common parameter model.
Diagram	
Type	extension of PharmMLRootType

1.4.5 Complex Type `mdef:CommonParameterType`

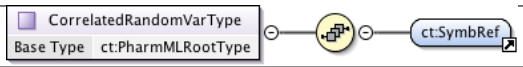
Namespace	http://www.pharmml.org/2013/03/ModelDefinition

Annotations	Abstract type defining the common properties of a parameter definition.
Diagram	
Type	extension of PharmMLRootType

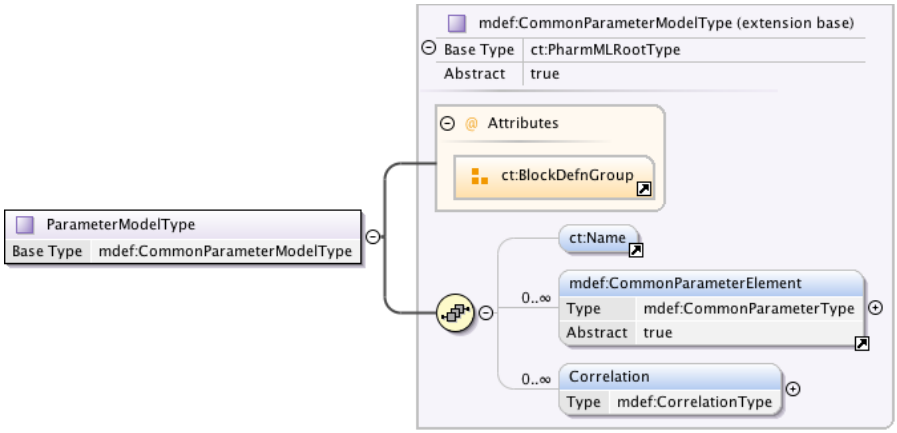
1.4.6 Complex Type `mdef:CorrelationType`

Namespace	http://www.pharmml.org/2013/03/ModelDefinition
Annotations	Type defining a correlation between random effects.
Diagram	
Type	extension of PharmMLRootType

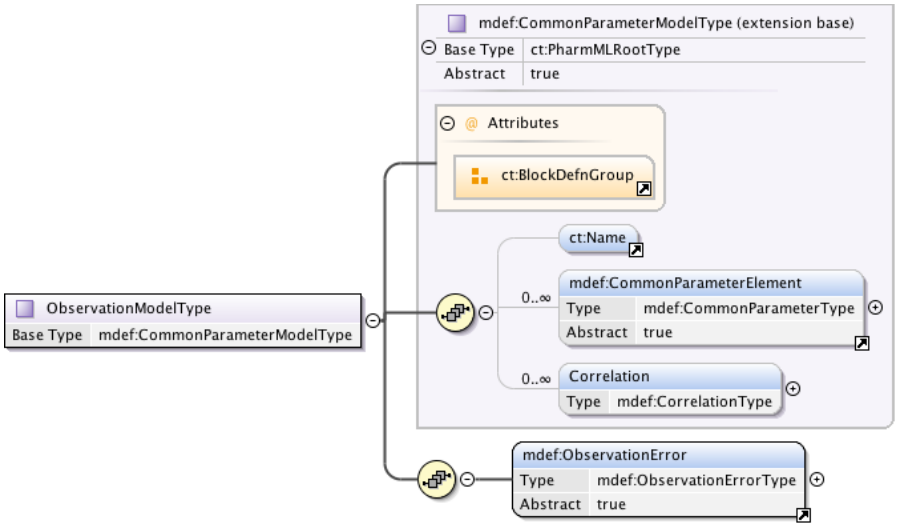
1.4.7 Complex Type `mdef:CorrelatedRandomVarType`

Namespace	http://www.pharmml.org/2013/03/ModelDefinition
Annotations	Type defining a correlated random variable.
Diagram	
Type	extension of PharmMLRootType

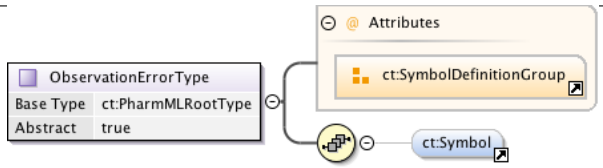
1.4.8 Complex Type `mdef:ParameterModelType`

Namespace	http://www.pharmml.org/2013/03/ModelDefinition
Annotations	The type defining a parameter model.
Diagram	
Type	extension of mdef:CommonParameterModelType

1.4.9 Complex Type `mdef:ObservationModelType`

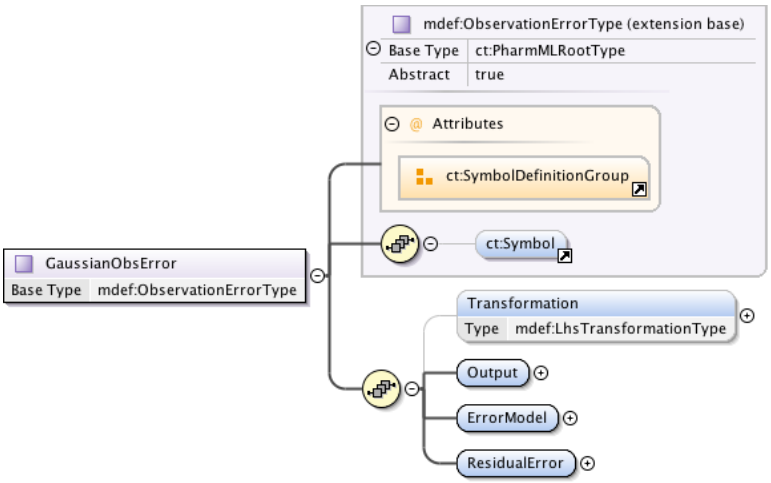
Namespace	http://www.pharmml.org/2013/03/ModelDefinition
Annotations	Type defining the observation model.
Diagram	
Type	extension of mdef:CommonParameterModelType

1.4.10 Complex Type `mdef:ObservationErrorType`

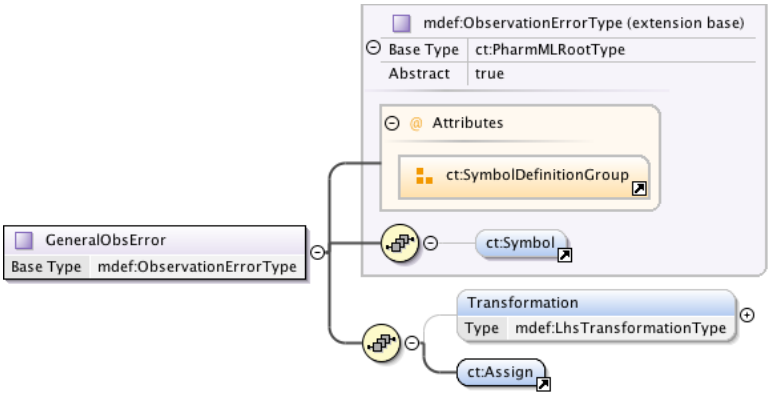
Namespace	http://www.pharmml.org/2013/03/ModelDefinition
Annotations	Base observation error type. This defines the name of the variable assigned with the result of the residual error.
Diagram	
Type	extension of PharmMLRootType

1.4.11 Complex Type `mdef:GaussianObsError`

Namespace	http://www.pharmml.org/2013/03/ModelDefinition
Annotations	Gaussian residual error definition. Definition is of the form: y = f + g * eps

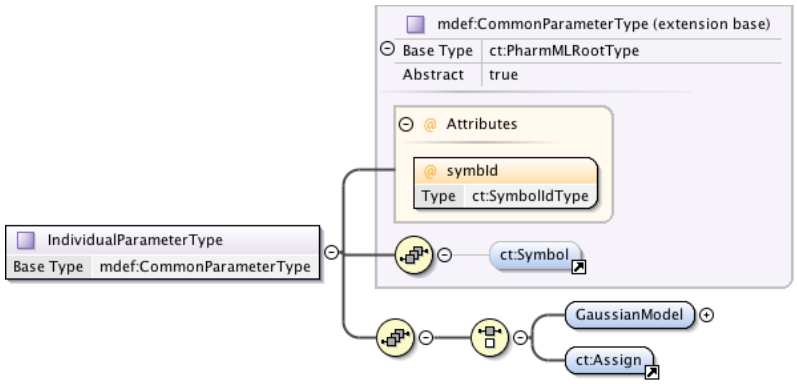
Diagram	
Type	extension of mdef:ObservationErrorType

1.4.12 Complex Type `mdef:GeneralObsError`

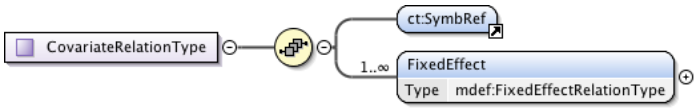
Namespace	http://www.pharmml.org/2013/03/ModelDefinition
Annotations	A general form of the residual error, where the error is unstructured and explicit.
Diagram	
Type	extension of mdef:ObservationErrorType

1.4.13 Complex Type `mdef:IndividualParameterType`

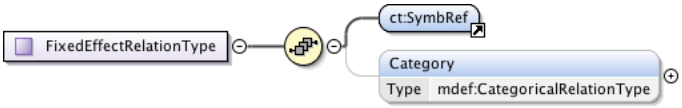
Namespace	http://www.pharmml.org/2013/03/ModelDefinition
Annotations	Describes an individual parameter. Three encodings of a parameter model are available: Type 1. explicit equation type of parameter model psi_i = H(beta, c_i, eta_i) Type 2. Gaussian model with general covariate model h(psi_i) = H(beta, c_i) + eta_i Type 3. Gaussian model with linear covariate model h(psi_i) = h(psi_pop) + beta c_i + eta_i

Diagram	
Type	extension of mdef:CommonParameterType

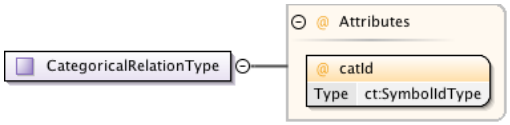
1.4.14 Complex Type `mdef:CovariateRelationType`

Namespace	http://www.pharmml.org/2013/03/ModelDefinition
Annotations	Type defining the relationship between the covariate and a fixed effect parameter. Typically this defines a linear relationships.
Diagram	

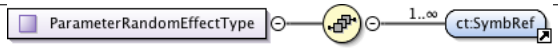
1.4.15 Complex Type `mdef:FixedEffectRelationType`

Namespace	http://www.pharmml.org/2013/03/ModelDefinition
Annotations	Defines a fixed effect.
Diagram	

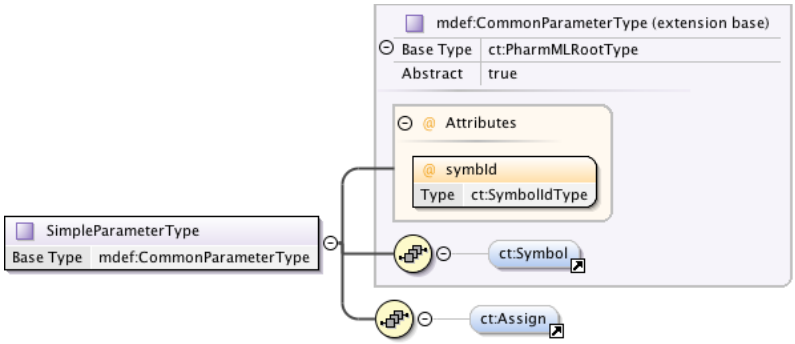
1.4.16 Complex Type `mdef:CategoricalRelationType`

Namespace	http://www.pharmml.org/2013/03/ModelDefinition
Annotations	Type specifying a relationship to a specific category value in a categorical covariate.
Diagram	
Attributes	**QName** / **Type** / **Use**: **catId** / SymbolIdType / required Specifies the category value of the covariate to which this relationship applies. For example if a covariate is sex then the Female category may be specified as catId="F".

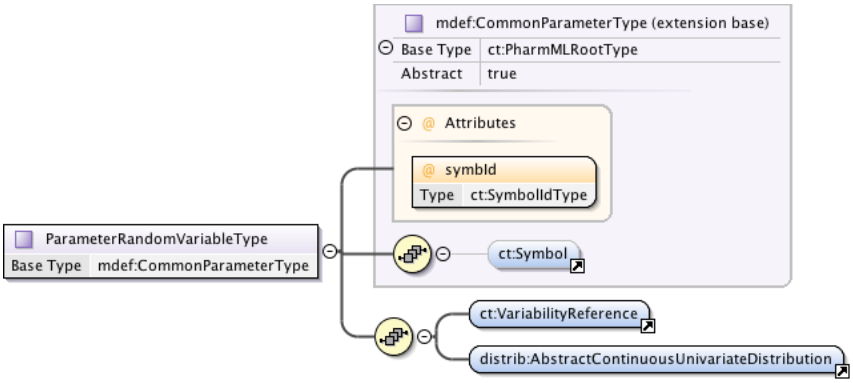
1.4.17 Complex Type `mdef:ParameterRandomEffectType`

Namespace	http://www.pharmml.org/2013/03/ModelDefinition
Annotations	Type defining a Random effect using a reference to a Random variable.
Diagram	

1.4.18 Complex Type `mdef:SimpleParameterType`

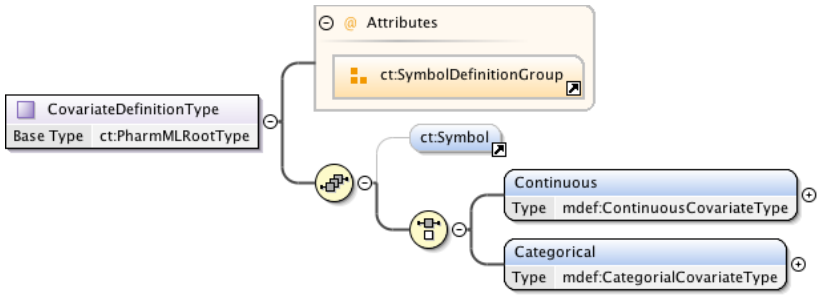
Namespace	http://www.pharmml.org/2013/03/ModelDefinition
Annotations	This defines a parameter that does not contain any random effects. Once initialised its value will not change over time. The parameter is of type real.
Diagram	
Type	extension of mdef:CommonParameterType

1.4.19 Complex Type `mdef:ParameterRandomVariableType`

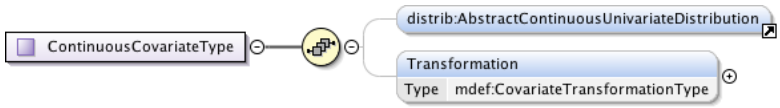
Namespace	http://www.pharmml.org/2013/03/ModelDefinition
Annotations	Type specifies a random variable definition.
Diagram	
Type	extension of mdef:CommonParameterType

1.4.20 Complex Type `mdef:CovariateDefinitionType`

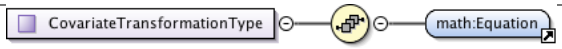
Namespace	http://www.pharmml.org/2013/03/ModelDefinition
Annotations	Type that specifies a covariate definition.

Diagram	
Type	extension of PharmMLRootType

1.4.21 Complex Type `mdef:ContinuousCovariateType`

Namespace	http://www.pharmml.org/2013/03/ModelDefinition
Annotations	Type defines a continuous covariate.
Diagram	

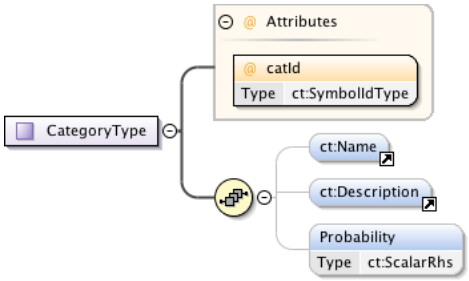
1.4.22 Complex Type `mdef:CovariateTransformationType`

Namespace	http://www.pharmml.org/2013/03/ModelDefinition
Annotations	Type defines how the covariate is transformed when used.
Diagram	

1.4.23 Complex Type `mdef:CategorialCovariateType`

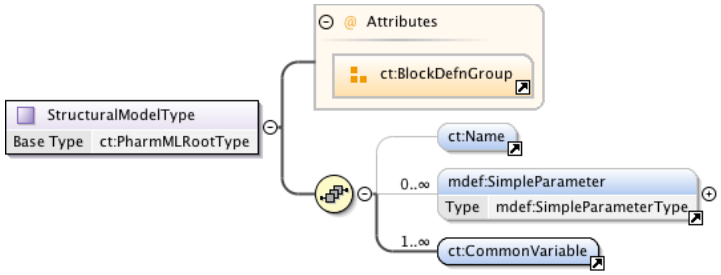
Namespace	http://www.pharmml.org/2013/03/ModelDefinition
Annotations	Type defines a categorical covariate.
Diagram	

1.4.24 Complex Type `mdef:CategoryType`

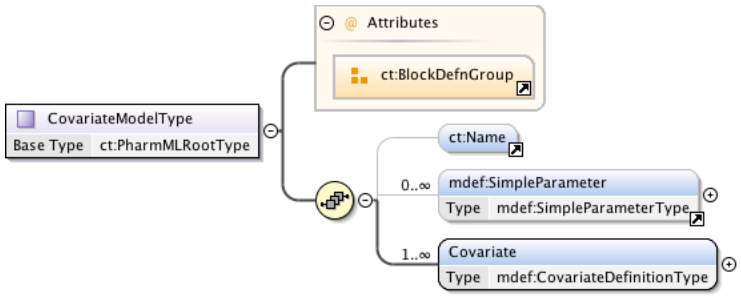
Namespace	http://www.pharmml.org/2013/03/ModelDefinition
Annotations	Type defines a category in a categorical covariate.
Diagram	

Attributes	QName	Type	Use			
	catId	SymbolIdType	required			
		The identifier of the category.				

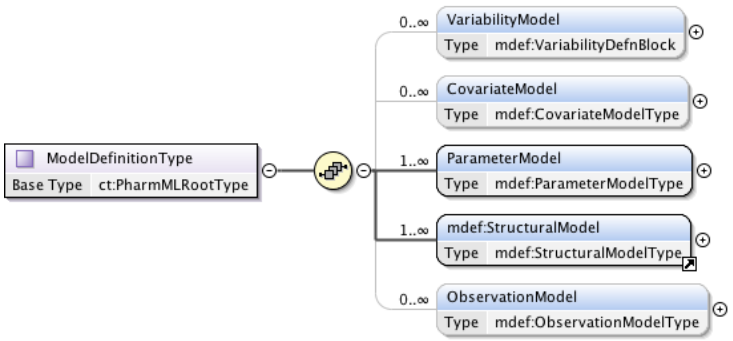
1.4.25 Complex Type `mdef:StructuralModelType`

Namespace	http://www.pharmml.org/2013/03/ModelDefinition
Annotations	Type that specifies a structural model.
Diagram	
Type	extension of PharmMLRootType

1.4.26 Complex Type `mdef:CovariateModelType`

Namespace	http://www.pharmml.org/2013/03/ModelDefinition
Annotations	A type defining a covariate model.
Diagram	
Type	extension of PharmMLRootType

1.4.27 Complex Type `mdef:ModelDefinitionType`

Namespace	http://www.pharmml.org/2013/03/ModelDefinition
Annotations	Type that specifies the model definition section of the PharmML document.
Diagram	

Type	extension of PharmMLRootType

2 Namespace: ""

2.1 Attribute(s)

2.1.1 Attribute `mdef:VariabilityDefnBlock /@type`

Namespace	No namespace		
Annotations	Defines the type of the variability model.		
Type	mdef:VariabilityType		
Facets	enumeration	`error`	Residual error variability.
	enumeration	`model`	Model variability.

2.1.2 Attribute `mdef:CommonParameterType /@symbId`

Namespace	No namespace
Annotations	The symbol id for this parameter.
Type	SymbolIdType

2.1.3 Attribute `mdef:CategoricalRelationType /@catId`

Namespace	No namespace
Annotations	Specifies the category value of the covariate to which this relationship applies. For example if a covariate is sex then the Female category may be specified as catId="F".
Type	SymbolIdType

2.1.4 Attribute `mdef:CategoryType /@catId`

Namespace	No namespace
Annotations	The identifier of the category.
Type	SymbolIdType

11.3 Trial Design

1 Namespace: "http://www.pharmml.org/2013/03/TrialDesign"

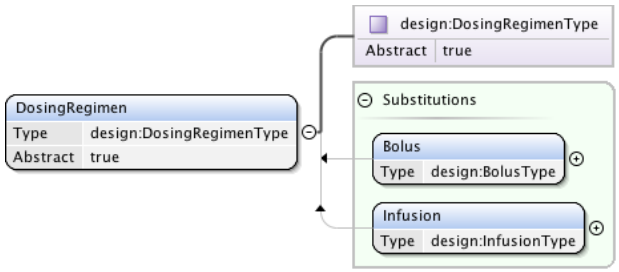
1.1 Schema(s)

1.1.1 Main schema `trialDesign.xsd`

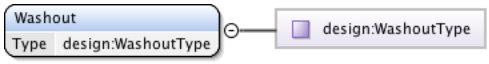
Namespace	http://www.pharmml.org/2013/03/TrialDesign
Annotations	Defines trial design section of PharmML.

1.2 Element(s)

1.2.1 Element `design:DosingRegimen`

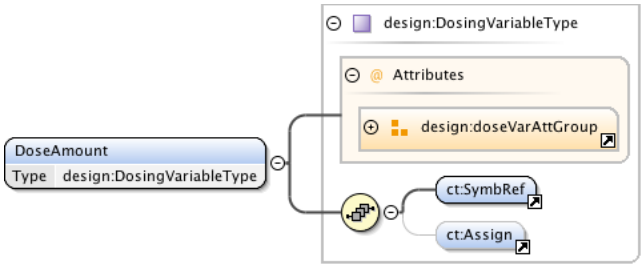
Namespace	http://www.pharmml.org/2013/03/TrialDesign
Annotations	Specifies dosing regimen.
Diagram	
Type	design:DosingRegimenType
Substitution Group	• design:Bolus • design:Infusion

1.2.2 Element `design:Washout`

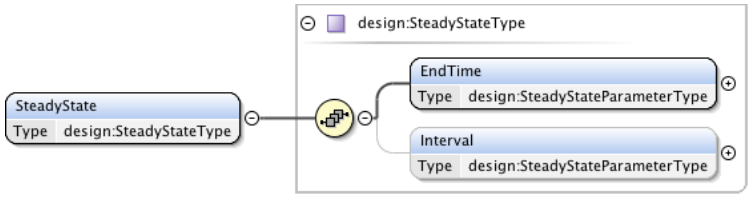
Namespace	http://www.pharmml.org/2013/03/TrialDesign
Annotations	Specifies a washout epoch, where variables are reset to their initial values.
Diagram	
Type	design:WashoutType

1.2.3 Element `design:BolusType /design:DoseAmount`

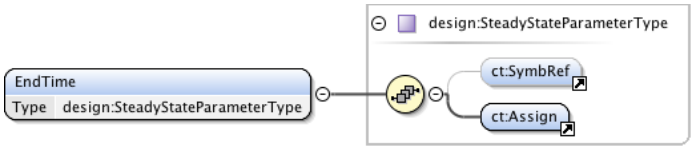
Namespace	http://www.pharmml.org/2013/03/TrialDesign
Annotations	Dosing information.

Diagram	
Type	design:DosingVariableType
Attributes	**QName** **Type** **Use** **inputType** design:DoseInputTypeType required Specifies the type of dosing. Is it assigned to a variable or is it an input to a system of ODEs.

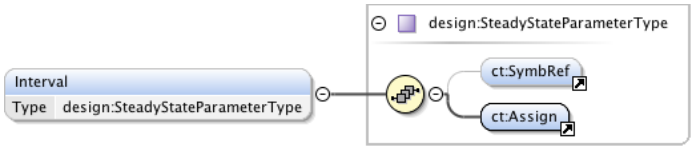
1.2.4 Element `design:BolusType /design:SteadyState`

Namespace	http://www.pharmml.org/2013/03/TrialDesign
Annotations	Steady state bolus dosing.
Diagram	
Type	design:SteadyStateType

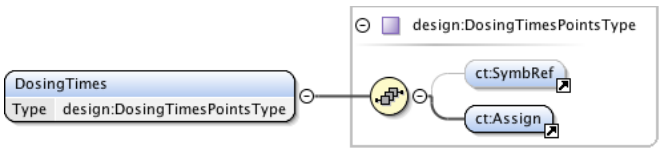
1.2.5 Element `design:SteadyStateType /design:EndTime`

Namespace	http://www.pharmml.org/2013/03/TrialDesign
Annotations	The last dosing time.
Diagram	
Type	design:SteadyStateParameterType

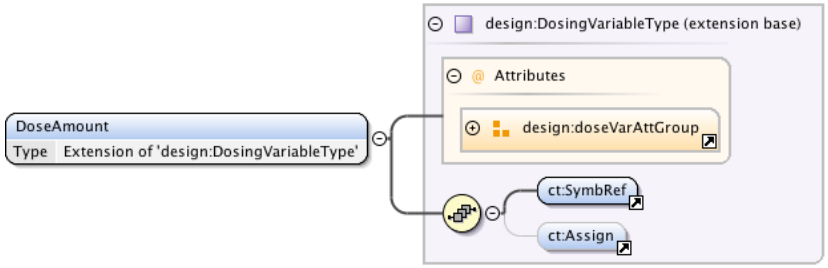
1.2.6 Element `design:SteadyStateType /design:Interval`

Namespace	http://www.pharmml.org/2013/03/TrialDesign
Annotations	The dosing period.
Diagram	
Type	design:SteadyStateParameterType

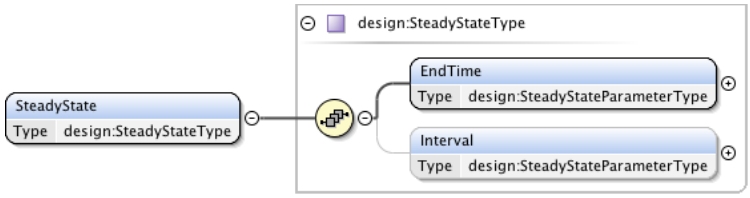
1.2.7 Element `design:BolusType /design:DosingTimes`

Namespace	http://www.pharmml.org/2013/03/TrialDesign
Annotations	The dosing times.
Diagram	
Type	design:DosingTimesPointsType

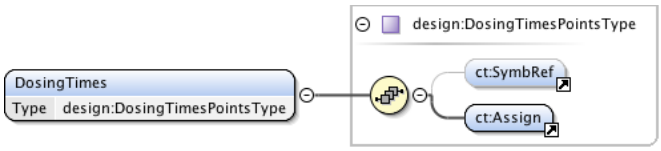
1.2.8 Element `design:InfusionType /design:DoseAmount`

Namespace	http://www.pharmml.org/2013/03/TrialDesign
Annotations	Dosing information.
Diagram	
Type	extension of design:DosingVariableType
Attributes	**QName** / **Type** / **Use**: **inputType** — design:DoseInputTypeType — required Specifies the type of dosing. Is it assigned to a variable or is it an input to a system of ODEs.

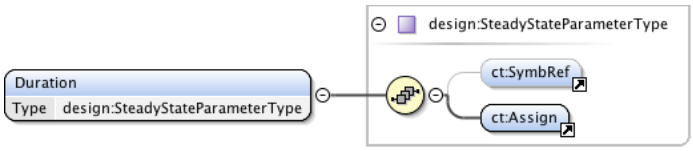
1.2.9 Element `design:InfusionType /design:SteadyState`

Namespace	http://www.pharmml.org/2013/03/TrialDesign
Annotations	Steady state infusion dosing.
Diagram	
Type	design:SteadyStateType

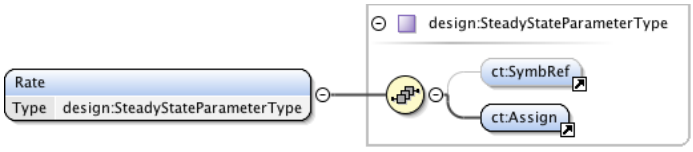
1.2.10 Element `design:InfusionType /design:DosingTimes`

Namespace	http://www.pharmml.org/2013/03/TrialDesign
Annotations	The dosing times.
Diagram	
Type	design:DosingTimesPointsType

1.2.11 Element `design:InfusionType /design:Duration`

Namespace	http://www.pharmml.org/2013/03/TrialDesign
Annotations	The duration of the infusion.
Diagram	
Type	design:SteadyStateParameterType

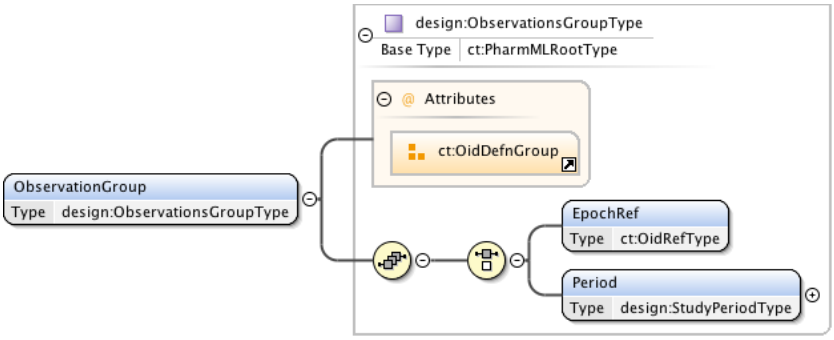
1.2.12 Element `design:InfusionType /design:Rate`

Namespace	http://www.pharmml.org/2013/03/TrialDesign
Diagram	
Type	design:SteadyStateParameterType

1.2.13 Element `design:StudyEventType /design:ArmRef`

Namespace	http://www.pharmml.org/2013/03/TrialDesign
Annotations	In effect defines the population of subjects that this event happens to.
Diagram	
Type	OidRefType

1.2.14 Element `design:ObservationsType /design:ObservationGroup`

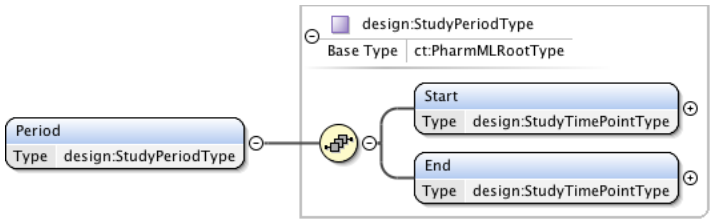
Namespace	http://www.pharmml.org/2013/03/TrialDesign
Diagram	
Type	design:ObservationsGroupType

1.2.15 Element `design:ObservationsGroupType /design:EpochRef`

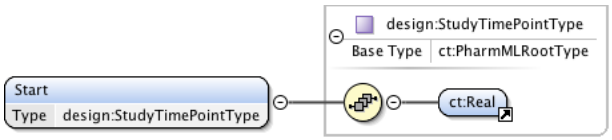
Namespace	http://www.pharmml.org/2013/03/TrialDesign
Annotations	Refers to the epoch during which this group of observations occurred.

Diagram	
Type	OidRefType

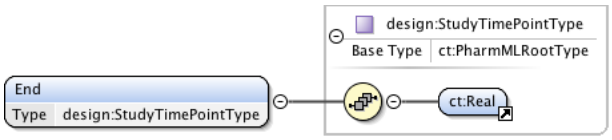
1.2.16 Element `design:ObservationsGroupType /design:Period`

Namespace	http://www.pharmml.org/2013/03/TrialDesign
Annotations	Defines the time period within the study during which this group of observations occurred.
Diagram	
Type	design:StudyPeriodType

1.2.17 Element `design:StudyPeriodType /design:Start`

Namespace	http://www.pharmml.org/2013/03/TrialDesign
Annotations	The start time of the period.
Diagram	
Type	design:StudyTimePointType

1.2.18 Element `design:StudyPeriodType /design:End`

Namespace	http://www.pharmml.org/2013/03/TrialDesign
Annotations	The end time of the period.
Diagram	
Type	design:StudyTimePointType

1.2.19 Element `design:Order`

Namespace	http://www.pharmml.org/2013/03/TrialDesign
Annotations	Specifies the order of elements in the trial structure.
Diagram	
Type	IntValueType

1.2.20 Element design:CellDefnType /design:EpochRef

Namespace	http://www.pharmml.org/2013/03/TrialDesign
Annotations	Refers to the epoch in which this cell occurrs.
Diagram	
Type	OidRefType

1.2.21 Element design:CellDefnType /design:ArmRef

Namespace	http://www.pharmml.org/2013/03/TrialDesign
Annotations	Refers to the arm in which this cell occurs.
Diagram	
Type	OidRefType

1.2.22 Element design:CellDefnType /design:SegmentRef

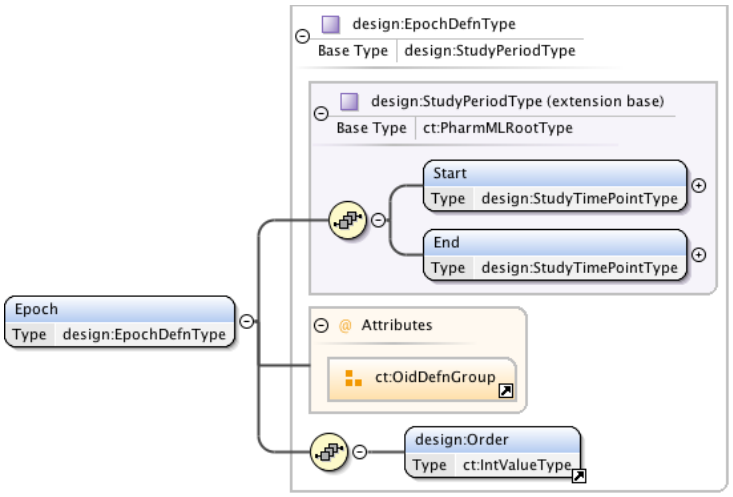
Namespace	http://www.pharmml.org/2013/03/TrialDesign
Annotations	Refers to a Segment which is used in this Cell (segments can be referred to by more than one cell.
Diagram	
Type	OidRefType

1.2.23 Element design:SegmentDefnType /design:ActivityRef

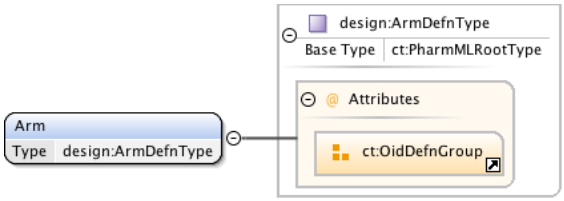
Namespace	http://www.pharmml.org/2013/03/TrialDesign
Annotations	Refers to an activity carried out in the segment of the study.
Diagram	
Type	OidRefType

1.2.24 Element design:TrialStructureType /design:Epoch

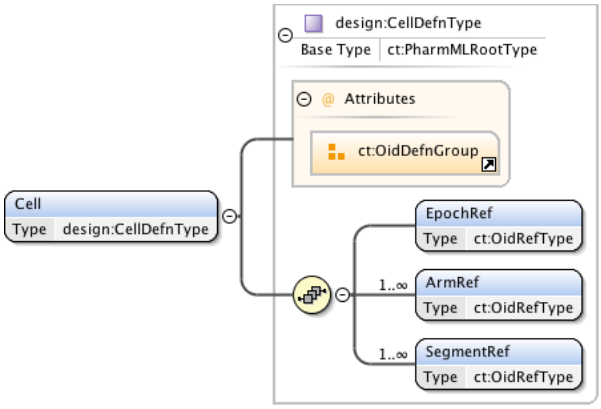
Namespace	http://www.pharmml.org/2013/03/TrialDesign
Annotations	Defines an epoch in the study.

Diagram	
Type	design:EpochDefnType

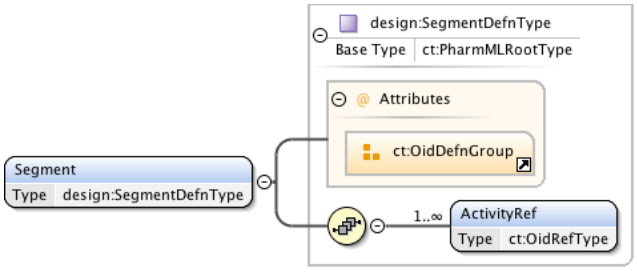
1.2.25 Element `design:TrialStructureType /design:Arm`

Namespace	http://www.pharmml.org/2013/03/TrialDesign
Annotations	Defines an arm of the study.
Diagram	
Type	design:ArmDefnType

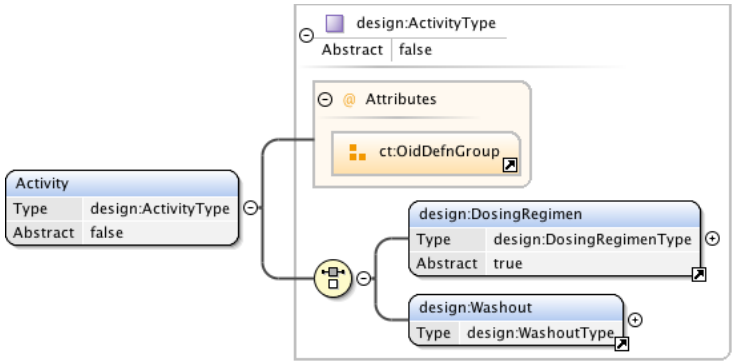
1.2.26 Element `design:TrialStructureType /design:Cell`

Namespace	http://www.pharmml.org/2013/03/TrialDesign
Annotations	Defines a cell in the study.
Diagram	
Type	design:CellDefnType

1.2.27 Element `design:TrialStructureType /design:Segment`

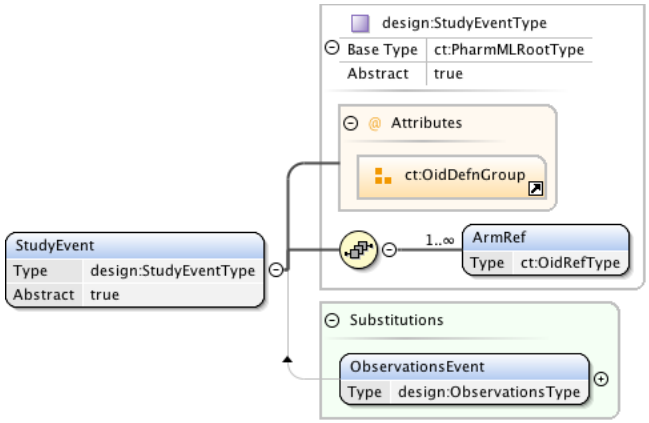
Namespace	http://www.pharmml.org/2013/03/TrialDesign
Annotations	Defines a segment of the study.
Diagram	
Type	design:SegmentDefnType

1.2.28 Element `design:Activity`

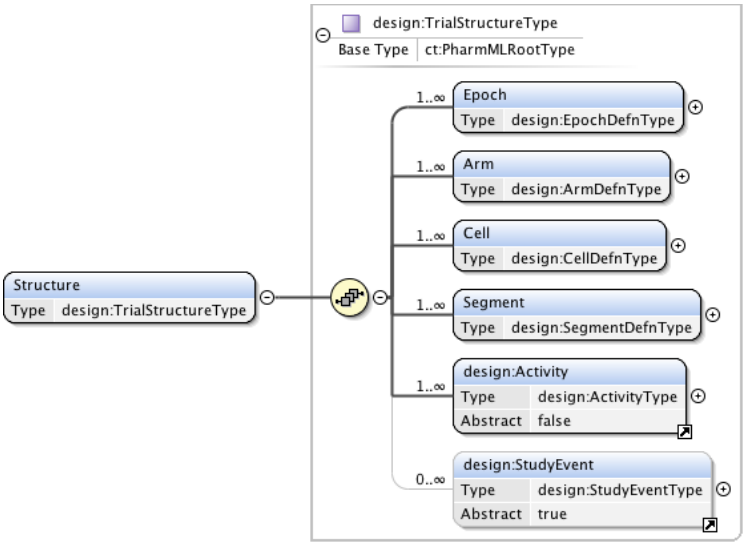
Namespace	http://www.pharmml.org/2013/03/TrialDesign
Annotations	Specifies an activity in the trial structure.
Diagram	
Type	design:ActivityType

1.2.29 Element `design:StudyEvent`

Namespace	http://www.pharmml.org/2013/03/TrialDesign
Annotations	An abstract element for all study events. There is only one element inheriting from this so this be superfluous in the future.

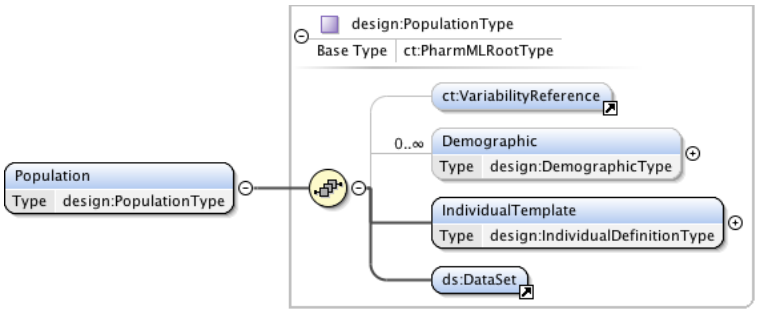
Diagram	
Type	design:StudyEventType
Substitution Group	• design:ObservationsEvent

1.2.30 Element `design:TrialDesignType /design:Structure`

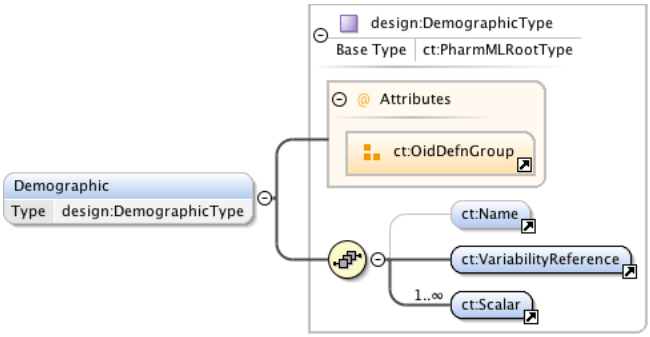
Namespace	http://www.pharmml.org/2013/03/TrialDesign
Annotations	Defines the structure of the study.
Diagram	
Type	design:TrialStructureType

1.2.31 Element `design:TrialDesignType /design:Population`

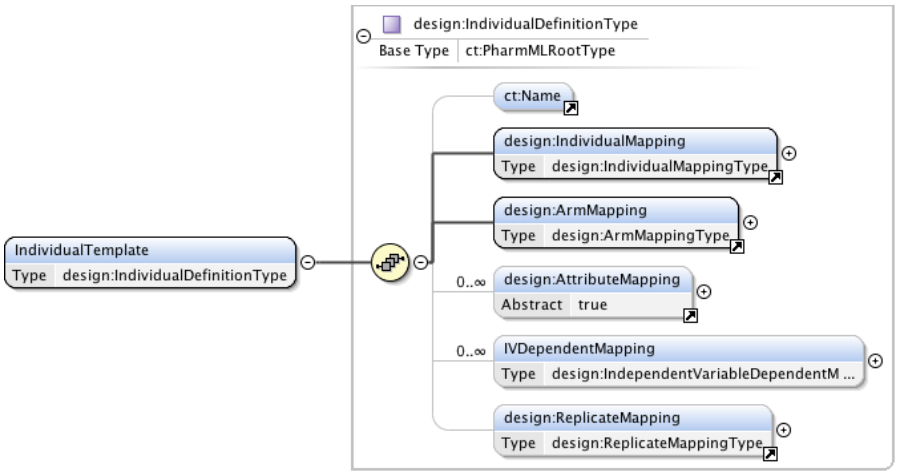
Namespace	http://www.pharmml.org/2013/03/TrialDesign
Annotations	Defines the population of the study.

Diagram	
Type	design:PopulationType

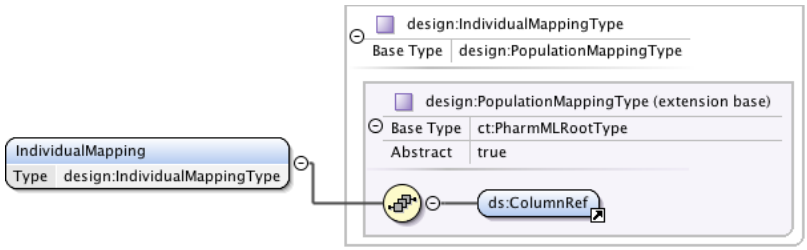
1.2.32 Element `design:PopulationType /design:Demographic`

Namespace	http://www.pharmml.org/2013/03/TrialDesign
Annotations	Defines a property of the individuals in the study that is not a covariate, but is important in the design of the study.
Diagram	
Type	design:DemographicType

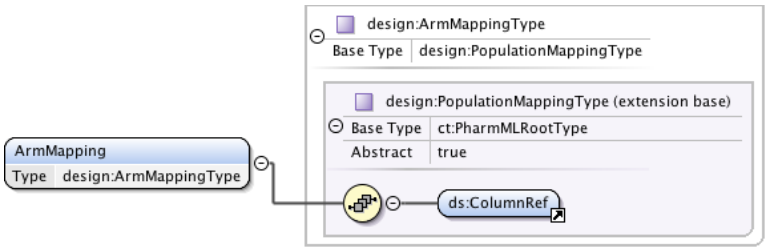
1.2.33 Element `design:PopulationType /design:IndividualTemplate`

Namespace	http://www.pharmml.org/2013/03/TrialDesign
Annotations	Defines a template of information about the subjects in the study.
Diagram	
Type	design:IndividualDefinitionType

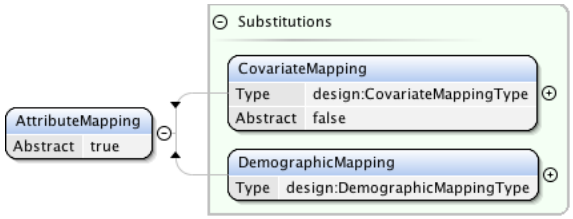
1.2.34 Element `design:IndividualMapping`

Namespace	http://www.pharmml.org/2013/03/TrialDesign
Annotations	Maps the individual to the dataset.
Diagram	
Type	design:IndividualMappingType

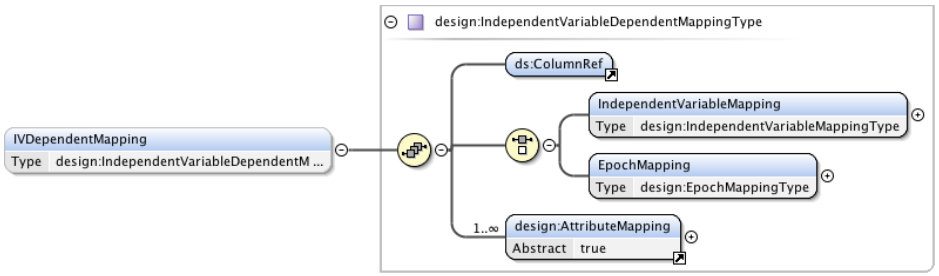
1.2.35 Element `design:ArmMapping`

Namespace	http://www.pharmml.org/2013/03/TrialDesign
Annotations	Maps the arm to the dataset.
Diagram	
Type	design:ArmMappingType

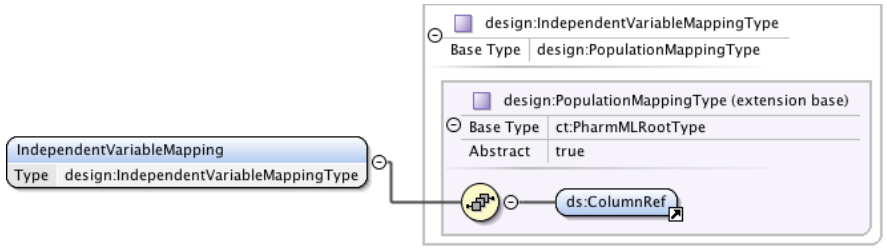
1.2.36 Element `design:AttributeMapping`

Namespace	http://www.pharmml.org/2013/03/TrialDesign
Annotations	Maps attributes to the dataset.
Diagram	
Substitution Group	• design:CovariateMapping • design:DemographicMapping

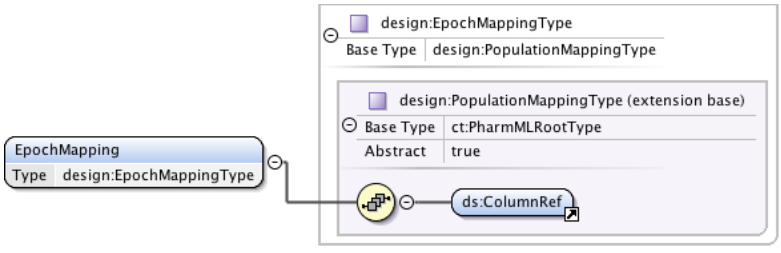
1.2.37 Element `design:IndividualDefinitionType /design:IVDependentMapping`

Namespace	http://www.pharmml.org/2013/03/TrialDesign
Annotations	Defines the indepedent (usually time) variable dependent mapping to the dataset.
Diagram	
Type	design:IndependentVariableDependentMappingType

1.2.38 Element `design:IndependentVariableDependentMappingType /design:IndependentVariableMapping`

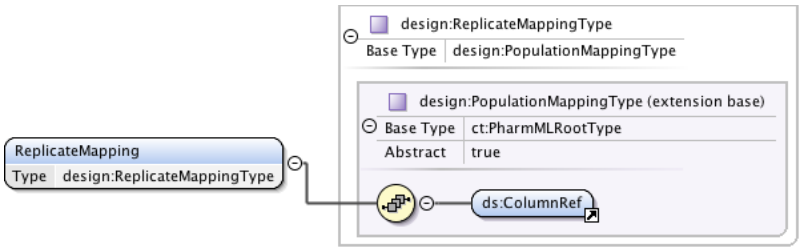
Namespace	http://www.pharmml.org/2013/03/TrialDesign
Annotations	Maps the independent variable (most often time) to a column in the dataset.
Diagram	
Type	design:IndependentVariableMappingType

1.2.39 Element `design:IndependentVariableDependentMappingType /design:EpochMapping`

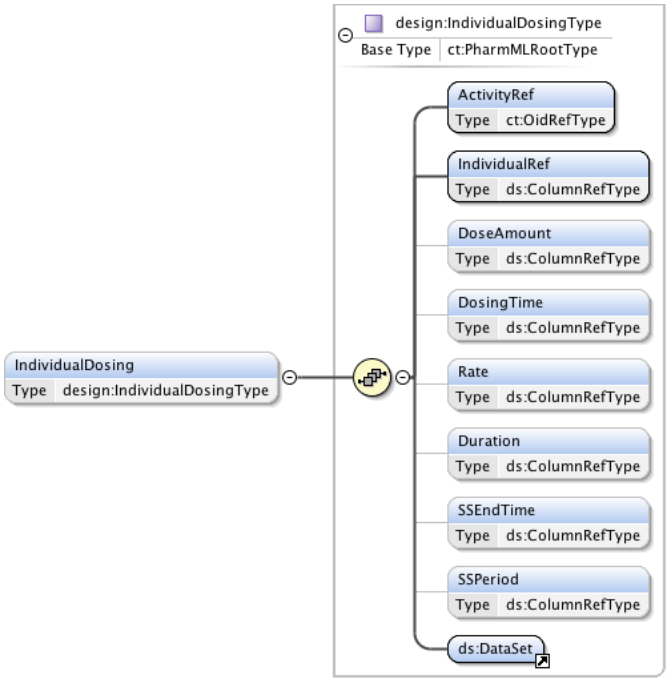
Namespace	http://www.pharmml.org/2013/03/TrialDesign
Annotations	Maps the epoch to a column in the dataset. The epoch defines the time period in which this property applies.
Diagram	
Type	design:EpochMappingType

1.2.40 Element `design:ReplicateMapping`

Namespace	http://www.pharmml.org/2013/03/TrialDesign
Annotations	Mapping to replicates in the dataset.

Diagram	
Type	design:ReplicateMappingType

1.2.41 Element `design:TrialDesignType /design:IndividualDosing`

Namespace	http://www.pharmml.org/2013/03/TrialDesign
Annotations	Provides time dependent dosing information for each individual in the study.
Diagram	
Type	design:IndividualDosingType

1.2.42 Element `design:IndividualDosingType /design:ActivityRef`

Namespace	http://www.pharmml.org/2013/03/TrialDesign
Annotations	Refers to the activity describing the dosing regimen being instantiated here.
Diagram	
Type	OidRefType

1.2.43 Element `design:IndividualDosingType /design:IndividualRef`

Namespace	http://www.pharmml.org/2013/03/TrialDesign
Annotations	Refers to the subject in the study that this dosing regimen is applied to.

Diagram	
Type	ColumnRefType

1.2.44 Element design:IndividualDosingType /design:DoseAmount

Namespace	http://www.pharmml.org/2013/03/TrialDesign
Annotations	Specifies the amount of dose.
Diagram	
Type	ColumnRefType

1.2.45 Element design:IndividualDosingType /design:DosingTime

Namespace	http://www.pharmml.org/2013/03/TrialDesign
Annotations	Specifies the dosing time.
Diagram	
Type	ColumnRefType

1.2.46 Element design:IndividualDosingType /design:Rate

Namespace	http://www.pharmml.org/2013/03/TrialDesign
Annotations	Specifies the rate of infusion.
Diagram	
Type	ColumnRefType

1.2.47 Element design:IndividualDosingType /design:Duration

Namespace	http://www.pharmml.org/2013/03/TrialDesign
Annotations	Specifies the duration of infusion.
Diagram	
Type	ColumnRefType

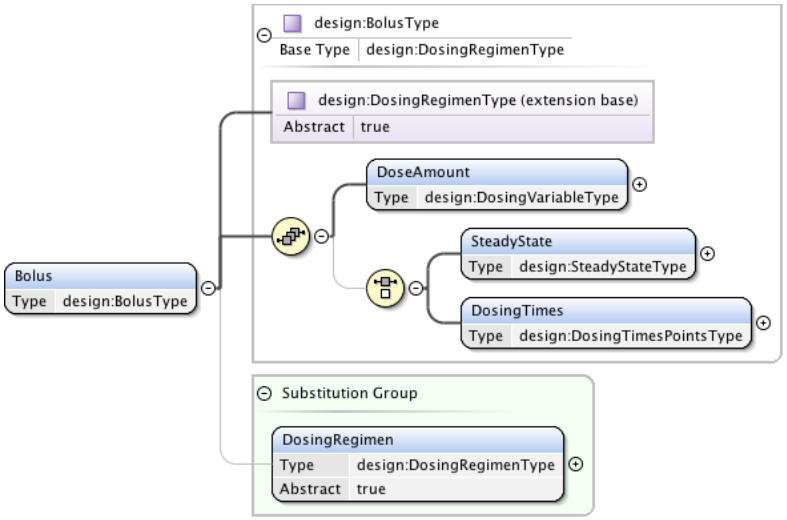
1.2.48 Element design:IndividualDosingType /design:SSEndTime

Namespace	http://www.pharmml.org/2013/03/TrialDesign
Annotations	Specifies the last dosing time in steady state dosing.
Diagram	
Type	ColumnRefType

1.2.49 Element design:IndividualDosingType /design:SSPeriod

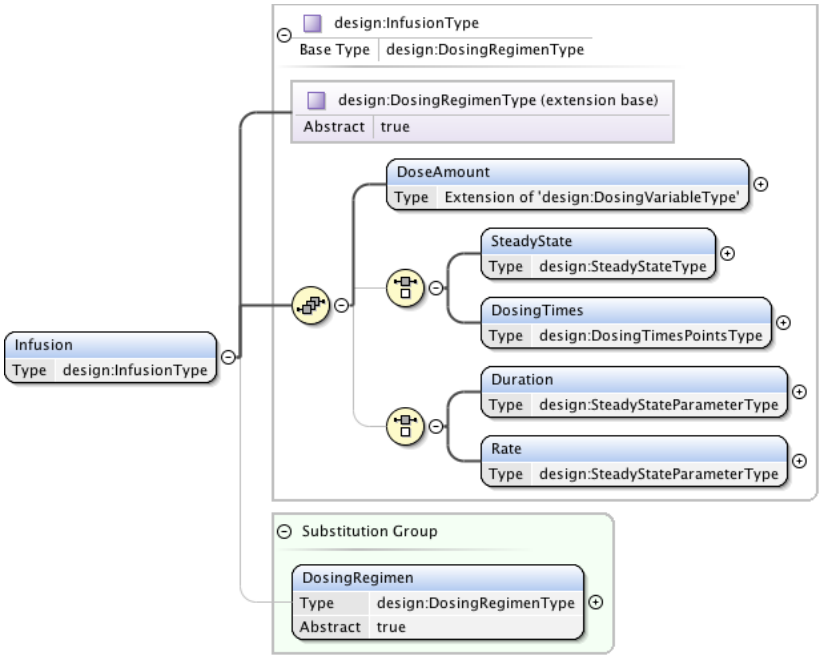
Namespace	http://www.pharmml.org/2013/03/TrialDesign
Annotations	Specifies the period in steady state dosing.
Diagram	
Type	ColumnRefType

1.2.50 Element `design:Bolus`

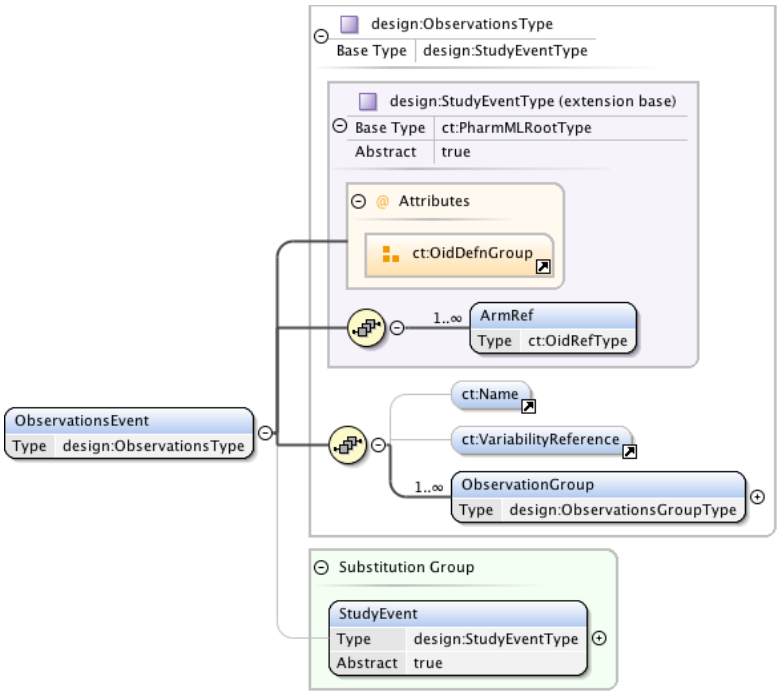
Namespace	http://www.pharmml.org/2013/03/TrialDesign
Annotations	Specifies a bolus dosing regiment.
Diagram	
Type	design:BolusType
Substitution Group Affiliation	• design:DosingRegimen

1.2.51 Element `design:Infusion`

Namespace	http://www.pharmml.org/2013/03/TrialDesign
Annotations	Specifies a infusion dosing regiment.

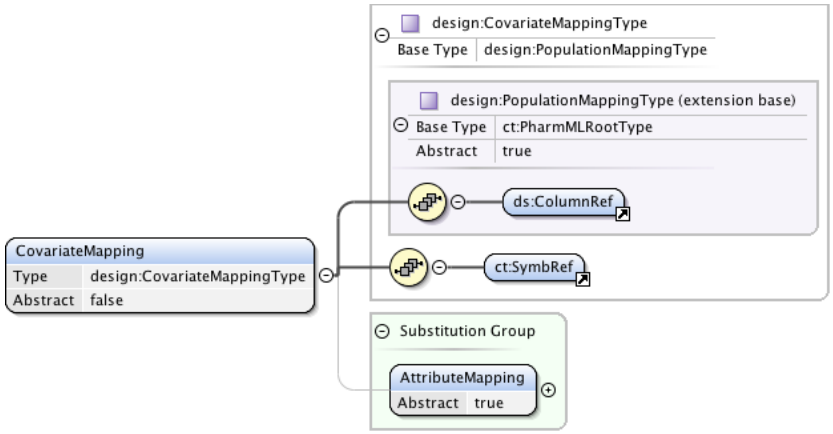
Diagram	
Type	design:InfusionType
Substitution Group Affiliation	• design:DosingRegimen

1.2.52 Element `design:ObservationsEvent`

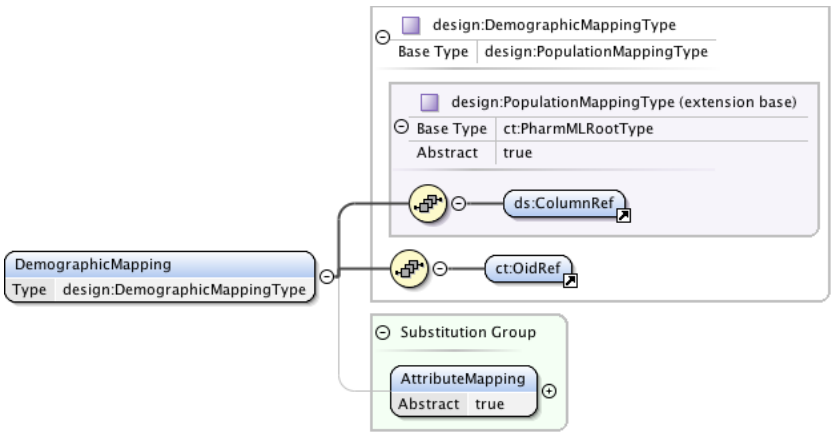
Namespace	http://www.pharmml.org/2013/03/TrialDesign
Annotations	Defines an observations event in the study,
Diagram	
Type	design:ObservationsType

Substitution Group Affiliation	• design:StudyEvent

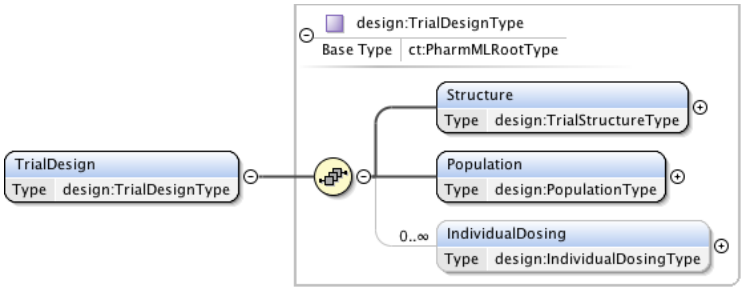
1.2.53 Element `design:CovariateMapping`

Namespace	http://www.pharmml.org/2013/03/TrialDesign
Annotations	Mapping to covariates in the dataset.
Diagram	
Type	design:CovariateMappingType
Substitution Group Affiliation	• design:AttributeMapping

1.2.54 Element `design:DemographicMapping`

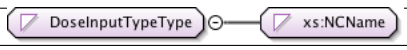
Namespace	http://www.pharmml.org/2013/03/TrialDesign
Annotations	Mapping to demographics in the dataset.
Diagram	
Type	design:DemographicMappingType
Substitution Group Affiliation	• design:AttributeMapping

1.2.55 Element `design:TrialDesign`

Namespace	http://www.pharmml.org/2013/03/TrialDesign
Annotations	Defines the trial design. Top level element of this schema.
Diagram	
Type	design:TrialDesignType

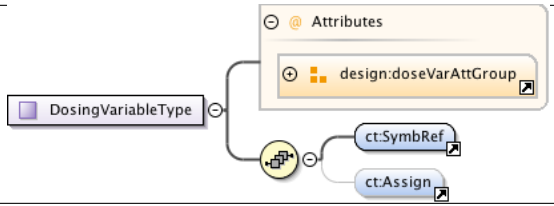
1.3 Simple Type(s)

1.3.1 Simple Type `design:DoseInputTypeType`

Namespace	http://www.pharmml.org/2013/03/TrialDesign
Annotations	Defines the dosing input type.
Diagram	
Type	restriction of xs:NCName
Facets	enumeration `dose` — Dose is assigned to a dosing variable. enumeration `target` — Dose is an input to a system of ODEs.

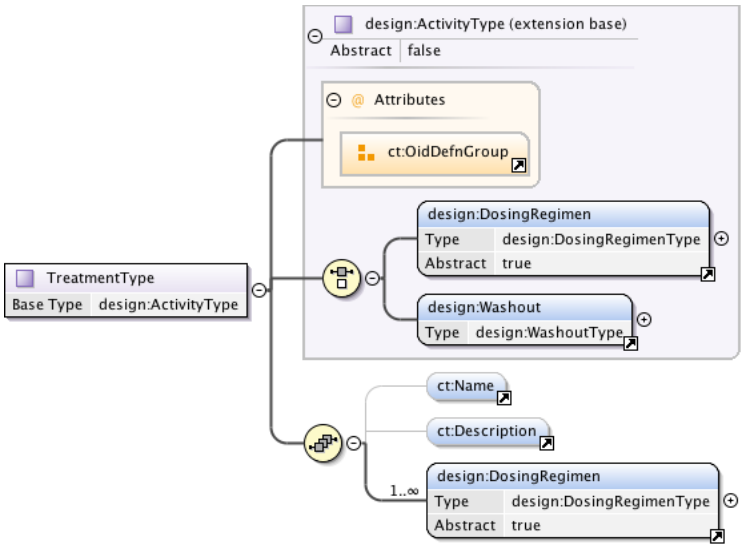
1.4 Complex Type(s)

1.4.1 Complex Type `design:DosingVariableType`

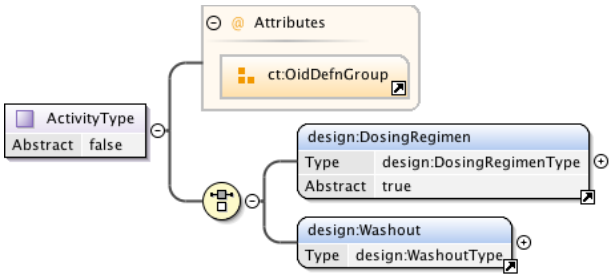
Namespace	http://www.pharmml.org/2013/03/TrialDesign
Annotations	The type that specifies a dosing variable.
Diagram	
Attributes	**QName** inputType — **Type** design:DoseInputTypeType — **Use** required Specifies the type of dosing. Is it assigned to a variable or is it an input to a system of ODEs.

1.4.2 Complex Type `design:TreatmentType`

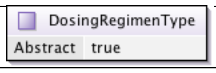
Namespace	http://www.pharmml.org/2013/03/TrialDesign

Annotations	Describes a treatment, by which we mean one or more regimens that can be appied to a subject.
Diagram	
Type	extension of design:ActivityType

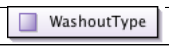
1.4.3 Complex Type `design:ActivityType`

Namespace	http://www.pharmml.org/2013/03/TrialDesign
Annotations	An activity that occurs during dosing.
Diagram	

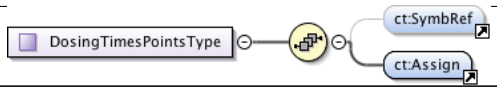
1.4.4 Complex Type `design:DosingRegimenType`

Namespace	http://www.pharmml.org/2013/03/TrialDesign
Annotations	Defines a dosing regimen type.
Diagram	

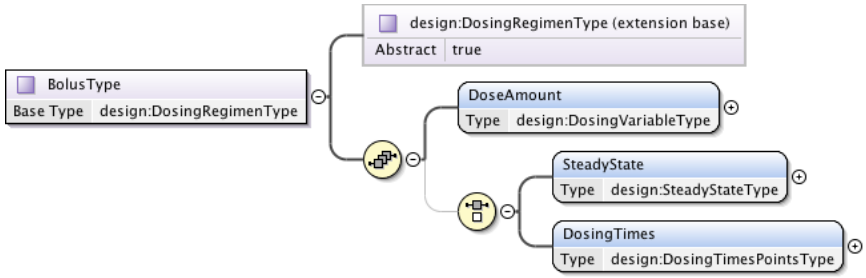
1.4.5 Complex Type `design:WashoutType`

Namespace	http://www.pharmml.org/2013/03/TrialDesign
Annotations	A type that defines a washout epoch. When this applies the system is reinitialised.
Diagram	

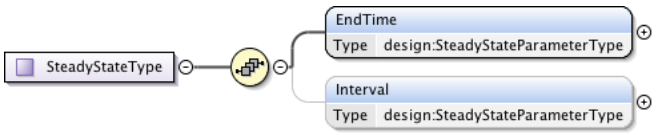
1.4.6 Complex Type `design:DosingTimesPointsType`

Namespace	http://www.pharmml.org/2013/03/TrialDesign
Annotations	Defines the dosing timepoints.
Diagram	

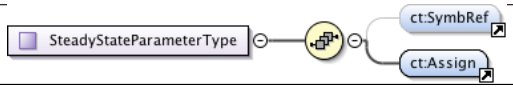
1.4.7 Complex Type `design:BolusType`

Namespace	http://www.pharmml.org/2013/03/TrialDesign
Annotations	Type the defines bolus dosing.
Diagram	
Type	extension of design:DosingRegimenType

1.4.8 Complex Type `design:SteadyStateType`

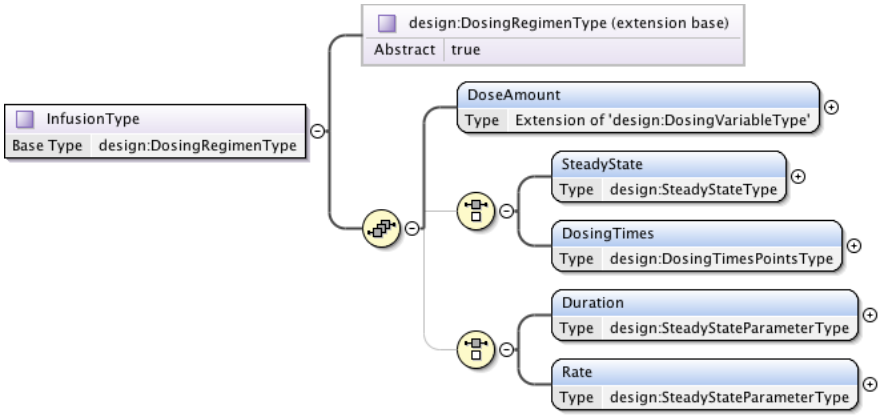
Namespace	http://www.pharmml.org/2013/03/TrialDesign
Annotations	Type that specifies steady state dosing.
Diagram	

1.4.9 Complex Type `design:SteadyStateParameterType`

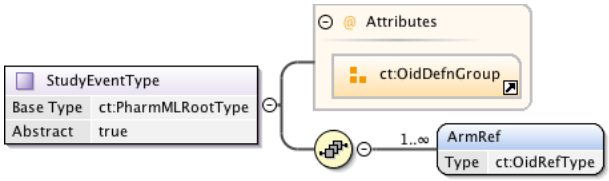
Namespace	http://www.pharmml.org/2013/03/TrialDesign
Annotations	Type that defines a paremeter used by steady state dosing.
Diagram	

1.4.10 Complex Type `design:InfusionType`

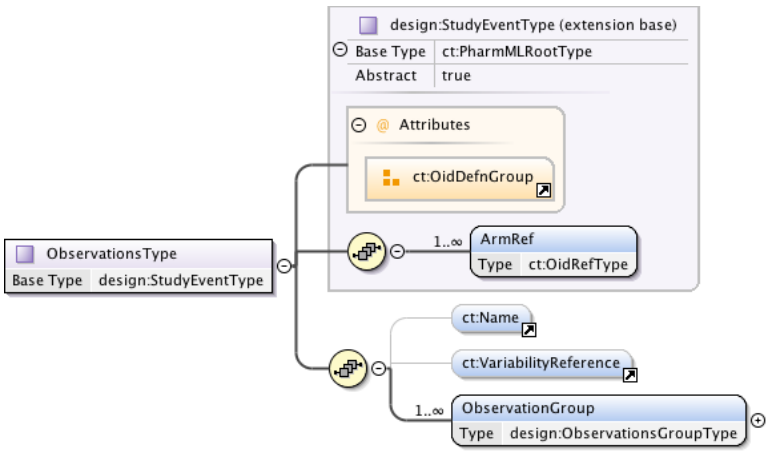
Namespace	http://www.pharmml.org/2013/03/TrialDesign
Annotations	Type the defines infusion dosing.

Diagram	
Type	extension of design:DosingRegimenType

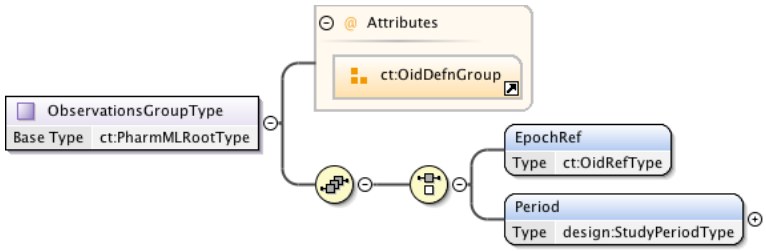
1.4.11 Complex Type `design:StudyEventType`

Namespace	http://www.pharmml.org/2013/03/TrialDesign
Annotations	A type defining a study event (a CDISC term). In our case such events are observations take during the study.
Diagram	
Type	extension of PharmMLRootType

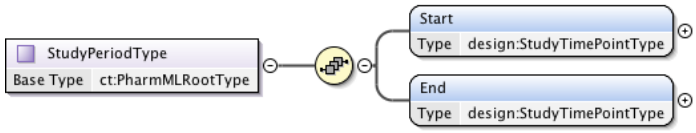
1.4.12 Complex Type `design:ObservationsType`

Namespace	http://www.pharmml.org/2013/03/TrialDesign
Annotations	A type defining the set of observations taken from a set of subjects.
Diagram	
Type	extension of design:StudyEventType

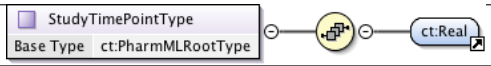
1.4.13 Complex Type `design:ObservationsGroupType`

Namespace	http://www.pharmml.org/2013/03/TrialDesign
Annotations	A type defining the timepoint in a study.
Diagram	
Type	extension of PharmMLRootType

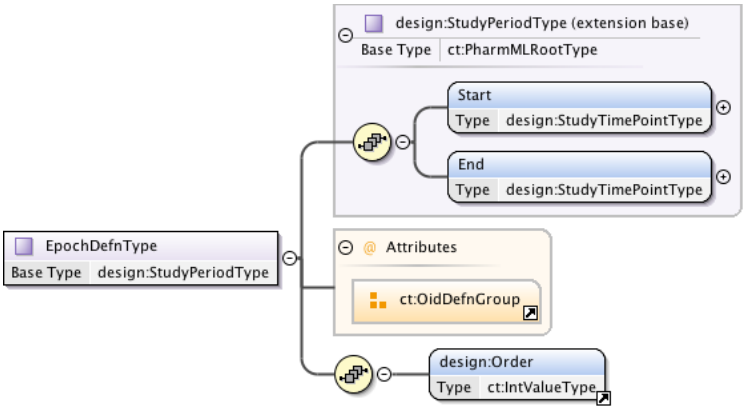
1.4.14 Complex Type `design:StudyPeriodType`

Namespace	http://www.pharmml.org/2013/03/TrialDesign
Annotations	A type defining the time period in a study.
Diagram	
Type	extension of PharmMLRootType

1.4.15 Complex Type `design:StudyTimePointType`

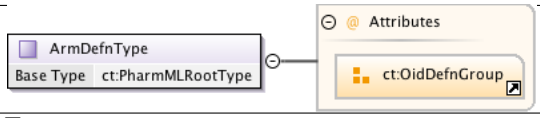
Namespace	http://www.pharmml.org/2013/03/TrialDesign
Annotations	A type defining the timepoint in a study.
Diagram	
Type	extension of PharmMLRootType

1.4.16 Complex Type `design:EpochDefnType`

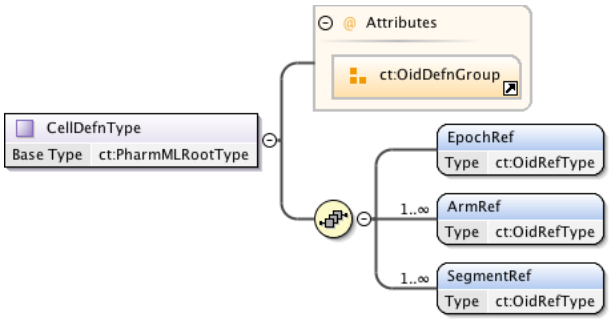
Namespace	http://www.pharmml.org/2013/03/TrialDesign
Annotations	Type that defines an epoch of the study.
Diagram	

Type	extension of design:StudyPeriodType

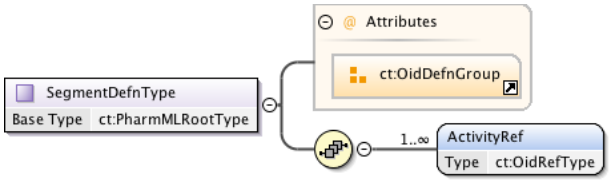
1.4.17 Complex Type `design:ArmDefnType`

Namespace	http://www.pharmml.org/2013/03/TrialDesign
Annotations	Type defining an arm of the study.
Diagram	
Type	extension of PharmMLRootType

1.4.18 Complex Type `design:CellDefnType`

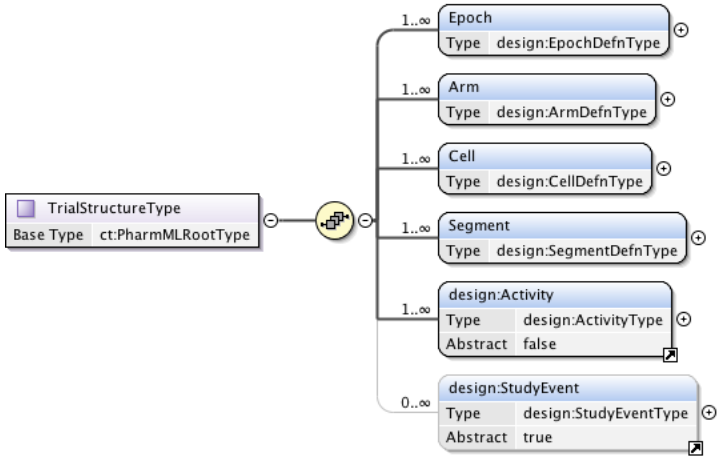
Namespace	http://www.pharmml.org/2013/03/TrialDesign
Annotations	Type defining a Cell in the study.
Diagram	
Type	extension of PharmMLRootType

1.4.19 Complex Type `design:SegmentDefnType`

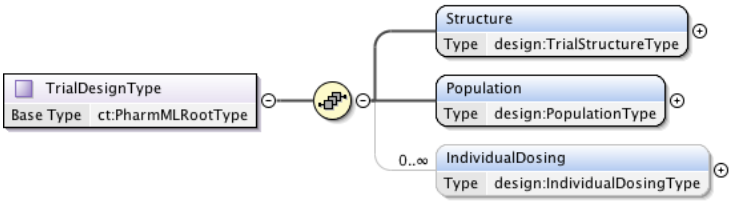
Namespace	http://www.pharmml.org/2013/03/TrialDesign
Annotations	Type defining a segment of the study.
Diagram	
Type	extension of PharmMLRootType

1.4.20 Complex Type `design:TrialStructureType`

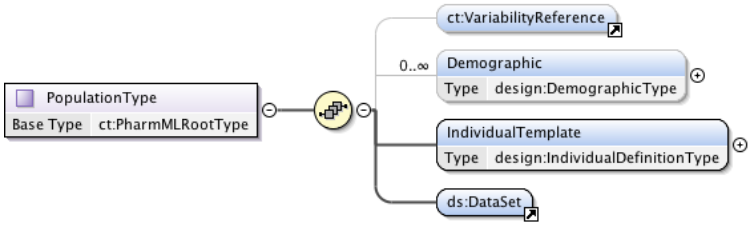
Namespace	http://www.pharmml.org/2013/03/TrialDesign
Annotations	Type that defines the structure of the study.

Diagram	
Type	extension of PharmMLRootType

1.4.21 Complex Type `design:TrialDesignType`

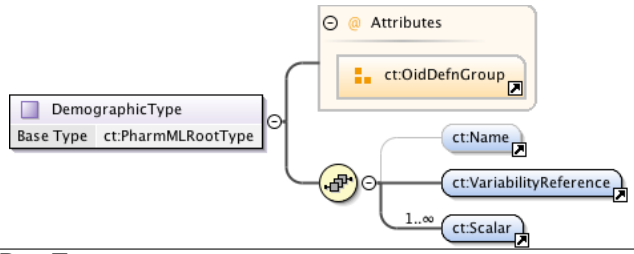
Namespace	http://www.pharmml.org/2013/03/TrialDesign
Annotations	A type that defines the design of the study.
Diagram	
Type	extension of PharmMLRootType

1.4.22 Complex Type `design:PopulationType`

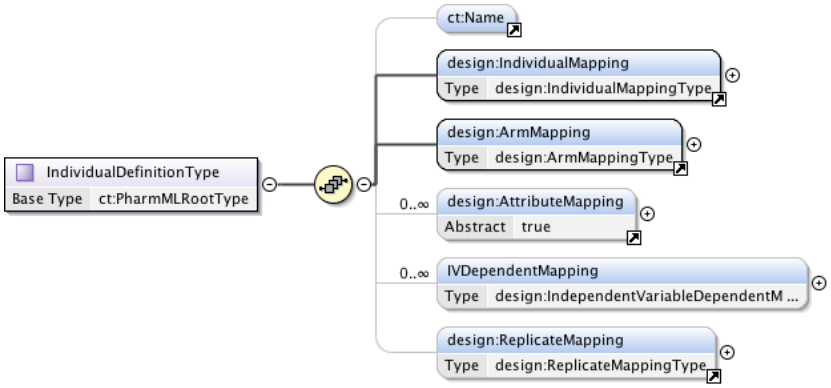
Namespace	http://www.pharmml.org/2013/03/TrialDesign
Annotations	Type defining the population of subjects in the study.
Diagram	
Type	extension of PharmMLRootType

1.4.23 Complex Type `design:DemographicType`

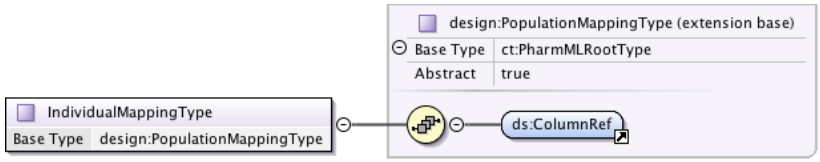
Namespace	http://www.pharmml.org/2013/03/TrialDesign
Annotations	The type defining a demographic attribute of the subject.

Diagram	
Type	extension of PharmMLRootType

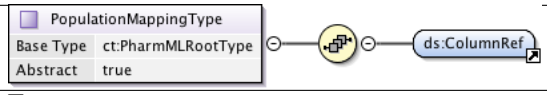
1.4.24 Complex Type `design:IndividualDefinitionType`

Namespace	http://www.pharmml.org/2013/03/TrialDesign
Annotations	A type defining a template definition of the individuals in the study.
Diagram	
Type	extension of PharmMLRootType

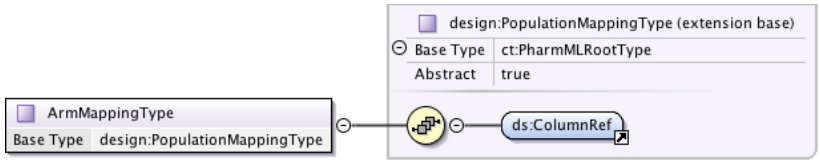
1.4.25 Complex Type `design:IndividualMappingType`

Namespace	http://www.pharmml.org/2013/03/TrialDesign
Annotations	A type defining the mapping of the individual identifier to the dataset.
Diagram	
Type	extension of design:PopulationMappingType

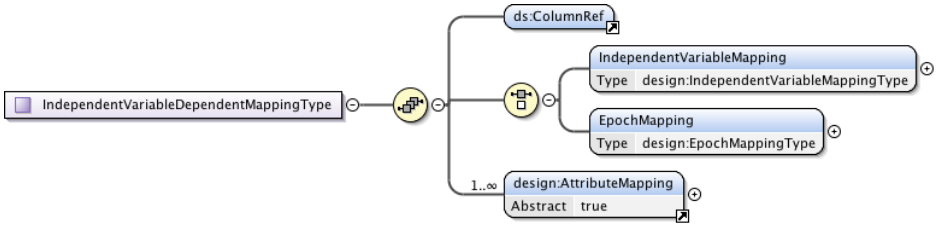
1.4.26 Complex Type `design:PopulationMappingType`

Namespace	http://www.pharmml.org/2013/03/TrialDesign
Annotations	A type defining the mapping from a property of the individual template the the dataset defining each individual.
Diagram	
Type	extension of PharmMLRootType

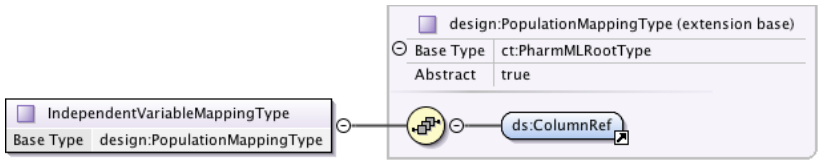
1.4.27 Complex Type `design:ArmMappingType`

Namespace	http://www.pharmml.org/2013/03/TrialDesign
Annotations	A type defining the mapping of the arm to the dataset.
Diagram	
Type	extension of design:PopulationMappingType

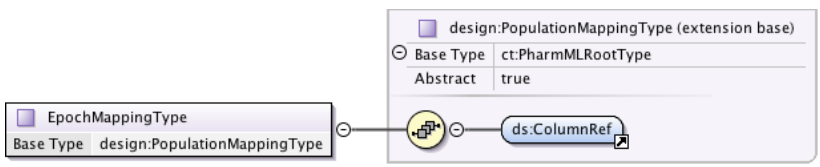
1.4.28 Complex Type `design:IndependentVariableDependentMappingType`

Namespace	http://www.pharmml.org/2013/03/TrialDesign
Annotations	A type defining the mapping of indendent variable dependent properties.
Diagram	

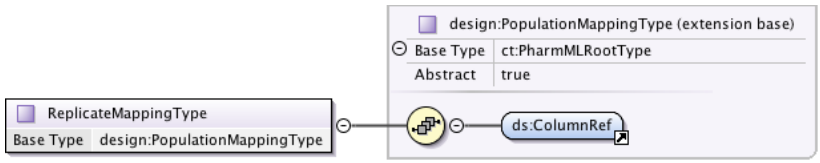
1.4.29 Complex Type `design:IndependentVariableMappingType`

Namespace	http://www.pharmml.org/2013/03/TrialDesign
Annotations	A type defining the independent variable mapping.
Diagram	
Type	extension of design:PopulationMappingType

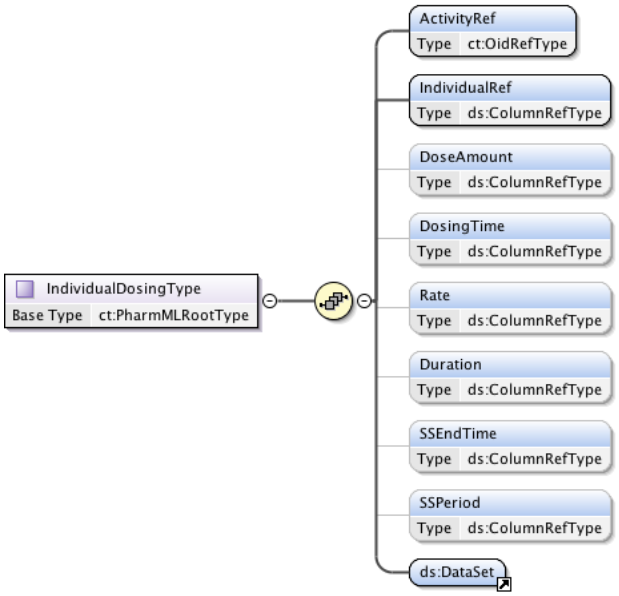
1.4.30 Complex Type `design:EpochMappingType`

Namespace	http://www.pharmml.org/2013/03/TrialDesign
Annotations	A type defining a mapping from the epoch to values in the column of a dataset.
Diagram	
Type	extension of design:PopulationMappingType

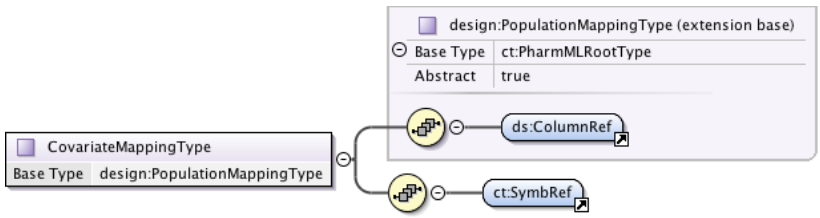
1.4.31 Complex Type `design:ReplicateMappingType`

Namespace	http://www.pharmml.org/2013/03/TrialDesign
Annotations	A type defining the mapping of values specified in a column of a dataset that defines the number time the values in this row should be repeated. This is a convenient way of defining groups of individuals with the same properties, without enumerating each individual explicitly.
Diagram	
Type	extension of design:PopulationMappingType

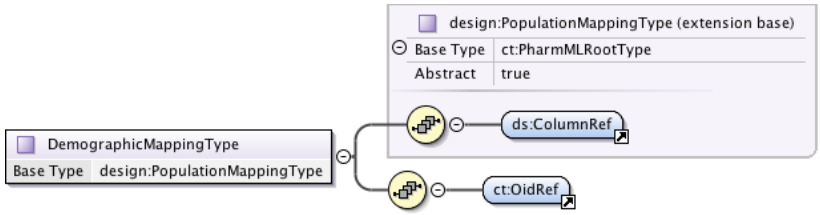
1.4.32 Complex Type `design:IndividualDosingType`

Namespace	http://www.pharmml.org/2013/03/TrialDesign
Diagram	
Type	extension of PharmMLRootType

1.4.33 Complex Type `design:CovariateMappingType`

Namespace	http://www.pharmml.org/2013/03/TrialDesign
Annotations	A type defining the mapping of a covariate to the values sepcified in a column of the dataset.
Diagram	
Type	extension of design:PopulationMappingType

1.4.34 Complex Type `design:DemographicMappingType`

Namespace	http://www.pharmml.org/2013/03/TrialDesign
Annotations	A type defining the mapping of a demographic attribute to the values sepcified in a column of the dataset.
Diagram	
Type	extension of design:PopulationMappingType

11.4 Modelling Steps

1 Namespace: "http://www.pharmml.org/2013/03/ModellingSteps"

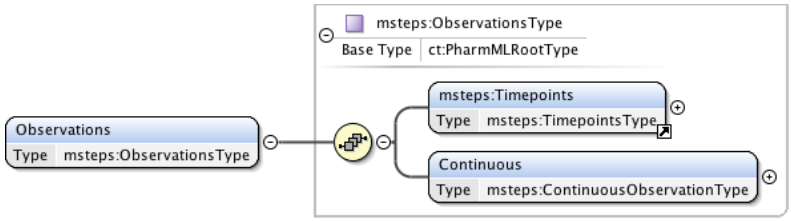
1.1 Schema(s)

1.1.1 Main schema `modellingSteps.xsd`

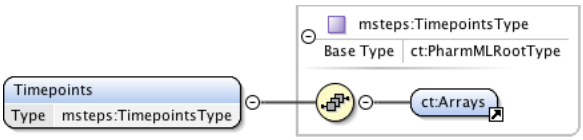
Namespace	http://www.pharmml.org/2013/03/ModellingSteps

1.2 Element(s)

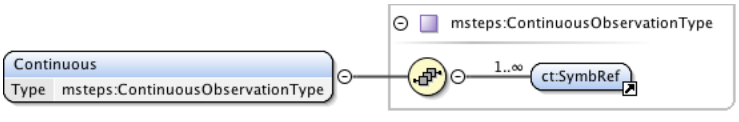
1.2.1 Element `msteps:SimulationStepType /msteps:Observations`

Namespace	http://www.pharmml.org/2013/03/ModellingSteps
Annotations	The observations to be generated by the simulation.
Diagram	
Type	msteps:ObservationsType

1.2.2 Element `msteps:Timepoints`

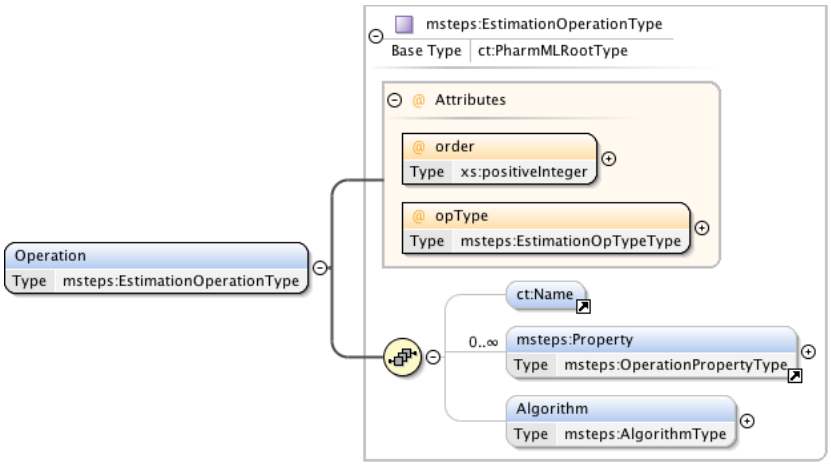
Namespace	http://www.pharmml.org/2013/03/ModellingSteps
Diagram	
Type	msteps:TimepointsType

1.2.3 Element `msteps:ObservationsType /msteps:Continuous`

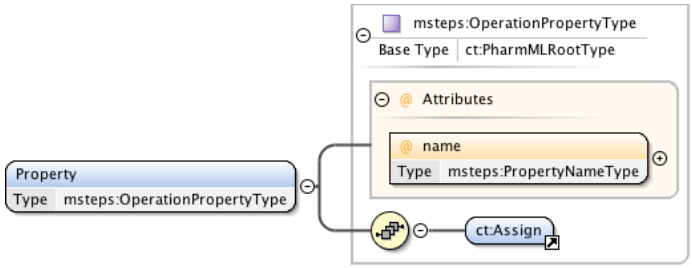
Namespace	http://www.pharmml.org/2013/03/ModellingSteps
Annotations	Defines the continuous variable output.
Diagram	
Type	msteps:ContinuousObservationType

1.2.4 Element `msteps:Operation`

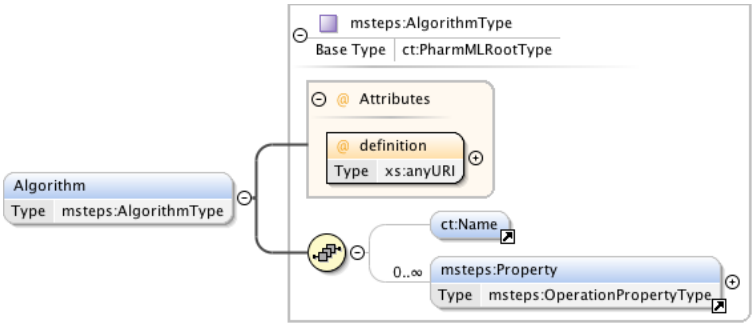
Namespace	http://www.pharmml.org/2013/03/ModellingSteps
Annotations	The estimation operation to be carried out.

Diagram	
Type	msteps:EstimationOperationType

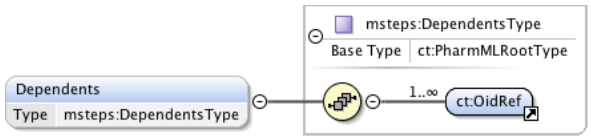
1.2.5 Element `msteps:Property`

Namespace	http://www.pharmml.org/2013/03/ModellingSteps
Annotations	Specifies a property.
Diagram	
Type	msteps:OperationPropertyType

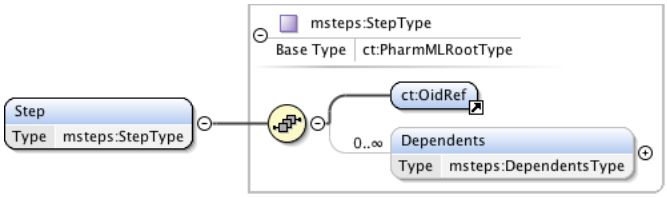
1.2.6 Element `msteps:EstimationOperationType /msteps:Algorithm`

Namespace	http://www.pharmml.org/2013/03/ModellingSteps
Annotations	Specifies the information about the estimation algorithms used.
Diagram	
Type	msteps:AlgorithmType

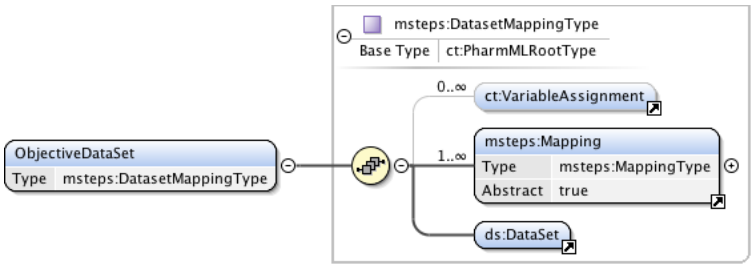
1.2.7 Element `msteps:StepType /msteps:Dependents`

Namespace	http://www.pharmml.org/2013/03/ModellingSteps
Annotations	The step dependent on this one.
Diagram	
Type	msteps:DependentsType

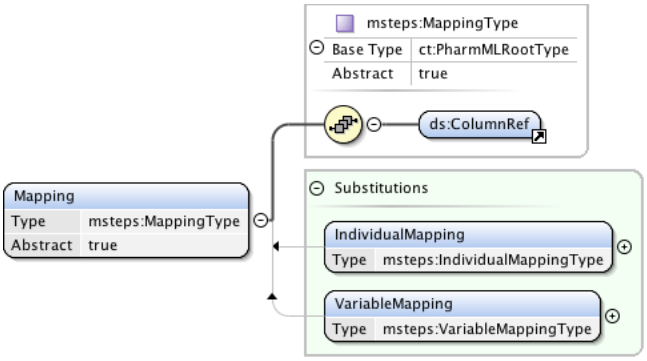
1.2.8 Element `msteps:StepDependencyType /msteps:Step`

Namespace	http://www.pharmml.org/2013/03/ModellingSteps
Annotations	Defines a step in the dependency graph.
Diagram	
Type	msteps:StepType

1.2.9 Element `msteps:EstimationStepType /msteps:ObjectiveDataSet`

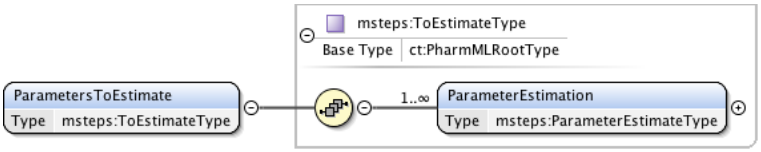
Namespace	http://www.pharmml.org/2013/03/ModellingSteps
Annotations	Defines the objective data to use in the estimation.
Diagram	
Type	msteps:DatasetMappingType

1.2.10 Element `msteps:Mapping`

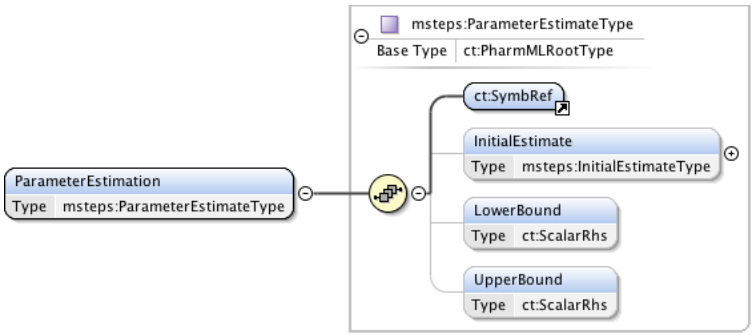
Namespace	http://www.pharmml.org/2013/03/ModellingSteps
Annotations	Defines mappings between the dataset and the model.
Diagram	

Type	msteps:MappingType
Substitution Group	• msteps:IndividualMapping • msteps:VariableMapping

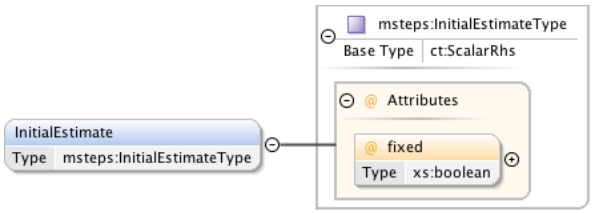
1.2.11 Element `msteps:EstimationStepType /msteps:ParametersToEstimate`

Namespace	http://www.pharmml.org/2013/03/ModellingSteps
Annotations	Specified the parameters of the model to be estimated.
Diagram	
Type	msteps:ToEstimateType

1.2.12 Element `msteps:ToEstimateType /msteps:ParameterEstimation`

Namespace	http://www.pharmml.org/2013/03/ModellingSteps
Annotations	Defines a symbol to be estimated. Note that this cannot be an individual parameter.
Diagram	
Type	msteps:ParameterEstimateType

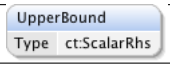
1.2.13 Element `msteps:ParameterEstimateType /msteps:InitialEstimate`

Namespace	http://www.pharmml.org/2013/03/ModellingSteps
Annotations	The initial estimate to use.
Diagram	
Type	msteps:InitialEstimateType

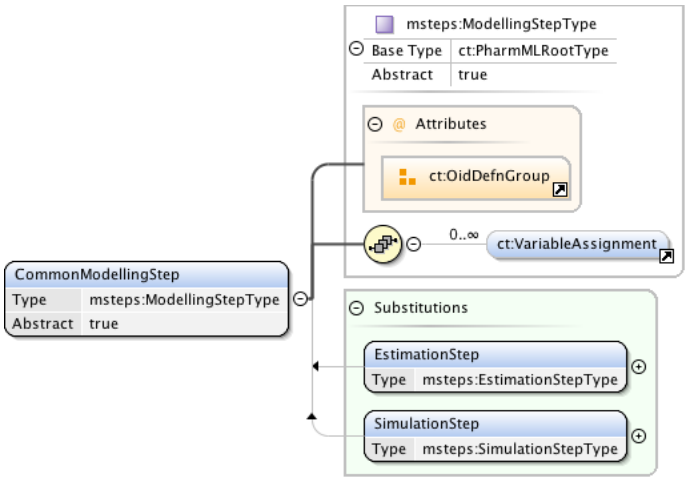
1.2.14 Element `msteps:ParameterEstimateType /msteps:LowerBound`

Namespace	http://www.pharmml.org/2013/03/ModellingSteps
Annotations	The lower bounds of the estimate.
Diagram	
Type	ScalarRhs

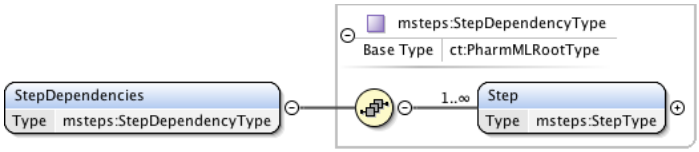
1.2.15 Element `msteps:ParameterEstimateType /msteps:UpperBound`

Namespace	http://www.pharmml.org/2013/03/ModellingSteps
Annotations	The upper bounds of the estimate.
Diagram	
Type	ScalarRhs

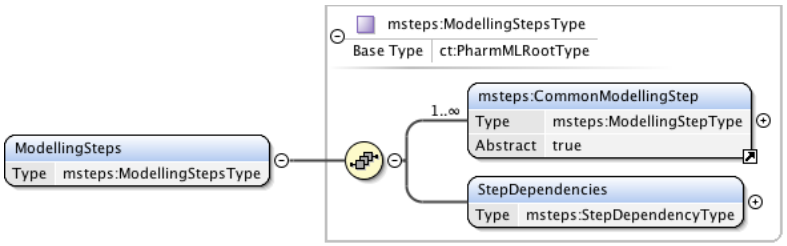
1.2.16 Element `msteps:CommonModellingStep`

Namespace	http://www.pharmml.org/2013/03/ModellingSteps
Annotations	Any modeling step.
Diagram	
Type	msteps:ModellingStepType
Substitution Group	• msteps:EstimationStep • msteps:SimulationStep

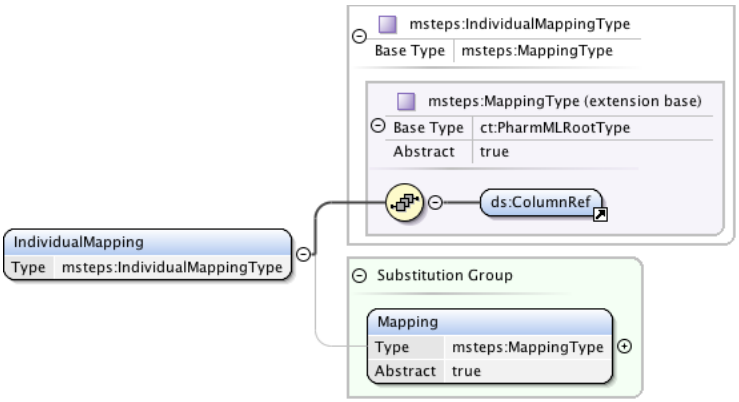
1.2.17 Element `msteps:ModellingStepsType /msteps:StepDependencies`

Namespace	http://www.pharmml.org/2013/03/ModellingSteps
Annotations	Defines dependencies between steps.
Diagram	
Type	msteps:StepDependencyType

1.2.18 Element `msteps:ModellingSteps`

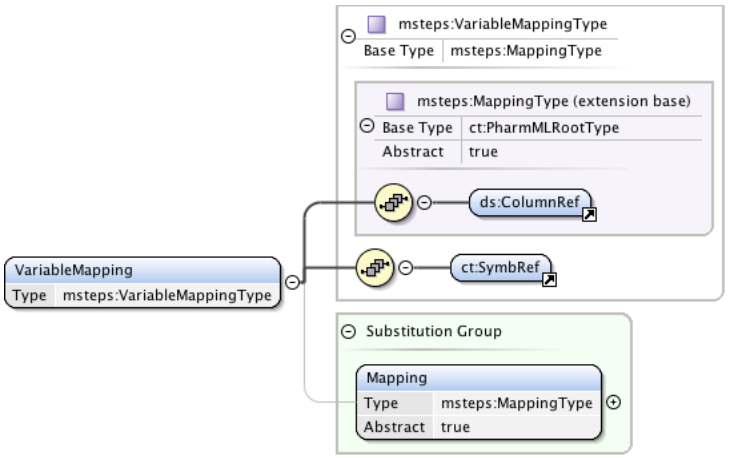
Namespace	http://www.pharmml.org/2013/03/ModellingSteps
Annotations	The modelling steps of the model.
Diagram	
Type	msteps:ModellingStepsType

1.2.19 Element `msteps:IndividualMapping`

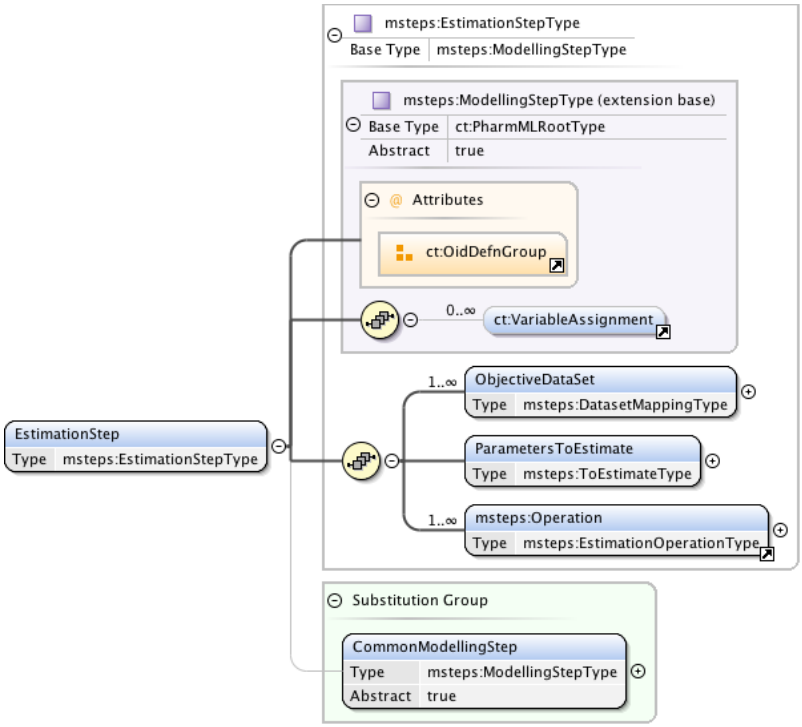
Namespace	http://www.pharmml.org/2013/03/ModellingSteps
Annotations	Specifies the mapping to the individual.
Diagram	
Type	msteps:IndividualMappingType
Substitution Group Affiliation	• msteps:Mapping

1.2.20 Element `msteps:VariableMapping`

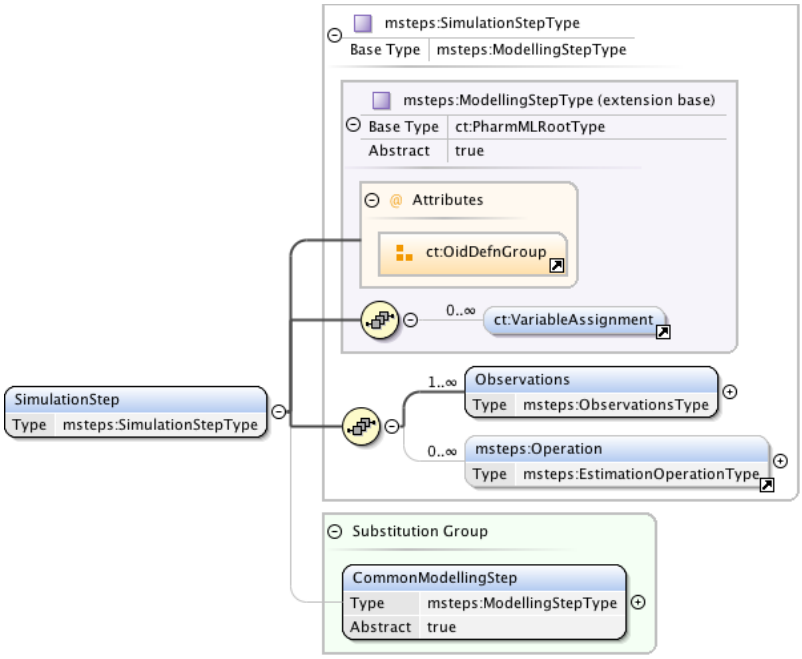
Namespace	http://www.pharmml.org/2013/03/ModellingSteps
Annotations	Specifies a mapping to a variable in the model.

Diagram	
Type	msteps:VariableMappingType
Substitution Group Affiliation	• msteps:Mapping

1.2.21 Element `msteps:EstimationStep`

Namespace	http://www.pharmml.org/2013/03/ModellingSteps
Annotations	An estimation step.
Diagram	
Type	msteps:EstimationStepType
Substitution Group Affiliation	• msteps:CommonModellingStep

1.2.22 Element `msteps:SimulationStep`

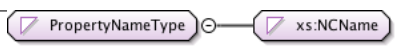
Namespace	http://www.pharmml.org/2013/03/ModellingSteps
Annotations	A simulation step.
Diagram	
Type	msteps:SimulationStepType
Substitution Group Affiliation	• msteps:CommonModellingStep

1.3 Simple Type(s)

1.3.1 Simple Type `msteps:EstimationOpTypeType`

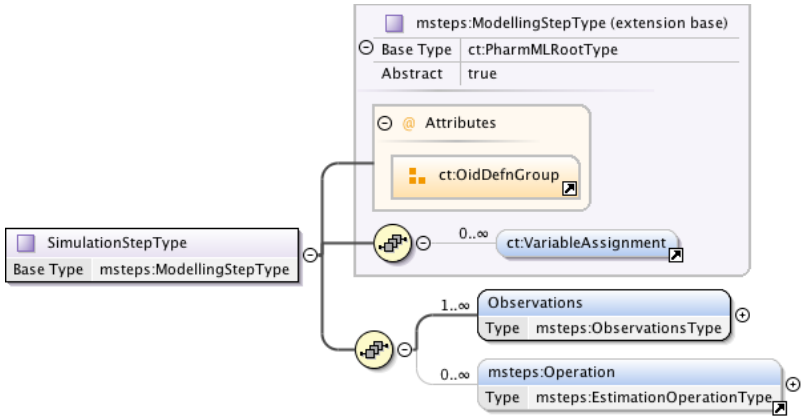
Namespace	http://www.pharmml.org/2013/03/ModellingSteps
Annotations	Type specifying the types of estimation operation.
Diagram	
Type	restriction of SymbolIdType

1.3.2 Simple Type `msteps:PropertyNameType`

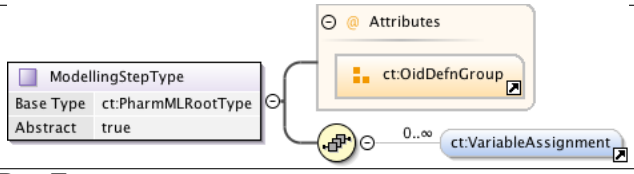
Namespace	http://www.pharmml.org/2013/03/ModellingSteps
Annotations	Type defining a property name.
Diagram	
Type	xs:NCName

1.4 Complex Type(s)

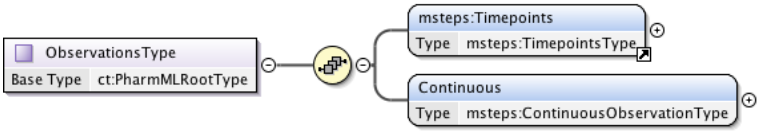
1.4.1 Complex Type `msteps:SimulationStepType`

Namespace	http://www.pharmml.org/2013/03/ModellingSteps
Annotations	Type defining a simulation step.
Diagram	
Type	extension of msteps:ModellingStepType

1.4.2 Complex Type `msteps:ModellingStepType`

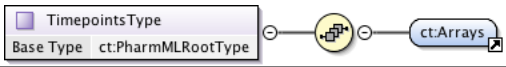
Namespace	http://www.pharmml.org/2013/03/ModellingSteps
Annotations	Abstract type specifying the common features of a modelling step.
Diagram	
Type	extension of PharmMLRootType

1.4.3 Complex Type `msteps:ObservationsType`

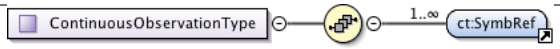
Namespace	http://www.pharmml.org/2013/03/ModellingSteps
Annotations	Type defining the observation timepoints.
Diagram	
Type	extension of PharmMLRootType

1.4.4 Complex Type `msteps:TimepointsType`

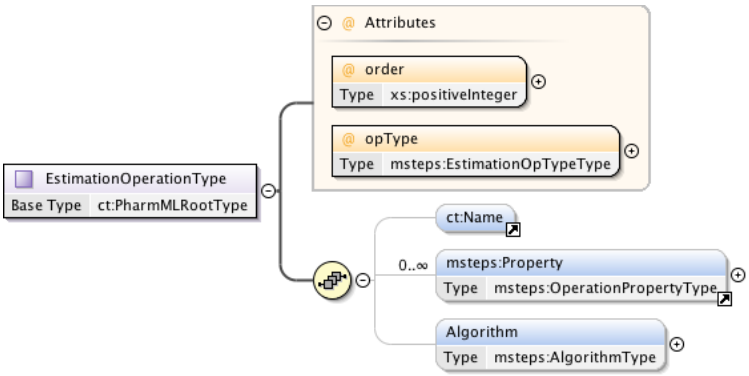
Namespace	http://www.pharmml.org/2013/03/ModellingSteps
Annotations	Timepoints.

Diagram	
Type	extension of PharmMLRootType

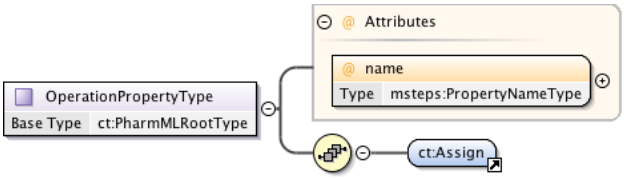
1.4.5 Complex Type `msteps:ContinuousObservationType`

Namespace	http://www.pharmml.org/2013/03/ModellingSteps
Annotations	Type defining the type of a continuous observation variable to be simulated.
Diagram	

1.4.6 Complex Type `msteps:EstimationOperationType`

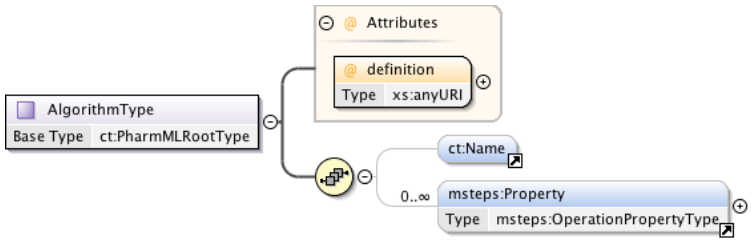
Namespace	http://www.pharmml.org/2013/03/ModellingSteps
Annotations	Type defining the estimation operation.
Diagram	
Type	extension of PharmMLRootType

1.4.7 Complex Type `msteps:OperationPropertyType`

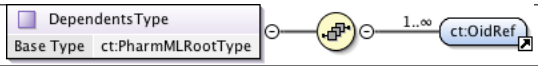
Namespace	http://www.pharmml.org/2013/03/ModellingSteps
Annotations	Type defining an operation property.
Diagram	
Type	extension of PharmMLRootType

1.4.8 Complex Type `msteps:AlgorithmType`

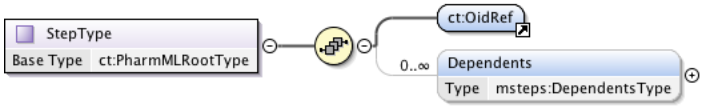
Namespace	http://www.pharmml.org/2013/03/ModellingSteps
Annotations	Type defining the type of the algorithm.

Diagram	
Type	extension of PharmMLRootType

1.4.9 Complex Type `msteps:DependentsType`

Namespace	http://www.pharmml.org/2013/03/ModellingSteps
Annotations	Type defining the dependent steps.
Diagram	
Type	extension of PharmMLRootType

1.4.10 Complex Type `msteps:StepType`

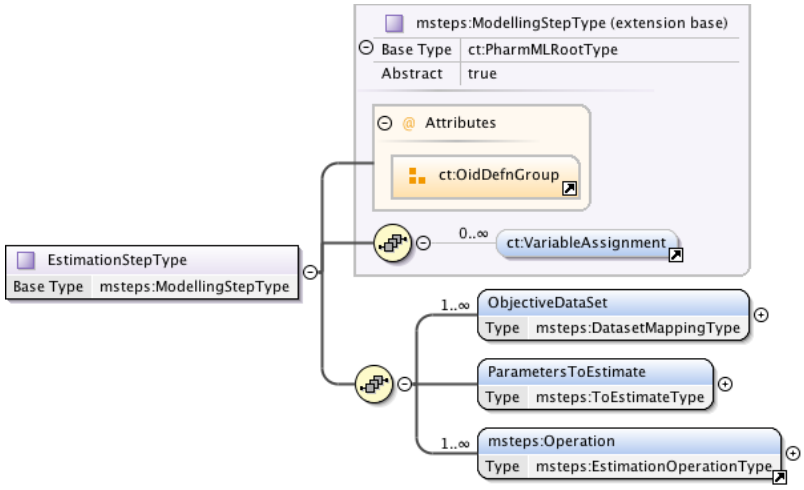
Namespace	http://www.pharmml.org/2013/03/ModellingSteps
Annotations	Type defining a modelling step and its dependencies.
Diagram	
Type	extension of PharmMLRootType

1.4.11 Complex Type `msteps:StepDependencyType`

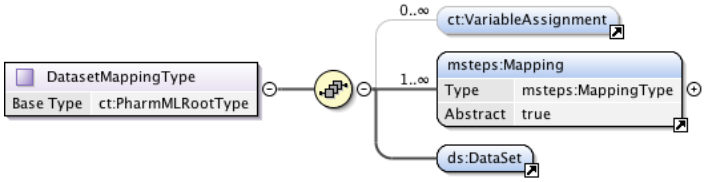
Namespace	http://www.pharmml.org/2013/03/ModellingSteps
Annotations	Type defining step dependencies.
Diagram	
Type	extension of PharmMLRootType

1.4.12 Complex Type `msteps:EstimationStepType`

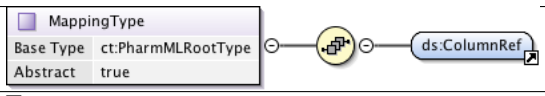
Namespace	http://www.pharmml.org/2013/03/ModellingSteps
Annotations	Type defining the estimation step.

Diagram	
Type	extension of msteps:ModellingStepType

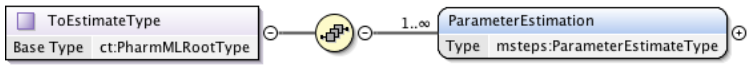
1.4.13 Complex Type `msteps:DatasetMappingType`

Namespace	http://www.pharmml.org/2013/03/ModellingSteps
Annotations	Type defining a mapping of objective data to the model.
Diagram	
Type	extension of PharmMLRootType

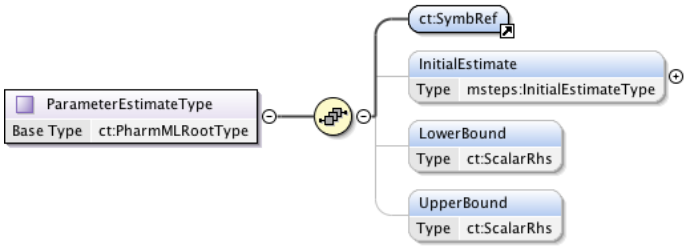
1.4.14 Complex Type `msteps:MappingType`

Namespace	http://www.pharmml.org/2013/03/ModellingSteps
Annotations	Abstract type that defines a mapping.
Diagram	
Type	extension of PharmMLRootType

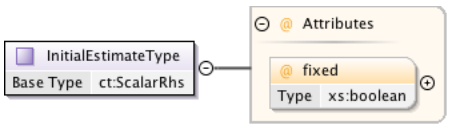
1.4.15 Complex Type `msteps:ToEstimateType`

Namespace	http://www.pharmml.org/2013/03/ModellingSteps
Annotations	Type defining the parameters to be estimated.
Diagram	
Type	extension of PharmMLRootType

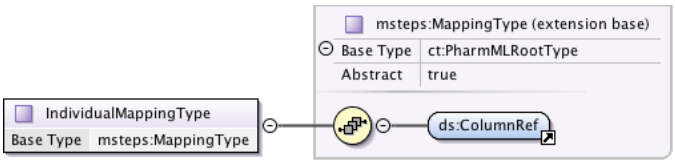
1.4.16 Complex Type `msteps:ParameterEstimateType`

Namespace	http://www.pharmml.org/2013/03/ModellingSteps
Annotations	Type defining paramaters to be estimated and their bounds and initial estimates.
Diagram	
Type	extension of PharmMLRootType

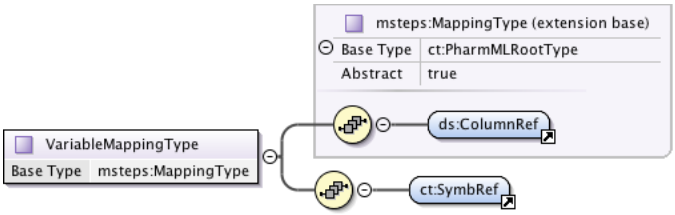
1.4.17 Complex Type `msteps:InitialEstimateType`

Namespace	http://www.pharmml.org/2013/03/ModellingSteps
Annotations	Type specifying an initial estimate.
Diagram	
Type	extension of ScalarRhs

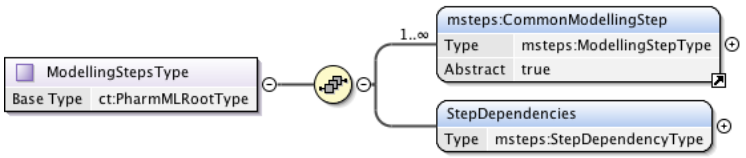
1.4.18 Complex Type `msteps:IndividualMappingType`

Namespace	http://www.pharmml.org/2013/03/ModellingSteps
Annotations	Type defining the mapping of a dataset to the individual.
Diagram	
Type	extension of msteps:MappingType

1.4.19 Complex Type `msteps:VariableMappingType`

Namespace	http://www.pharmml.org/2013/03/ModellingSteps
Annotations	Type defining a maping to a variable in the model.
Diagram	
Type	extension of msteps:MappingType

1.4.20 Complex Type `msteps:ModellingStepsType`

Namespace	http://www.pharmml.org/2013/03/ModellingSteps
Annotations	A type defining the modelling steps section.
Diagram	
Type	extension of PharmMLRootType

2 Namespace: ""

2.1 Attribute(s)

2.1.1 Attribute `msteps:OperationPropertyType /@name`

Namespace	No namespace
Annotations	The name of the property.
Type	msteps:PropertyNameType

2.1.2 Attribute `msteps:AlgorithmType /@definition`

Namespace	No namespace
Annotations	The estimation operation type.
Type	xs:anyURI

2.1.3 Attribute `msteps:EstimationOperationType /@order`

Namespace	No namespace
Annotations	Specifies the order of the operation.
Type	xs:positiveInteger

2.1.4 Attribute `msteps:EstimationOperationType /@opType`

Namespace	No namespace
Annotations	Specifies an estimation operation type.
Type	msteps:EstimationOpTypeType

2.1.5 Attribute `msteps:InitialEstimateType /@fixed`

Namespace	No namespace
Annotations	Specifies whether the initial estimate is fixed. If it is then this means that this parameter is not estimated, but assigned. If fixed is true then the upper and lower bounds are ignored.

Type	xs:boolean

11.5 Common Types

1 Namespace: "http://www.pharmml.org/2013/03/CommonTypes"

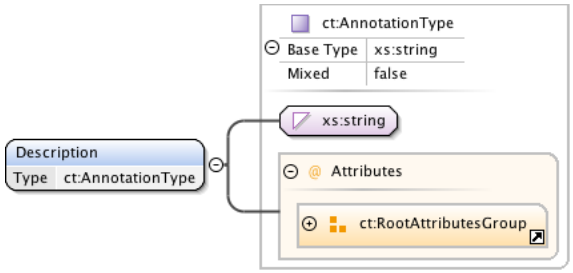
1.1 Schema(s)

1.1.1 Main schema `commonTypes.xsd`

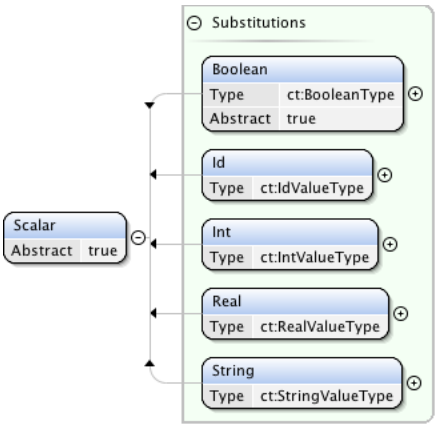
Namespace	http://www.pharmml.org/2013/03/CommonTypes

1.2 Element(s)

1.2.1 Element `ct:Description`

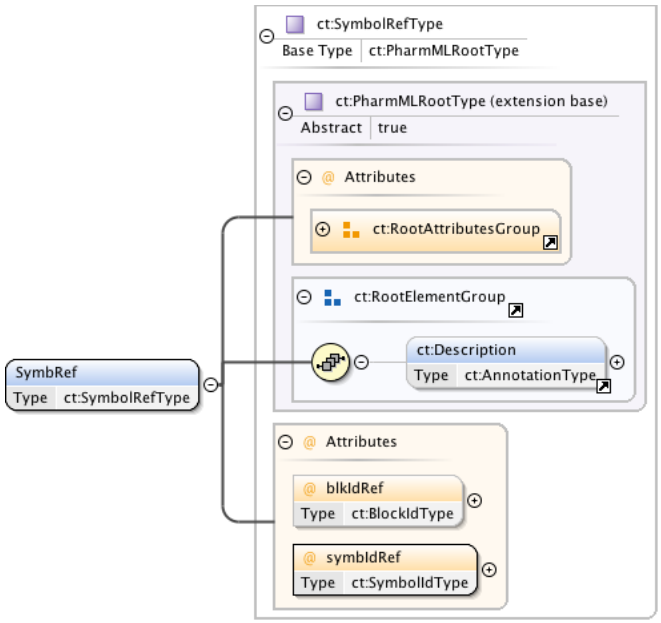
Namespace	http://www.pharmml.org/2013/03/CommonTypes
Annotations	Element provides additional documentation about its parent element.
Diagram	
Type	ct:AnnotationType
Attributes	**QName** / **Type** / **Use**: **id** / ct:IdType / optional The element identifier.

1.2.2 Element `ct:Scalar`

Namespace	http://www.pharmml.org/2013/03/CommonTypes
Annotations	An element that defines a scalar value. This element is abstract the specific scalar elements are specified by the substitution group.
Diagram	

Substitution Group	• ct:Int • ct:Real • ct:String • ct:Id • ct:Boolean • ct:True • ct:False

1.2.3 Element ct : SymbRef

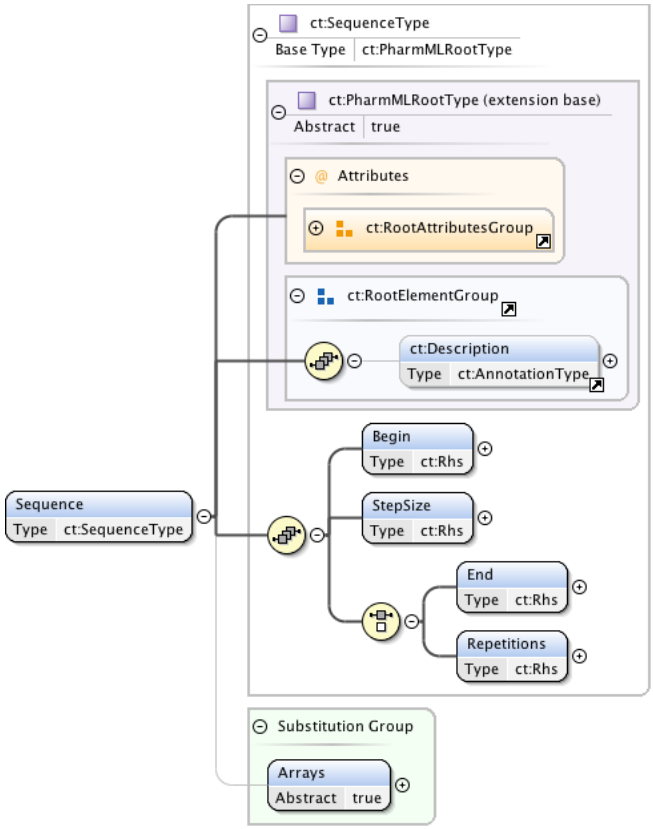
Namespace	http://www.pharmml.org/2013/03/CommonTypes
Annotations	Element references a symbol defined elsewhere in the document.
Diagram	
Type	ct:SymbolRefType

Attributes

QName	Type	Use			
blkIdRef	ct:BlockIdType	optional			
	ID referencing a Block.				
id	ct:IdType	optional			
	The element identifier.				
symbIdRef	ct:SymbolIdType	required			
	ID referencing a Symbol.				

1.2.4 Element ct : Sequence

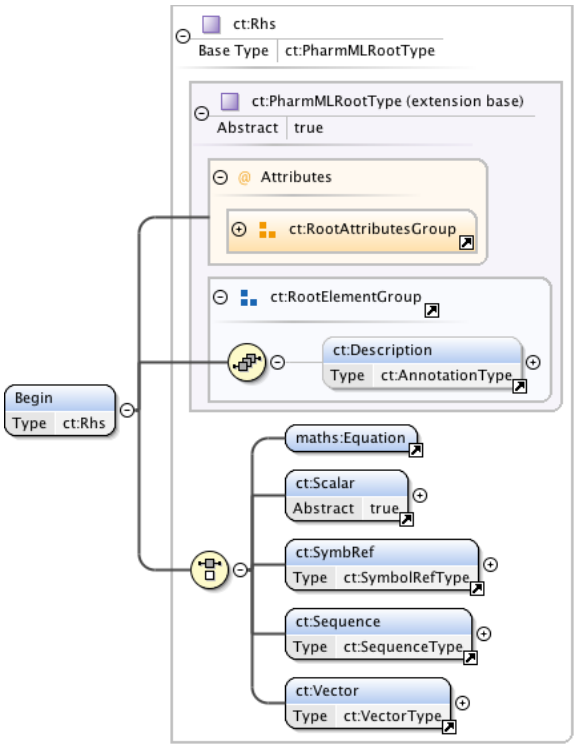
Namespace	http://www.pharmml.org/2013/03/CommonTypes
Annotations	Element defines a uniform sequence of values.

Diagram	
Type	ct:SequenceType
Substitution Group Affiliation	• ct:Arrays

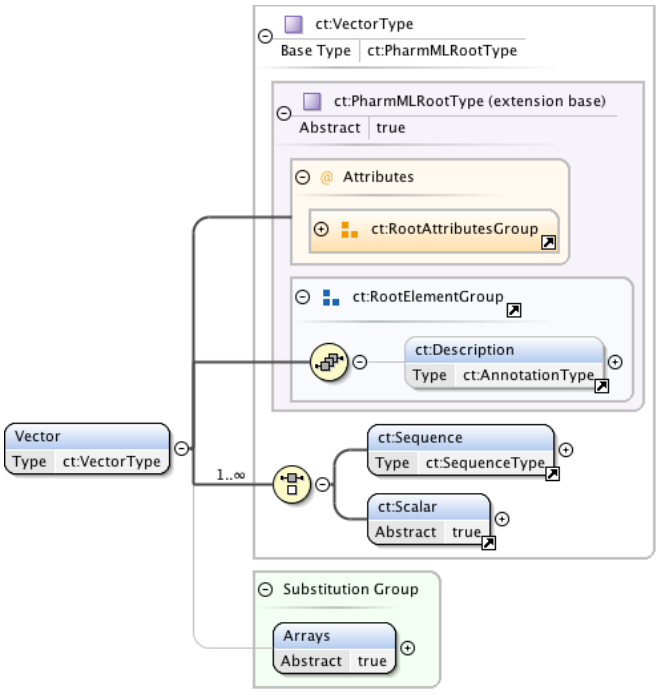
Attributes	QName	Type	Use			
	id	ct:IdType	optional			
		The element identifier.				

1.2.5 Element ct:SequenceType /ct:Begin

Namespace	http://www.pharmml.org/2013/03/CommonTypes
Annotations	The initial value of the sequence.

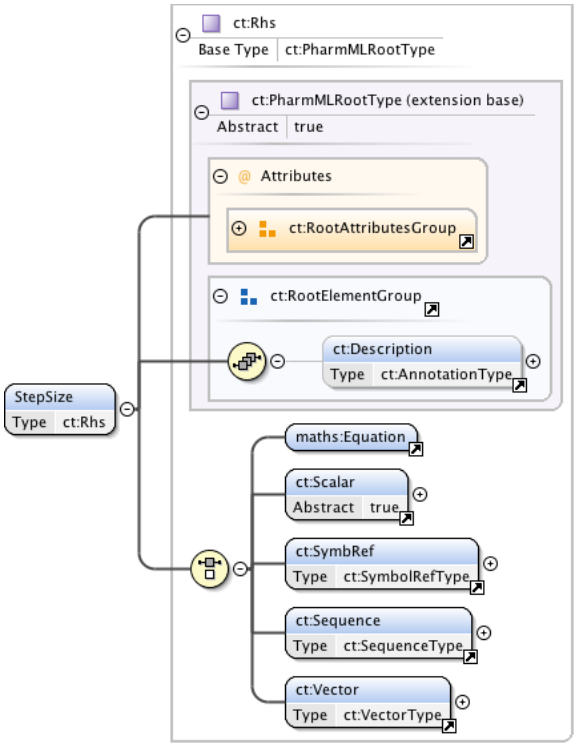
Diagram	
Type	ct:Rhs
Attributes	**QName** / **id** — **Type**: ct:IdType — **Use**: optional — The element identifier.

1.2.6 Element `ct:Vector`

Namespace	http://www.pharmml.org/2013/03/CommonTypes
Annotations	Element deines a vector of values.
Diagram	

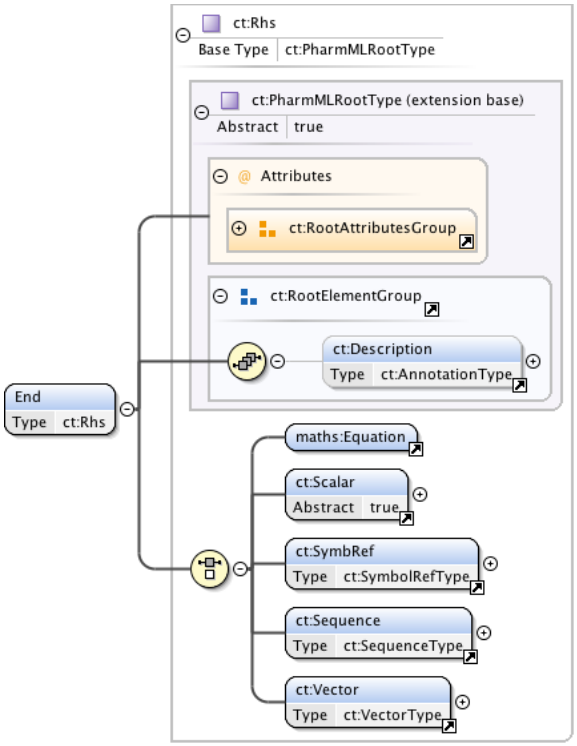
Type	ct:VectorType
Substitution Group Affiliation	• ct:Arrays
Attributes	**QName** / **Type** / **Use** **id** / ct:IdType / optional The element identifier.

1.2.7 Element ct:SequenceType /ct:StepSize

Namespace	http://www.pharmml.org/2013/03/CommonTypes
Annotations	The amount incremented to get the next value in the sequence. The step size can be negative
Diagram	
Type	ct:Rhs
Attributes	**QName** / **Type** / **Use** **id** / ct:IdType / optional The element identifier.

1.2.8 Element ct:SequenceType /ct:End

Namespace	http://www.pharmml.org/2013/03/CommonTypes
Annotations	The maximum possible value of the sequence.

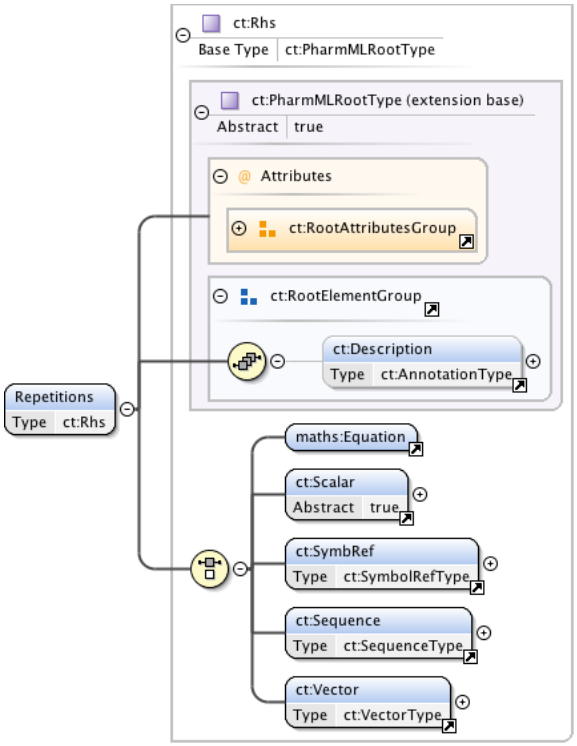
Diagram	
Type	ct:Rhs

Attributes

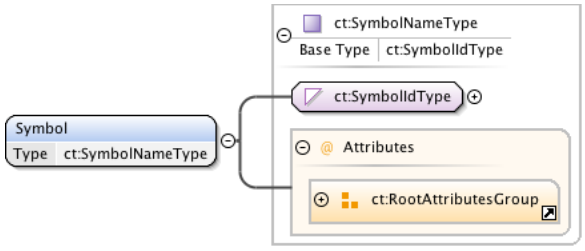
QName	Type	Use			
id	ct:IdType	optional			
	The element identifier.				

1.2.9 Element `ct:SequenceType /ct:Repetitions`

Namespace	http://www.pharmml.org/2013/03/CommonTypes
Annotations	The number of times to incrememnt the sequence by the step size.

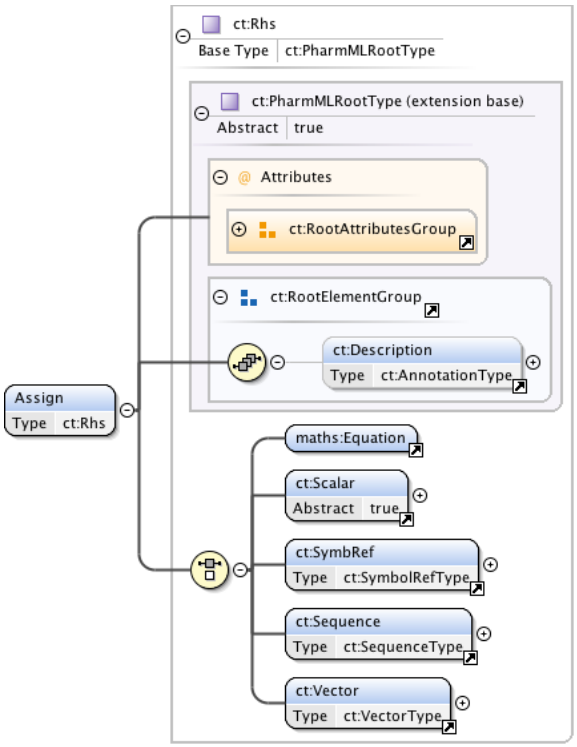
Diagram	
Type	ct:Rhs
Attributes	**QName** / **Type** / **Use** **id** / ct:IdType / optional The element identifier.

1.2.10 Element `ct:Symbol`

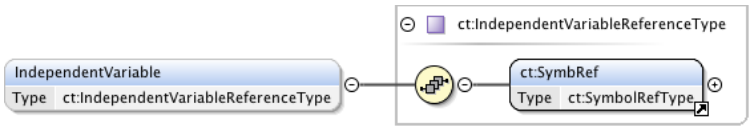
Namespace	http://www.pharmml.org/2013/03/CommonTypes
Annotations	Element defining the name of the symbol.
Diagram	
Type	ct:SymbolNameType
Attributes	**QName** / **Type** / **Use** **id** / ct:IdType / optional The element identifier.

1.2.11 Element `ct:Assign`

Namespace	http://www.pharmml.org/2013/03/CommonTypes
Annotations	This element indicates that the value(s) or equation defined by its child elements are to be assigned to its parent element. It is the equivalent of an assignment operator.

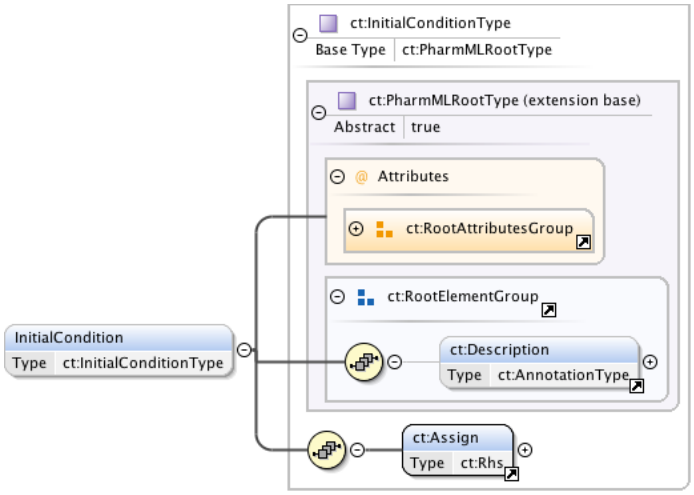
Diagram	
Type	ct:Rhs
Attributes	**QName** / **id**: **Type** ct:IdType; **Use** optional The element identifier.

1.2.12 Element `ct:DerivativeVariableType /ct:IndependentVariable`

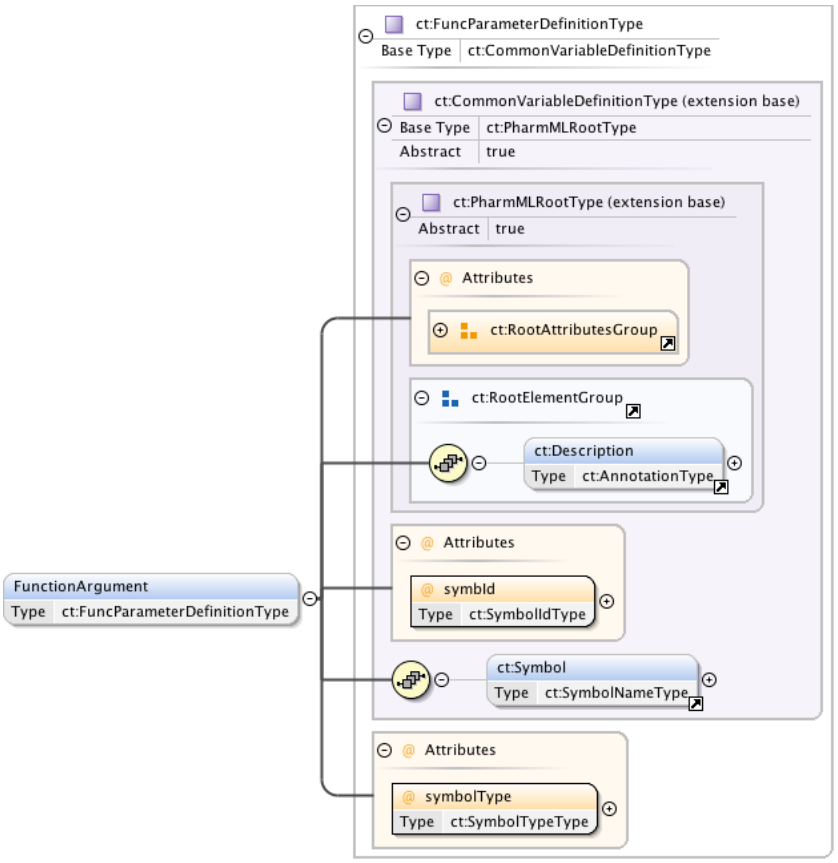
Namespace	http://www.pharmml.org/2013/03/CommonTypes
Annotations	The independent variable of the derivative.
Diagram	
Type	ct:IndependentVariableReferenceType

1.2.13 Element `ct:DerivativeVariableType /ct:InitialCondition`

Namespace	http://www.pharmml.org/2013/03/CommonTypes
Annotations	The value of the initial condition.

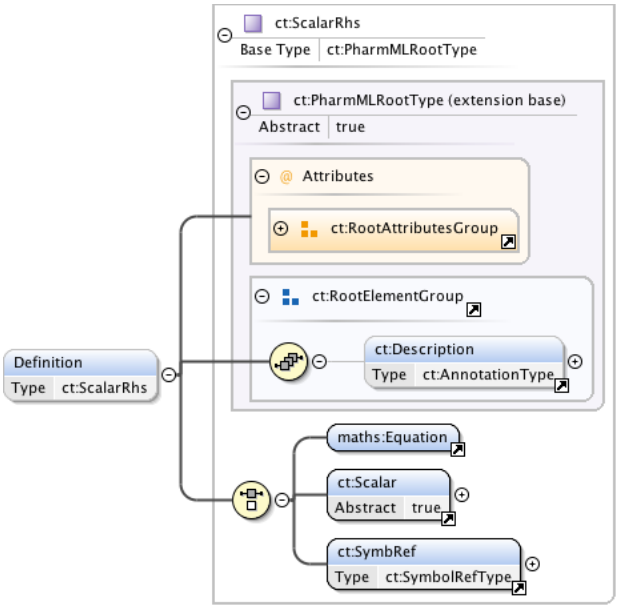
Diagram	
Type	ct:InitialConditionType
Attributes	**QName** / **id** / **Type** ct:IdType / **Use** optional / The element identifier.

1.2.14 Element `ct:FunctionDefinitionType /ct:FunctionArgument`

Namespace	http://www.pharmml.org/2013/03/CommonTypes
Annotations	The argument (parameter) to the function.
Diagram	
Type	ct:FuncParameterDefinitionType

Attributes	**QName**	**Type**	**Use**			
	id	ct:IdType	optional			
		The element identifier.				
	symbId	ct:SymbolIdType	required			
		The symbol id used to define the variable.				
	symbolType	ct:SymbolTypeType	required			
		The type of the function definition.				

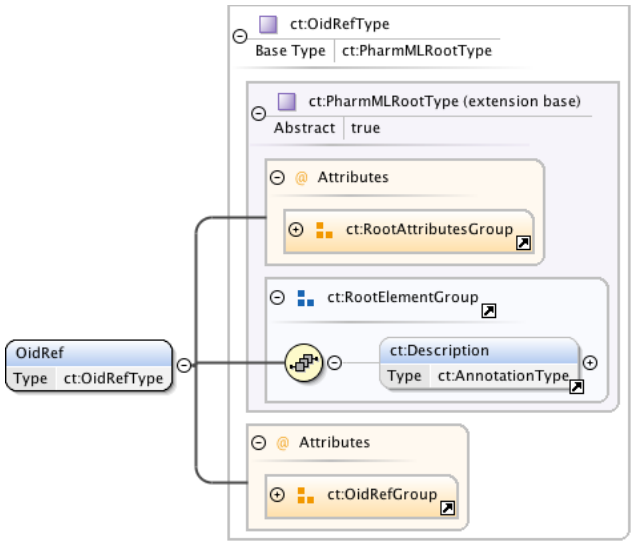
1.2.15 Element `ct:FunctionDefinitionType /ct:Definition`

Namespace	http://www.pharmml.org/2013/03/CommonTypes
Annotations	The body of the function defintion.
Diagram	
Type	ct:ScalarRhs

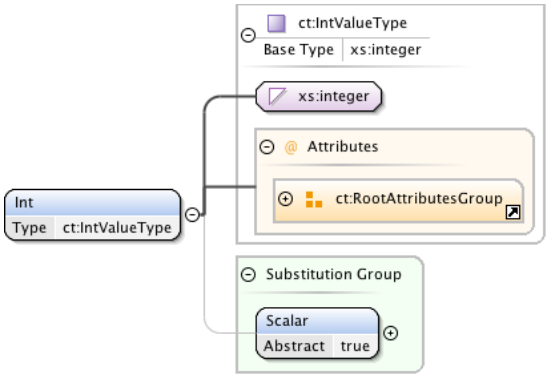
Attributes	**QName**	**Type**	**Use**			
	id	ct:IdType	optional			
		The element identifier.				

1.2.16 Element `ct:OidRef`

Namespace	http://www.pharmml.org/2013/03/CommonTypes
Annotations	An element that provides a reference to an OID.

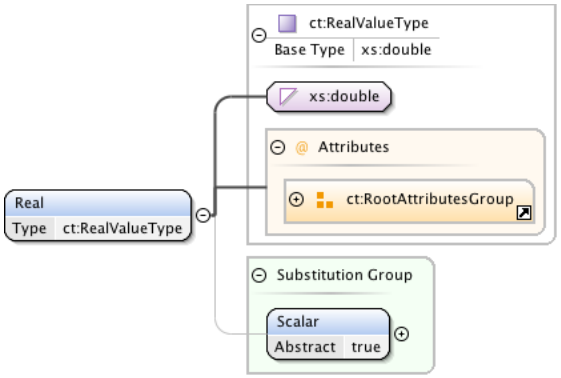
Diagram	
Type	ct:OidRefType
Attributes	**QName** / **Type** / **Use** **id** / ct:IdType / optional The element identifier. **oidRef** / ct:oidType / required

1.2.17 Element `ct:Int`

Namespace	http://www.pharmml.org/2013/03/CommonTypes
Annotations	The element defines an integer value. It is a scalar.
Diagram	
Type	ct:IntValueType
Substitution Group Affiliation	• ct:Scalar
Attributes	**QName** / **Type** / **Use** **id** / ct:IdType / optional The element identifier.

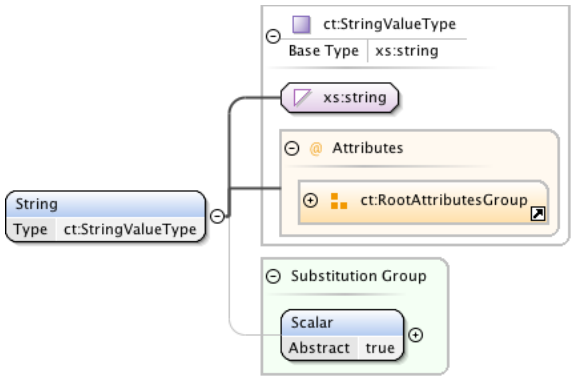
1.2.18 Element `ct:Real`

Namespace	http://www.pharmml.org/2013/03/CommonTypes

Annotations	The element defines an real value. It is a scalar.
Diagram	
Type	ct:RealValueType
Substitution Group Affiliation	• ct:Scalar

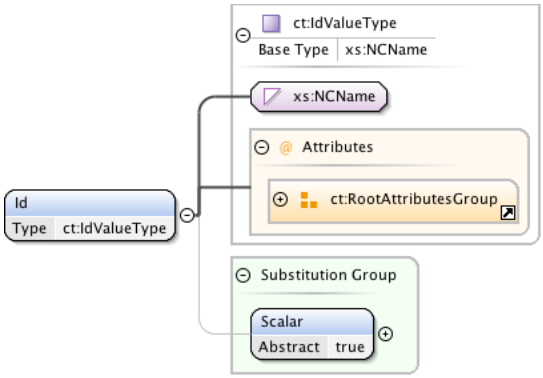
Attributes	QName	Type	Use			
	id	ct:IdType	optional			
		The element identifier.				

1.2.19 Element `ct:String`

Namespace	http://www.pharmml.org/2013/03/CommonTypes
Annotations	The element defines an string value. It is a scalar.
Diagram	
Type	ct:StringValueType
Substitution Group Affiliation	• ct:Scalar

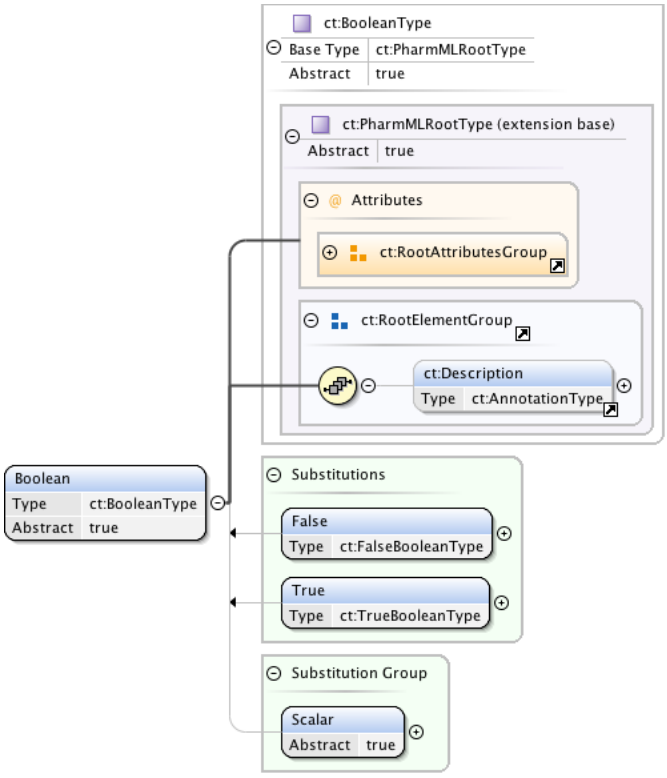
Attributes	QName	Type	Use			
	id	ct:IdType	optional			
		The element identifier.				

1.2.20 Element `ct:Id`

Namespace	http://www.pharmml.org/2013/03/CommonTypes
Annotations	Element defines a value of type Id.
Diagram	
Type	ct:IdValueType
Substitution Group Affiliation	• ct:Scalar

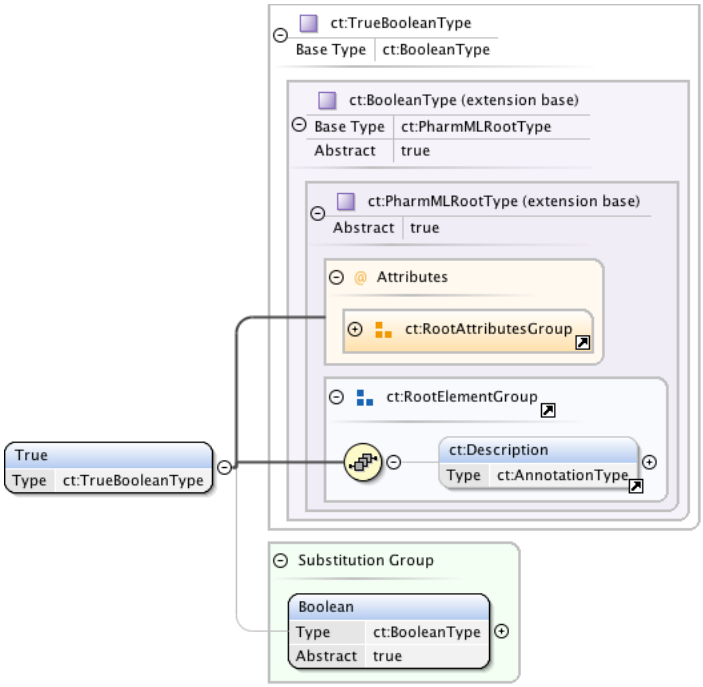
Attributes	QName	Type	Use			
	id	ct:IdType	optional			
		The element identifier.				

1.2.21 Element `ct:Boolean`

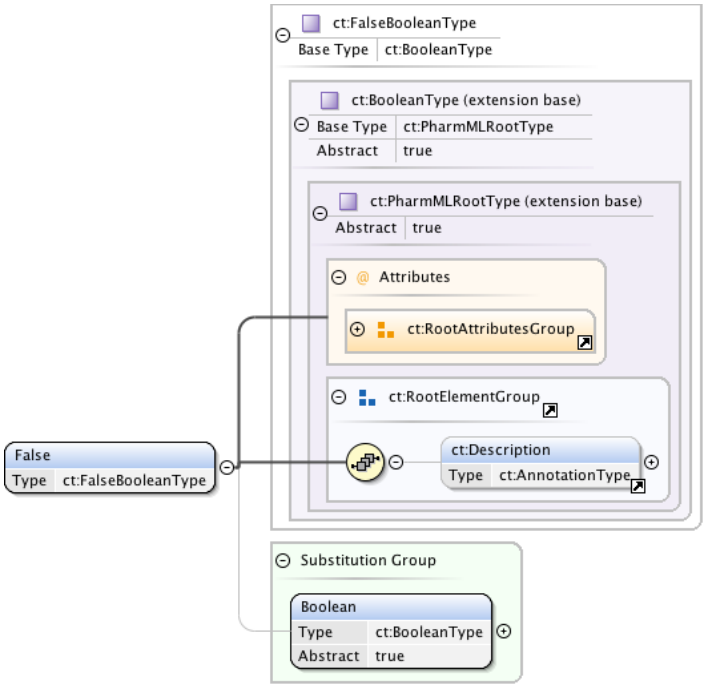
Namespace	http://www.pharmml.org/2013/03/CommonTypes
Annotations	This abstract element defines an Booelan value.
Diagram	

Type	ct:BooleanType
Substitution Group	• ct:True • ct:False
Substitution Group Affiliation	• ct:Scalar
Attributes	**QName** / **Type** / **Use** **id** / ct:IdType / optional The element identifier.

1.2.22 Element `ct:True`

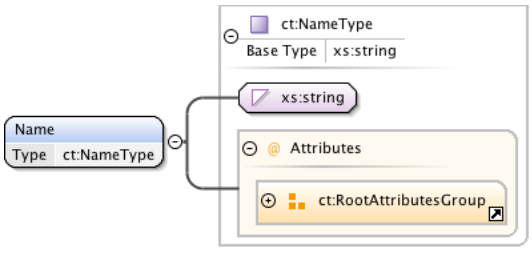
Namespace	http://www.pharmml.org/2013/03/CommonTypes
Annotations	This element defines a TRUE Booelan value.
Diagram	
Type	ct:TrueBooleanType
Substitution Group Affiliation	• ct:Boolean
Attributes	**QName** / **Type** / **Use** **id** / ct:IdType / optional The element identifier.

1.2.23 Element `ct:False`

Namespace	http://www.pharmml.org/2013/03/CommonTypes
Annotations	This element defines a FALSE Booelan value.
Diagram	
Type	ct:FalseBooleanType
Substitution Group Affiliation	• ct:Boolean

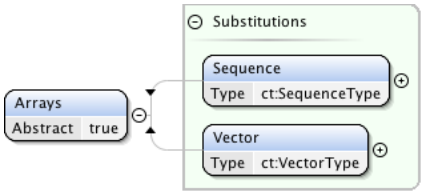
Attributes	QName	Type	Use			
	id	ct:IdType	optional			
		The element identifier.				

1.2.24 Element `ct:Name`

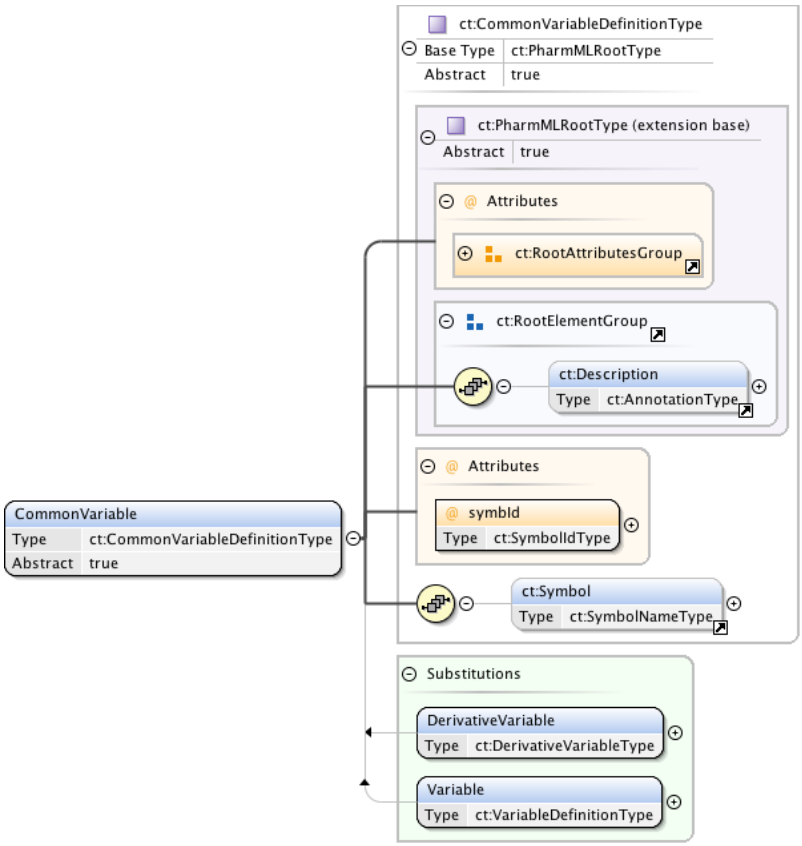
Namespace	http://www.pharmml.org/2013/03/CommonTypes
Annotations	Element defines a human readable/display name for its parent element.
Diagram	
Type	ct:NameType

Attributes	QName	Type	Use			
	id	ct:IdType	optional			
		The element identifier.				

1.2.25 Element `ct:Arrays`

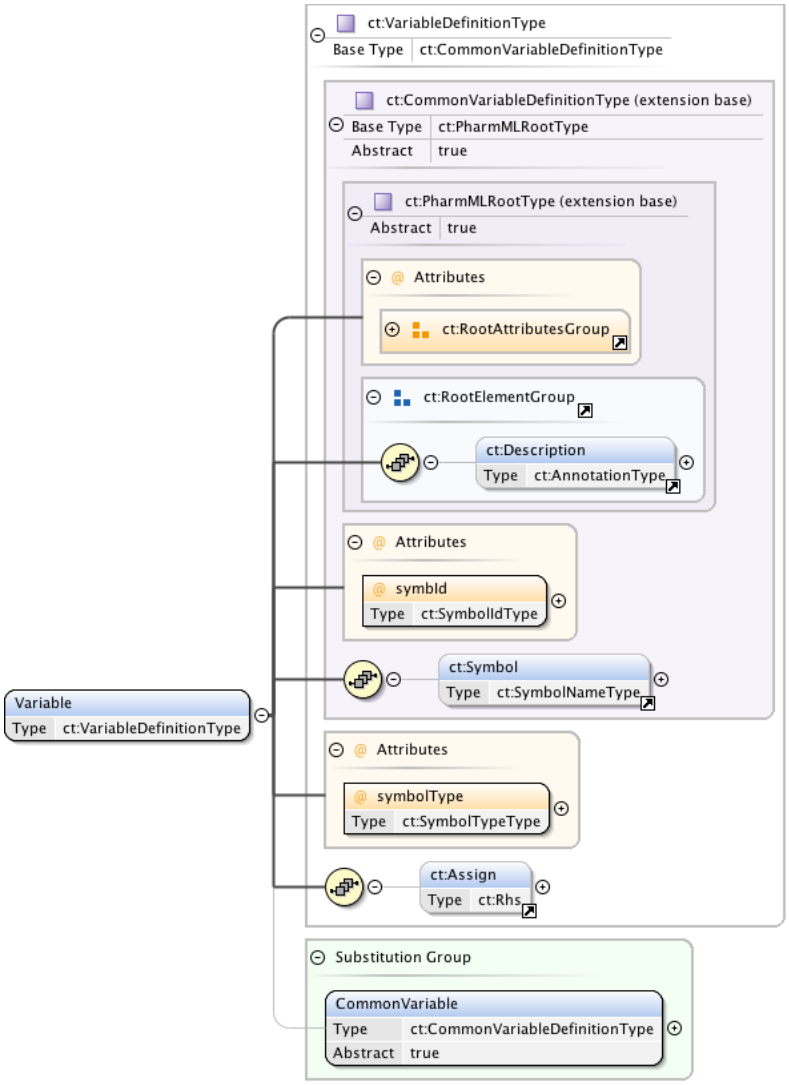
Namespace	http://www.pharmml.org/2013/03/CommonTypes
Annotations	An abstract element that defines an array of values.
Diagram	
Substitution Group	• ct:Sequence • ct:Vector

1.2.26 Element `ct:CommonVariable`

Namespace	http://www.pharmml.org/2013/03/CommonTypes
Annotations	An abstract element that defines a variety of variable declarations. The possible variable declarations are defined by the substitution group.
Diagram	
Type	ct:CommonVariableDefinitionType
Substitution Group	• ct:Variable • ct:DerivativeVariable

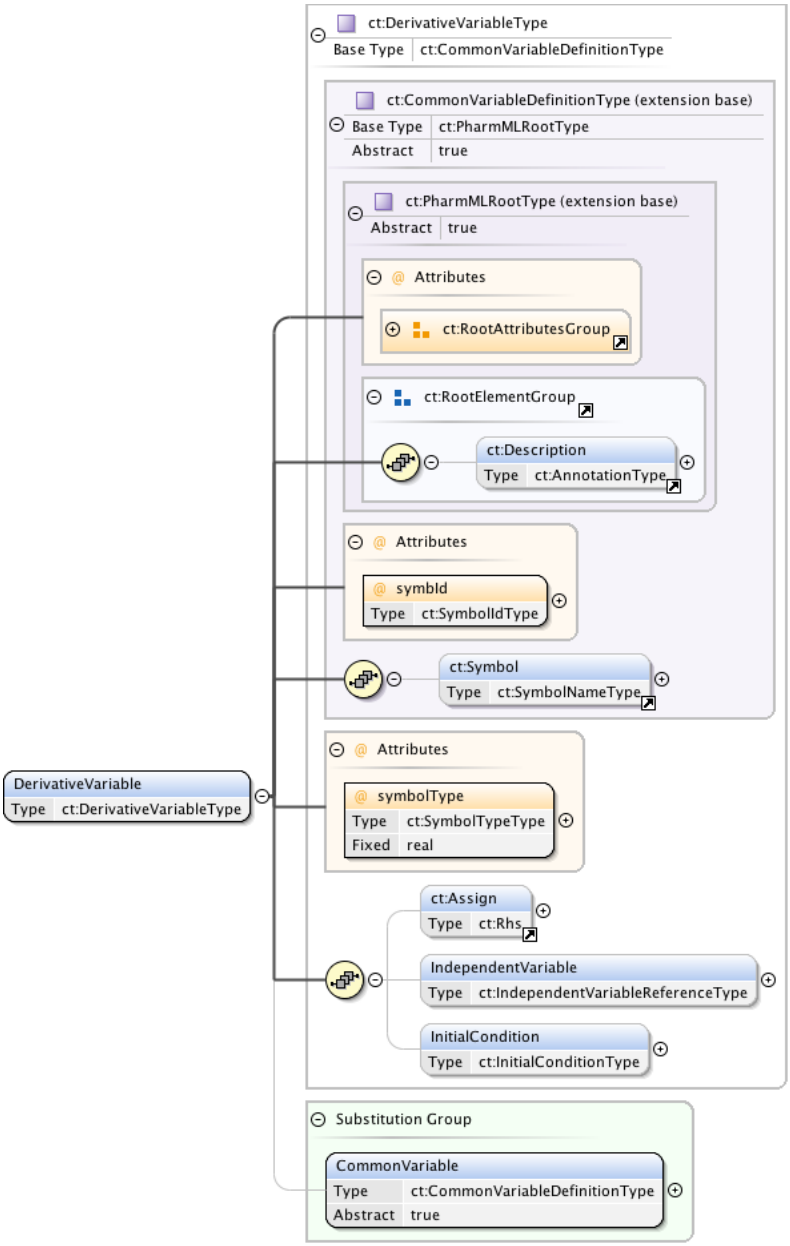
Attributes	QName	Type	Use			
	id	ct:IdType	optional			
		The element identifier.				
	symbId	ct:SymbolIdType	required			
		The symbol id used to define the variable.				

1.2.27 Element `ct:Variable`

Namespace	http://www.pharmml.org/2013/03/CommonTypes
Annotations	Element specifies a standard variable definition.
Diagram	
Type	ct:VariableDefinitionType
Substitution Group Affiliation	• ct:CommonVariable

Attributes	**QName**	**Type**	**Use**			
	id	ct:IdType	optional			
		The element identifier.				
	symbId	ct:SymbolIdType	required			
		The symbol id used to define the variable.				
	symbolType	ct:SymbolTypeType	required			
		The type of the variable.				

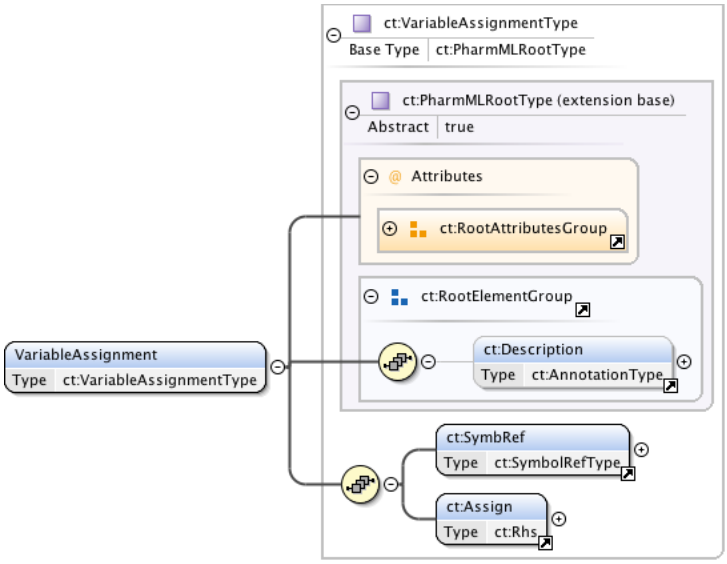
1.2.28 Element `ct:DerivativeVariable`

Namespace	http://www.pharmml.org/2013/03/CommonTypes
Annotations	Element specifies a derivative variable definition.
Diagram	
Type	ct:DerivativeVariableType

Substitution Group Affiliation	• ct:CommonVariable

Attributes	QName	Type	Fixed	Use		
	id	ct:IdType		optional		
		The element identifier.				
	symbId	ct:SymbolIdType		required		
		The symbol id used to define the variable.				
	symbolType	ct:SymbolTypeType	real	required		
		The symbol type of a derivative variable is always set to be a real.				

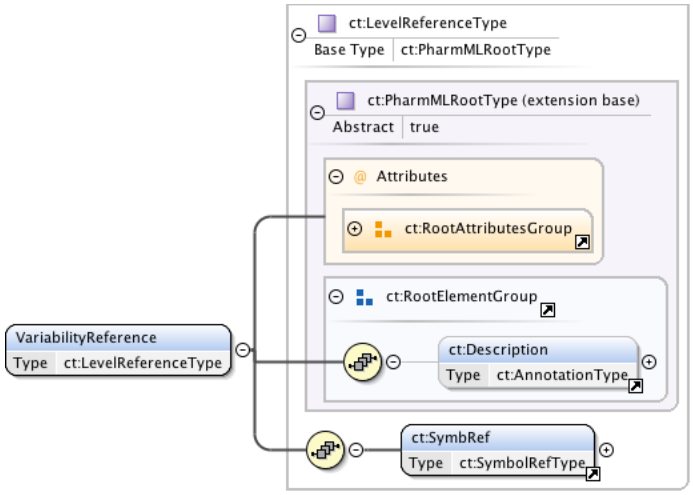
1.2.29 Element `ct:VariableAssignment`

Namespace	http://www.pharmml.org/2013/03/CommonTypes
Annotations	Element specifies a variable assignment.
Diagram	
Type	ct:VariableAssignmentType

Attributes	QName	Type	Use			
	id	ct:IdType	optional			
		The element identifier.				

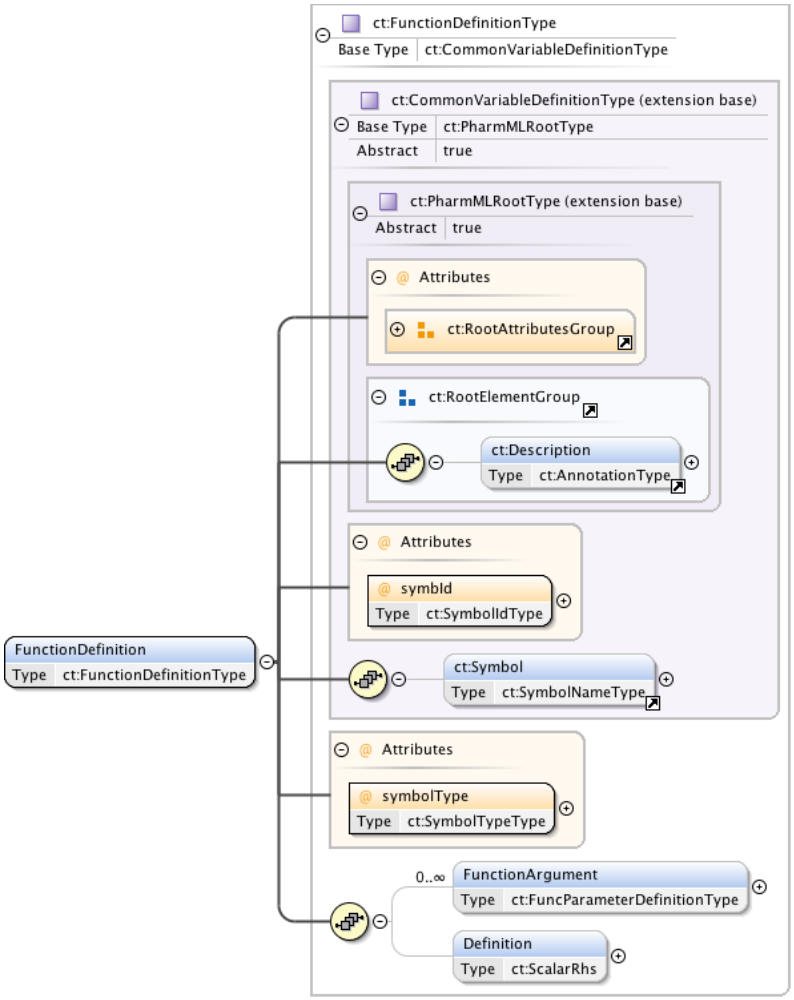
1.2.30 Element `ct:VariabilityReference`

Namespace	http://www.pharmml.org/2013/03/CommonTypes
Annotations	The element provides a reference to a variability level. It associates its parent element with the reference variability level.

Diagram	
Type	ct:LevelReferenceType
Attributes	**QName** / **Type** / **Use** **id** / ct:IdType / optional The element identifier.

1.2.31 Element `ct:FunctionDefinition`

Namespace	http://www.pharmml.org/2013/03/CommonTypes
Annotations	This element defines a function within the PharmML document.

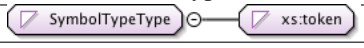
Diagram	
Type	ct:FunctionDefinitionType

Attributes

QName	Type	Use			
id	ct:IdType	optional			
	The element identifier.				
symbId	ct:SymbolIdType	required			
	The symbol id used to define the variable.				
symbolType	ct:SymbolTypeType	required			
	The type of the function definition.				

1.3 Simple Type(s)

1.3.1 Simple Type `ct:SymbolTypeType`

Namespace	http://www.pharmml.org/2013/03/CommonTypes
Annotations	Defines the symbol types. Restricted to the available types.
Diagram	
Type	restriction of xs:token

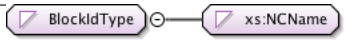
Facets

enumeration	`int`
enumeration	`real`
enumeration	`boolean`
enumeration	`string`
enumeration	`id`

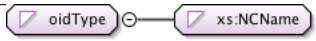
1.3.2 Simple Type ct:SymbolIdType

Namespace	http://www.pharmml.org/2013/03/CommonTypes
Annotations	Type for symbols identifiers.
Diagram	
Type	xs:NCName

1.3.3 Simple Type ct:BlockIdType

Namespace	http://www.pharmml.org/2013/03/CommonTypes
Annotations	Type for block identifiers.
Diagram	
Type	xs:NCName

1.3.4 Simple Type ct:oidType

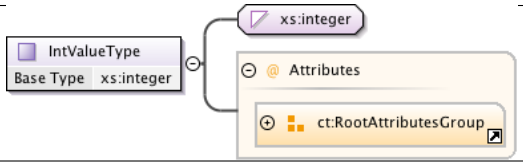
Namespace	http://www.pharmml.org/2013/03/CommonTypes
Annotations	Type for OID identifiers.
Diagram	
Type	xs:NCName

1.3.5 Simple Type ct:IdType

Namespace	http://www.pharmml.org/2013/03/CommonTypes
Annotations	Type of the element identifier.
Diagram	
Type	xs:NCName

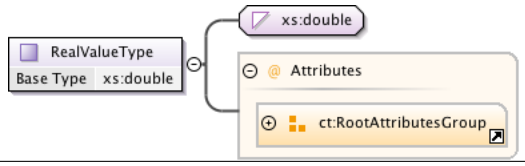
1.4 Complex Type(s)

1.4.1 Complex Type ct:IntValueType

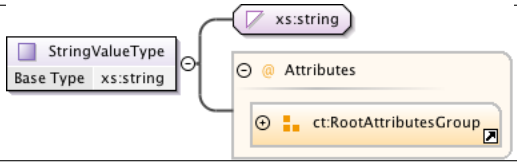
Namespace	http://www.pharmml.org/2013/03/CommonTypes
Annotations	Integer value.
Diagram	
Type	extension of xs:integer
Attributes	**QName** / **Type** / **Use**: **id** / ct:IdType / optional The element identifier.

1.4.2 Complex Type ct:RealValueType

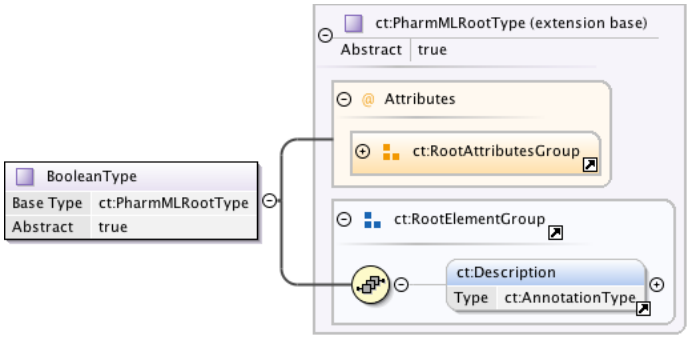
Namespace	http://www.pharmml.org/2013/03/CommonTypes
Annotations	Real value.

Diagram	
Type	extension of xs:double
Attributes	**QName** / **Type** / **Use** **id** / ct:IdType / optional The element identifier.

1.4.3 Complex Type `ct:StringValueType`

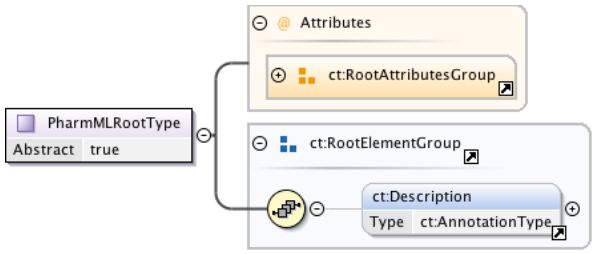
Namespace	http://www.pharmml.org/2013/03/CommonTypes
Annotations	String value.
Diagram	
Type	extension of xs:string
Attributes	**QName** / **Type** / **Use** **id** / ct:IdType / optional The element identifier.

1.4.4 Complex Type `ct:BooleanType`

Namespace	http://www.pharmml.org/2013/03/CommonTypes
Annotations	A Boolean type.
Diagram	
Type	extension of ct:PharmMLRootType
Attributes	**QName** / **Type** / **Use** **id** / ct:IdType / optional The element identifier.

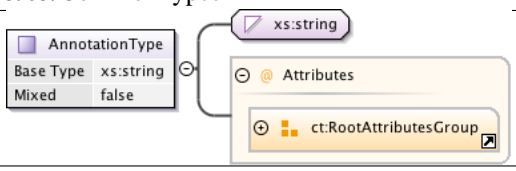
1.4.5 Complex Type `ct:PharmMLRootType`

Namespace	http://www.pharmml.org/2013/03/CommonTypes
Annotations	Root type of all elements and types defining elements in PharmML.

Diagram	

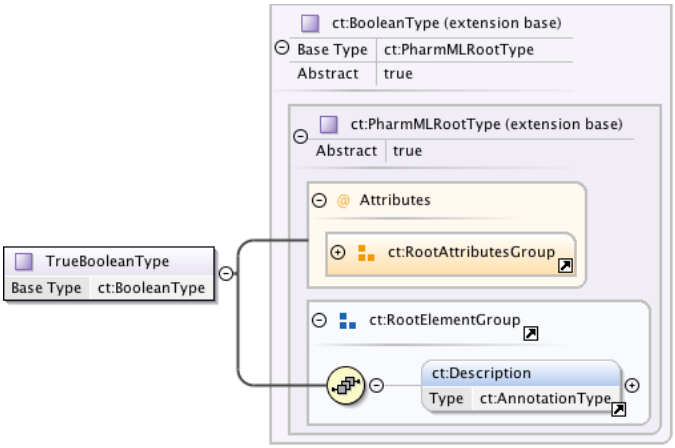
Attributes	QName	Type	Use			
	id	ct:IdType	optional			
		The element identifier.				

1.4.6 Complex Type `ct:AnnotationType`

Namespace	http://www.pharmml.org/2013/03/CommonTypes
Diagram	
Type	extension of xs:string

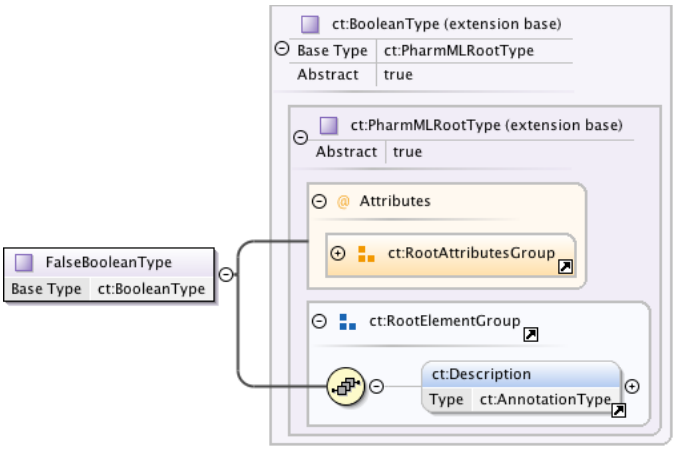
Attributes	QName	Type	Use			
	id	ct:IdType	optional			
		The element identifier.				

1.4.7 Complex Type `ct:TrueBooleanType`

Namespace	http://www.pharmml.org/2013/03/CommonTypes
Annotations	A literal Boolean value for true.
Diagram	
Type	extension of ct:BooleanType

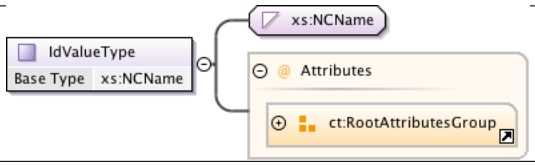
Attributes	QName	Type	Use			
	id	ct:IdType	optional			
		The element identifier.				

1.4.8 Complex Type `ct:FalseBooleanType`

Namespace	http://www.pharmml.org/2013/03/CommonTypes
Annotations	A literal Boolean value for false.
Diagram	
Type	extension of ct:BooleanType

Attributes	QName	Type	Use			
	id	ct:IdType	optional			
		The element identifier.				

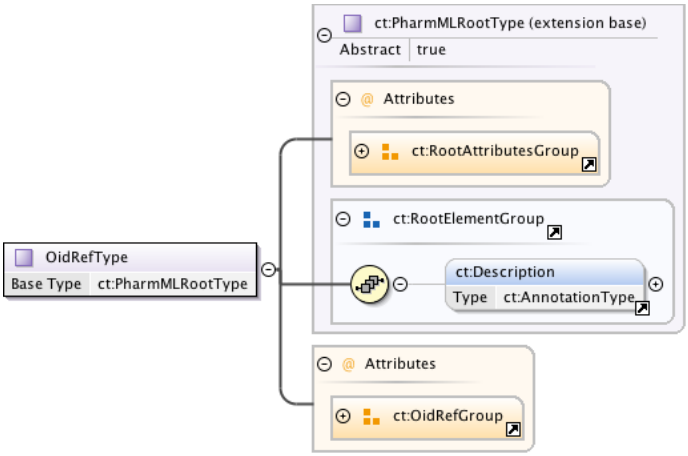
1.4.9 Complex Type `ct:IdValueType`

Namespace	http://www.pharmml.org/2013/03/CommonTypes
Annotations	A literal Id value. This has a type of `id'.
Diagram	
Type	extension of xs:NCName

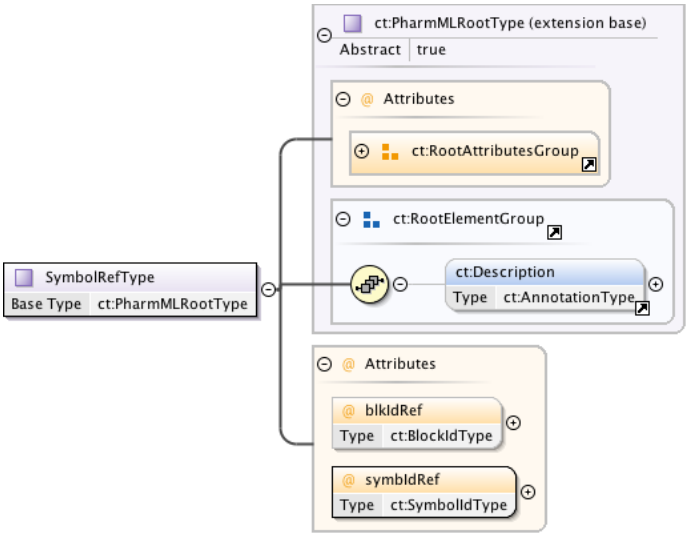
Attributes	QName	Type	Use			
	id	ct:IdType	optional			
		The element identifier.				

1.4.10 Complex Type `ct:OidRefType`

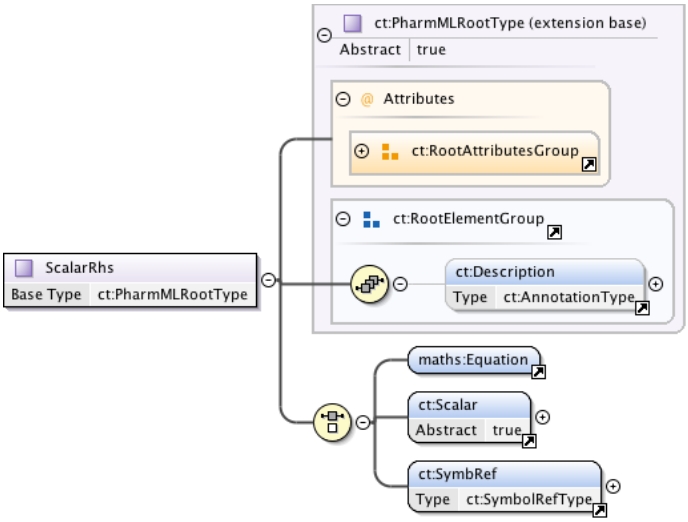
Namespace	http://www.pharmml.org/2013/03/CommonTypes
Annotations	Type used by element referencing an OID.

Diagram	
Type	extension of ct:PharmMLRootType
Attributes	**QName** / **Type** / **Use** **id** / ct:IdType / optional The element identifier. **oidRef** / ct:oidType / required

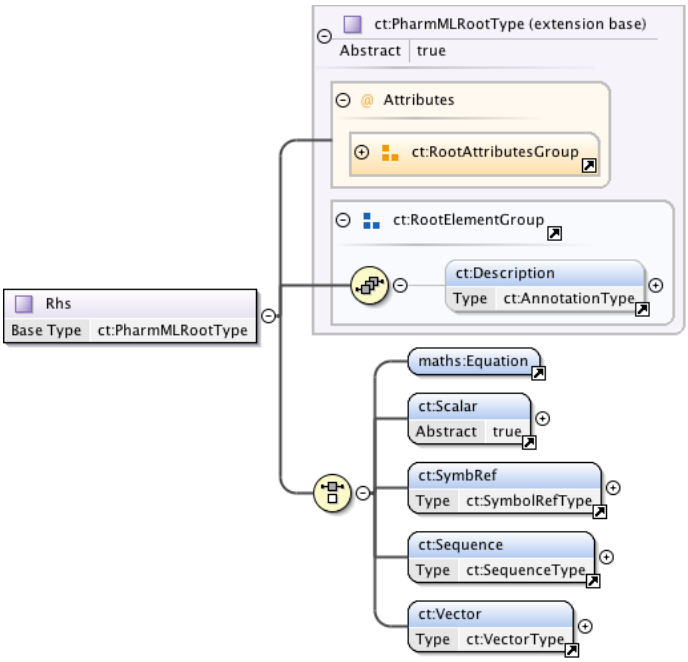
1.4.11 Complex Type `ct:SymbolRefType`

Namespace	http://www.pharmml.org/2013/03/CommonTypes
Diagram	
Type	extension of ct:PharmMLRootType
Attributes	**QName** / **Type** / **Use** **blkIdRef** / ct:BlockIdType / optional ID referencing a Block. **id** / ct:IdType / optional The element identifier. **symbIdRef** / ct:SymbolIdType / required ID referencing a Symbol.

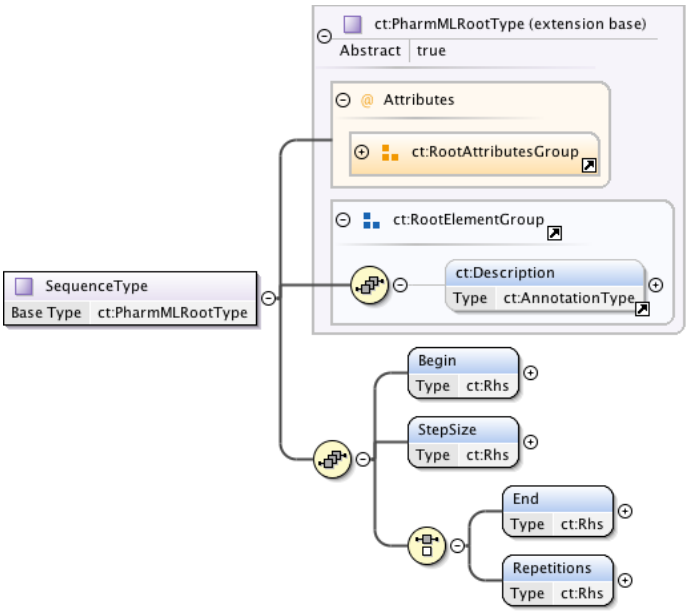
1.4.12 Complex Type `ct:ScalarRhs`

Namespace	http://www.pharmml.org/2013/03/CommonTypes
Diagram	
Type	extension of ct:PharmMLRootType
Attributes	**QName** / **Type** / **Use** **id** / ct:IdType / optional The element identifier.

1.4.13 Complex Type `ct:Rhs`

Namespace	http://www.pharmml.org/2013/03/CommonTypes
Diagram	
Type	extension of ct:PharmMLRootType
Attributes	**QName** / **Type** / **Use** **id** / ct:IdType / optional The element identifier.

1.4.14 Complex Type `ct:SequenceType`

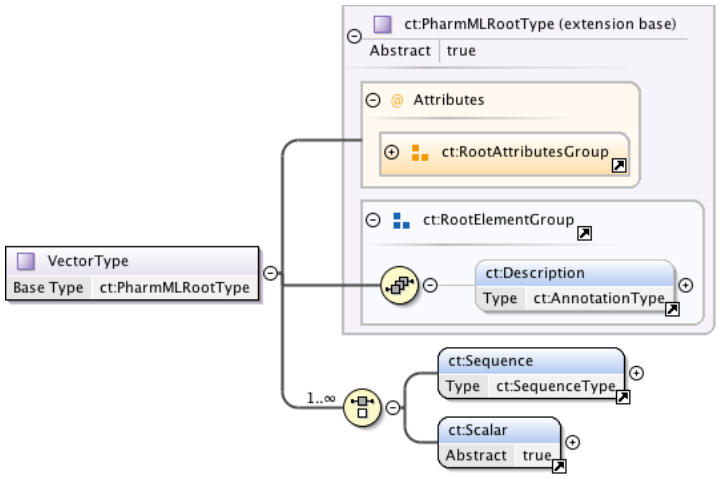
Namespace	http://www.pharmml.org/2013/03/CommonTypes
Annotations	The definition of a uniform sequence of numbers. Conceptually is has two forms. The first form takes an initial number, a step size and the last number. The sequence starts at the first number and a new value, incremented by the step size, is added to the sequence until this value exceeds the end value. In the second the step size is incremented r times, where r is the number of repetitions.
Diagram	
Type	extension of ct:PharmMLRootType

Attributes

QName	Type	Use			
id	ct:IdType	optional			
	The element identifier.				

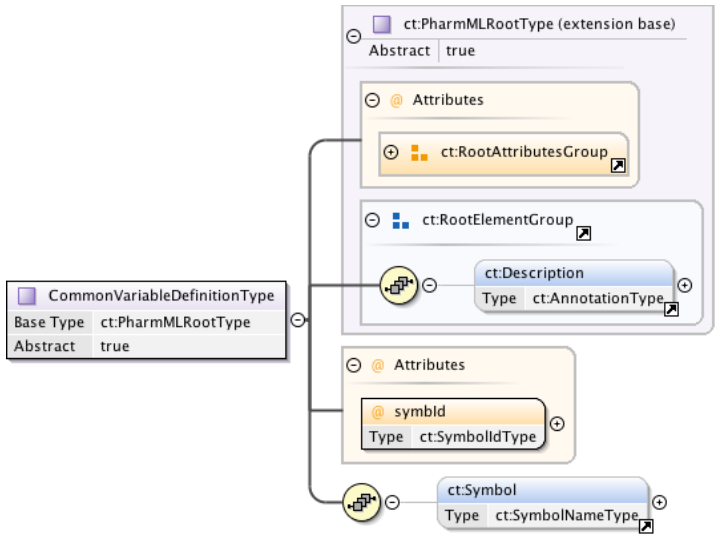
1.4.15 Complex Type `ct:VectorType`

Namespace	http://www.pharmml.org/2013/03/CommonTypes
Annotations	The definition of a non-uniform sequence of numbers. The vector is an ordered list of values. The values of the sequence element are inserted into the vector at the point of definition. For example, take the vector (the [] brackets denote a sequence): 0, 4, [0:1:3], 33. Inserting the sequence gives us the vector of values: 0, 4, 0, 1, 2, 3, 33.

Diagram	
Type	extension of ct:PharmMLRootType

Attributes	QName	Type	Use			
	id	ct:IdType	optional			
		The element identifier.				

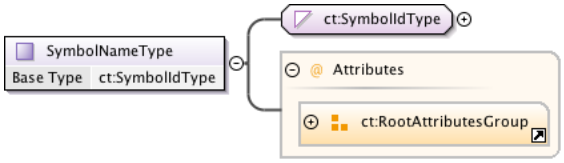
1.4.16 Complex Type `ct:CommonVariableDefinitionType`

Namespace	http://www.pharmml.org/2013/03/CommonTypes
Annotations	An abstract type defining the comment properties of a variable definition.
Diagram	
Type	extension of ct:PharmMLRootType

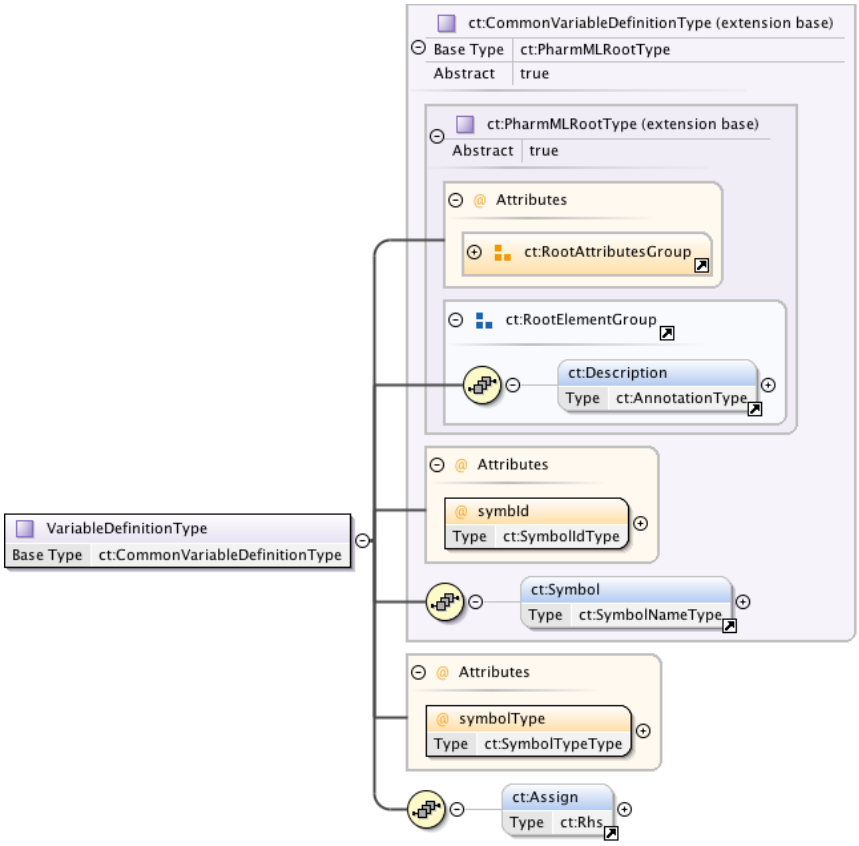
Attributes	QName	Type	Use			
	id	ct:IdType	optional			
		The element identifier.				
	symbId	ct:SymbolIdType	required			
		The symbol id used to define the variable.				

1.4.17 Complex Type `ct:SymbolNameType`

Namespace	http://www.pharmml.org/2013/03/CommonTypes

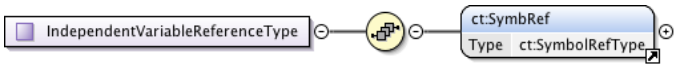
Annotations	Type defining the name of the symbol in a form suitable for display. Currently this should be plain text and not include any markup.
Diagram	
Type	extension of ct:SymbolIdType
Attributes	**QName** / **Type** / **Use** **id** / ct:IdType / optional The element identifier.

1.4.18 Complex Type `ct:VariableDefinitionType`

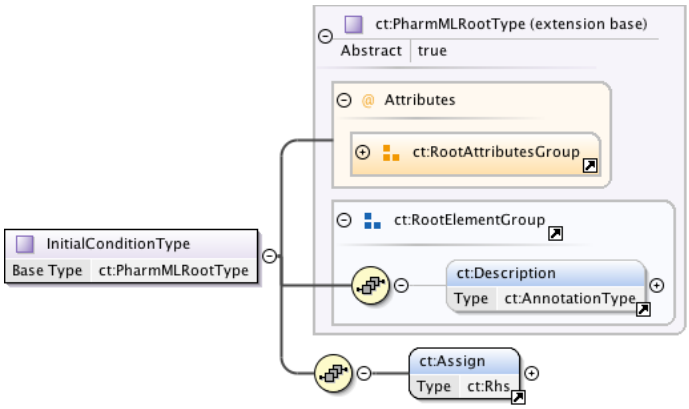
Namespace	http://www.pharmml.org/2013/03/CommonTypes
Annotations	A standard variable definition. Elements defined unsing this XML Schema Type have a Symbol type in addition to other common variable definition properties.
Diagram	
Type	extension of ct:CommonVariableDefinitionType

Attributes	**QName**	**Type**	**Use**			
	id	ct:IdType	optional			
		The element identifier.				
	symbId	ct:SymbolIdType	required			
		The symbol id used to define the variable.				
	symbolType	ct:SymbolTypeType	required			
		The type of the variable.				

1.4.19 Complex Type ct : IndependentVariableReferenceType

Namespace	http://www.pharmml.org/2013/03/CommonTypes
Annotations	References the independent variable.
Diagram	

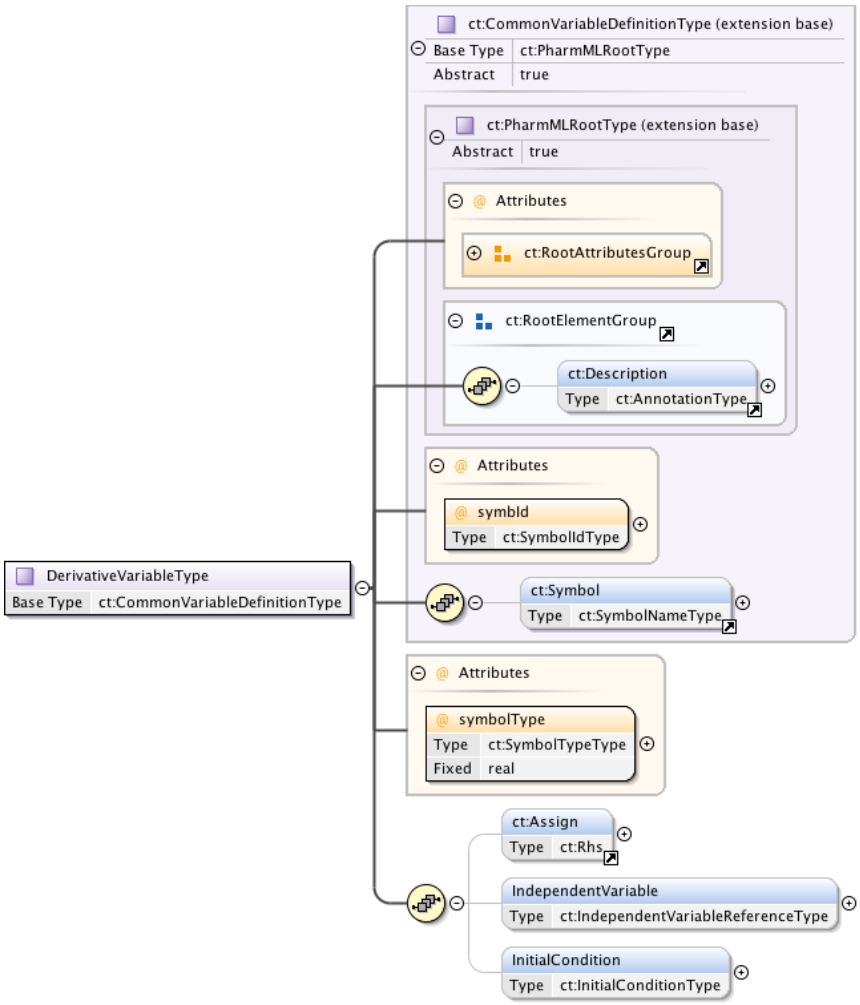
1.4.20 Complex Type ct : InitialConditionType

Namespace	http://www.pharmml.org/2013/03/CommonTypes
Annotations	The initial condition of the derivative variable.
Diagram	
Type	extension of ct:PharmMLRootType

Attributes	**QName**	**Type**	**Use**			
	id	ct:IdType	optional			
		The element identifier.				

1.4.21 Complex Type ct : DerivativeVariableType

Namespace	http://www.pharmml.org/2013/03/CommonTypes
Annotations	The type specifying a derivative variable.

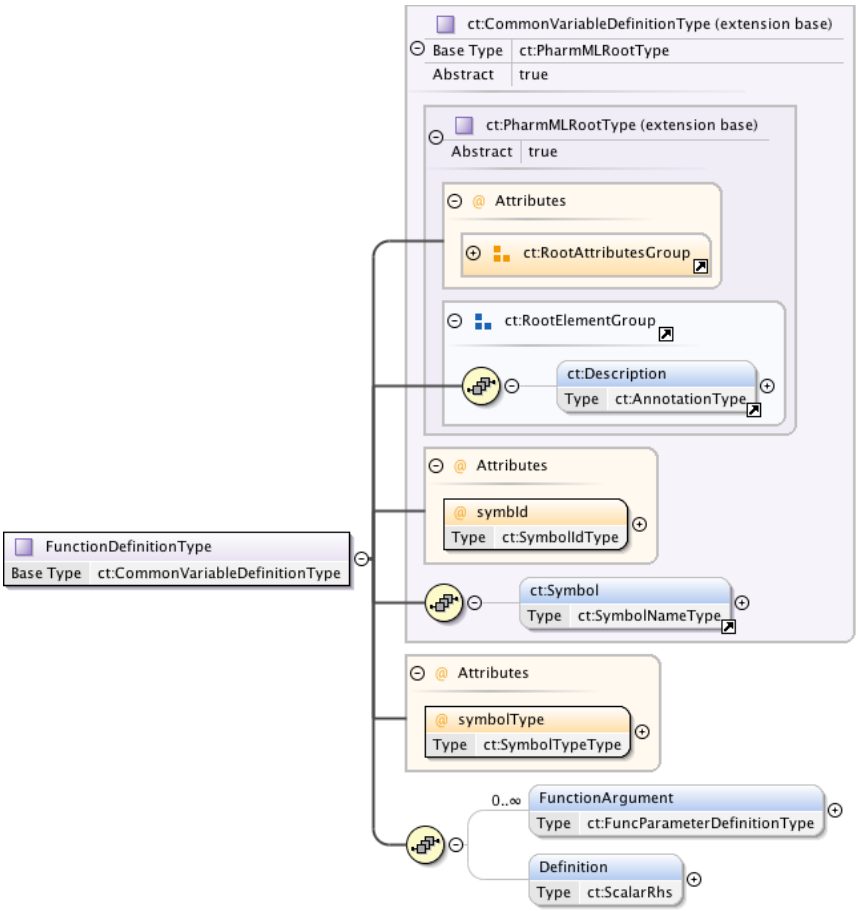
Diagram	
Type	extension of ct:CommonVariableDefinitionType

Attributes

QName	Type	Fixed	Use		
id	ct:IdType		optional		
	The element identifier.				
symbId	ct:SymbolIdType		required		
	The symbol id used to define the variable.				
symbolType	ct:SymbolTypeType	real	required		
	The symbol type of a derivative variable is always set to be a real.				

1.4.22 Complex Type `ct:FunctionDefinitionType`

Namespace	http://www.pharmml.org/2013/03/CommonTypes
Annotations	This defines a function that can be used elsewhere in the PharmML document.

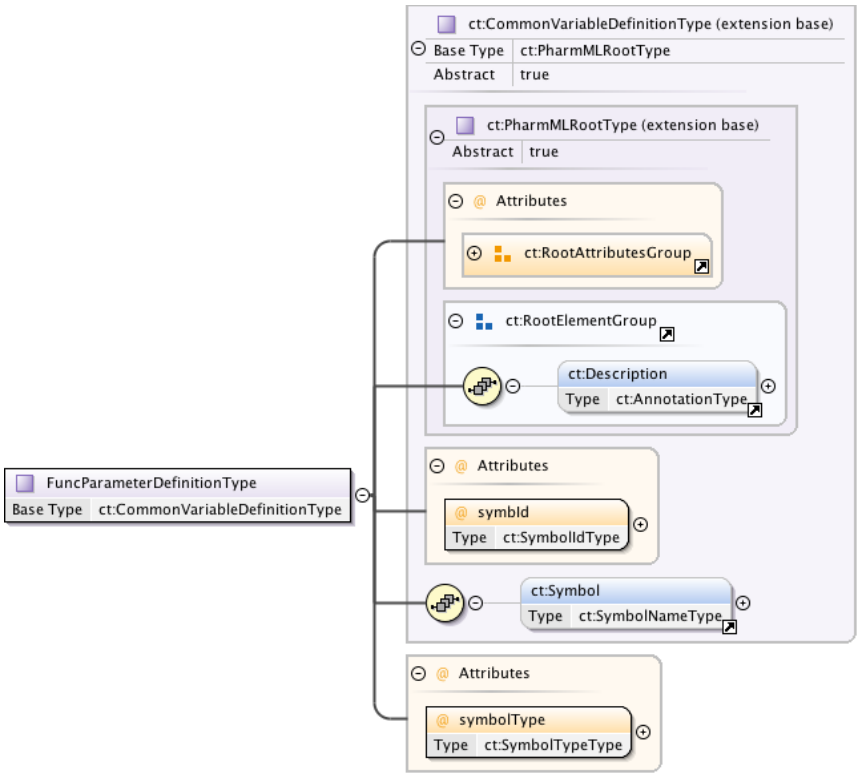
Diagram	
Type	extension of ct:CommonVariableDefinitionType

Attributes

QName	Type	Use			
id	ct:IdType	optional			
	The element identifier.				
symbId	ct:SymbolIdType	required			
	The symbol id used to define the variable.				
symbolType	ct:SymbolTypeType	required			
	The type of the function definition.				

1.4.23 Complex Type `ct:FuncParameterDefinitionType`

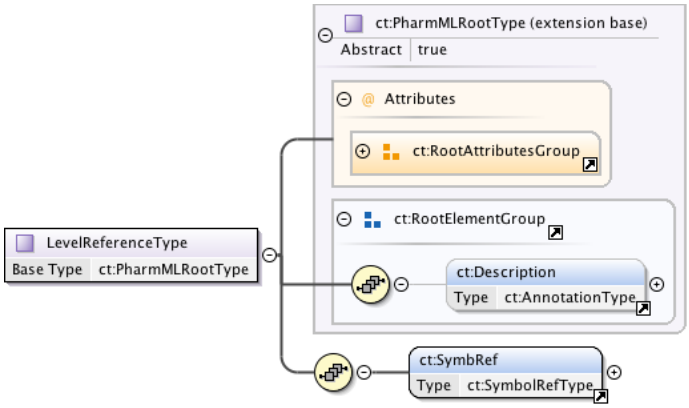
Namespace	http://www.pharmml.org/2013/03/CommonTypes
Annotations	Defines a function argument defintion type. The function argument has a symbol identifier, an optional name and a type.

Diagram	
Type	extension of ct:CommonVariableDefinitionType

Attributes

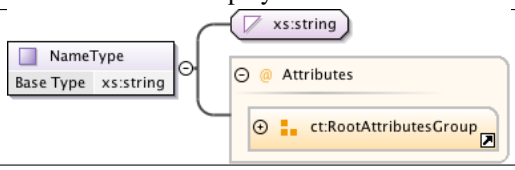
QName	Type	Use			
id	ct:IdType	optional			
	The element identifier.				
symbId	ct:SymbolIdType	required			
	The symbol id used to define the variable.				
symbolType	ct:SymbolTypeType	required			
	The type of the function definition.				

1.4.24 Complex Type `ct:LevelReferenceType`

Namespace	http://www.pharmml.org/2013/03/CommonTypes
Annotations	A reference to a variability level.
Diagram	
Type	extension of ct:PharmMLRootType

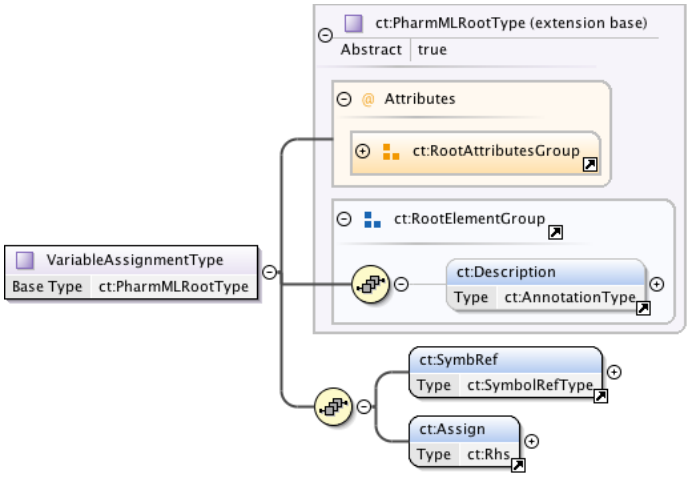
Attributes	**QName**	**Type**	**Use**			
	id	ct:IdType	optional			
		The element identifier.				

1.4.25 Complex Type `ct:NameType`

Namespace	http://www.pharmml.org/2013/03/CommonTypes
Annotations	Type specifying a descriptive name that can be displayed and so sshould be human readable.
Diagram	
Type	extension of xs:string

Attributes	**QName**	**Type**	**Use**			
	id	ct:IdType	optional			
		The element identifier.				

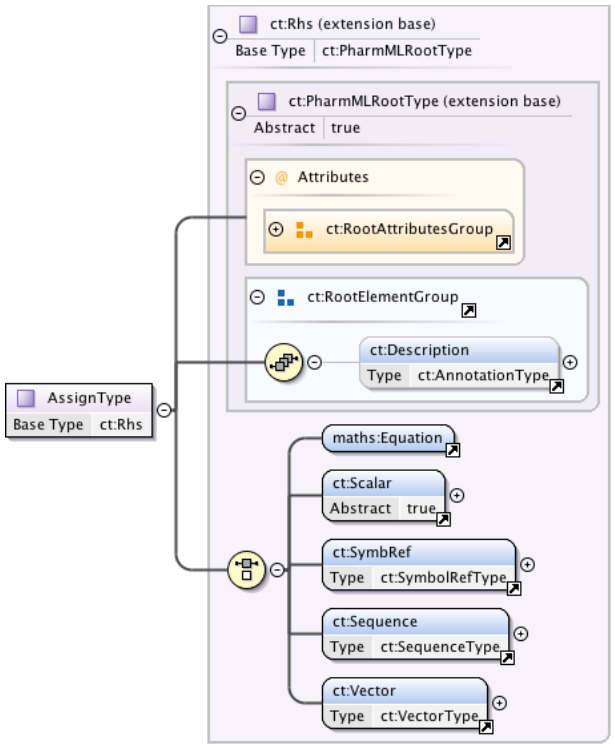
1.4.26 Complex Type `ct:VariableAssignmentType`

Namespace	http://www.pharmml.org/2013/03/CommonTypes
Annotations	A type that specifies a variable assignment.
Diagram	
Type	extension of ct:PharmMLRootType

Attributes	**QName**	**Type**	**Use**			
	id	ct:IdType	optional			
		The element identifier.				

1.4.27 Complex Type `ct:AssignType`

Namespace	http://www.pharmml.org/2013/03/CommonTypes
Annotations	The type specifies an assignment of a value(s) or equation to another element in the PharmML document.

Diagram	
Type	extension of ct:Rhs
Attributes	**QName** / **Type** / **Use** **id** / ct:IdType / optional The element identifier.

2 Namespace: ""

2.1 Attribute(s)

2.1.1 Attribute `ct:SymbolRefType /@blkIdRef`

Namespace	No namespace
Annotations	ID referencing a Block.
Type	ct:BlockIdType

2.1.2 Attribute `ct:SymbolRefType /@symbIdRef`

Namespace	No namespace
Annotations	ID referencing a Symbol.
Type	ct:SymbolIdType

2.1.3 Attribute `ct:CommonVariableDefinitionType /@symbId`

Namespace	No namespace
Annotations	The symbol id used to define the variable.
Type	ct:SymbolIdType

2.1.4 Attribute `ct:VariableDefinitionType /@symbolType`

Namespace	No namespace
Annotations	The type of the variable.
Type	ct:SymbolTypeType
Facets	enumeration `int` enumeration `real` enumeration `boolean` enumeration `string` enumeration `id`

2.1.5 Attribute `ct:DerivativeVariableType /@symbolType`

Namespace	No namespace
Annotations	The symbol type of a derivative variable is always set to be a real.
Type	ct:SymbolTypeType
Facets	enumeration `int` enumeration `real` enumeration `boolean` enumeration `string` enumeration `id`

2.1.6 Attribute `ct:FuncParameterDefinitionType /@symbolType`

Namespace	No namespace
Annotations	The type of the function definition.
Type	ct:SymbolTypeType
Facets	enumeration `int` enumeration `real` enumeration `boolean` enumeration `string` enumeration `id`

2.1.7 Attribute `ct:FunctionDefinitionType /@symbolType`

Namespace	No namespace
Annotations	The type of the function definition.
Type	ct:SymbolTypeType
Facets	enumeration `int` enumeration `real` enumeration `boolean` enumeration `string` enumeration `id`

11.6 Dataset

1 Namespace: "http://www.pharmml.org/2013/08/Dataset"

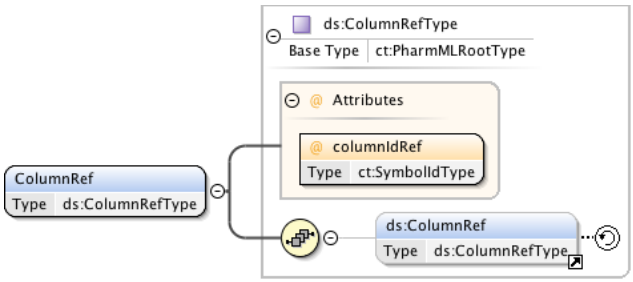
1.1 Schema(s)

1.1.1 Main schema `dataset.xsd`

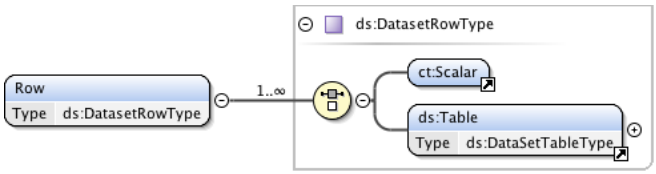
Namespace	http://www.pharmml.org/2013/08/Dataset
Annotations	The schema defines the dataset and it related structures used in a PharmML document.

1.2 Element(s)

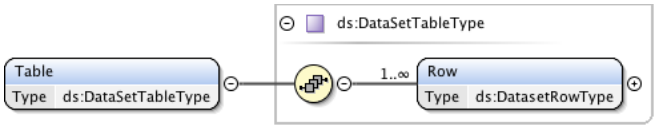
1.2.1 Element `ColumnRef`

Namespace	http://www.pharmml.org/2013/08/Dataset
Annotations	Element to a column in a dataset or nested table.
Diagram	
Type	ColumnRefType

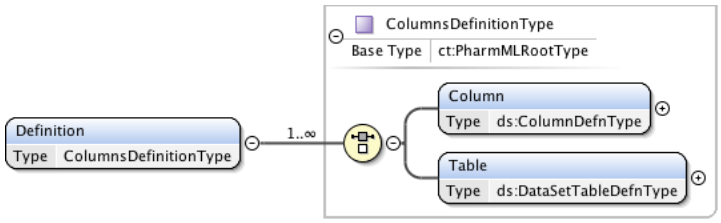
1.2.2 Element `DataSetTableType /Row`

Namespace	http://www.pharmml.org/2013/08/Dataset
Annotations	A row in the dataset.
Diagram	
Type	DatasetRowType

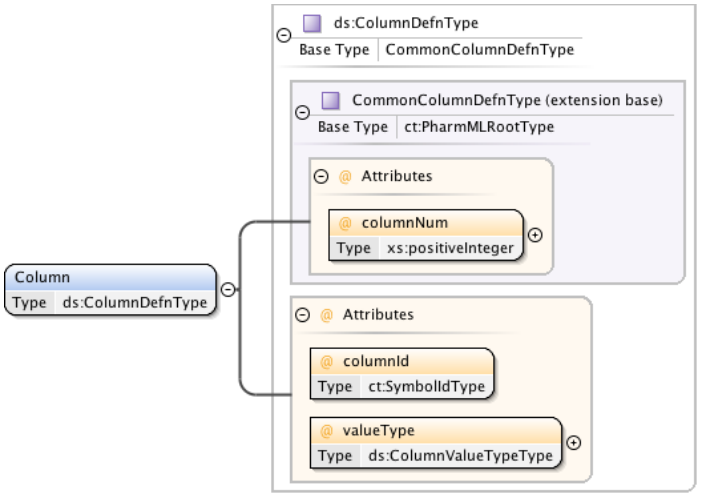
1.2.3 Element `Table`

Namespace	http://www.pharmml.org/2013/08/Dataset
Annotations	Element specifies the content of the dataset or nested table.
Diagram	
Type	DataSetTableType

1.2.4 Element `Definition`

Namespace	http://www.pharmml.org/2013/08/Dataset
Annotations	Defines the columns and nested tables in a dataset or nested table.
Diagram	
Type	ColumnsDefinitionType

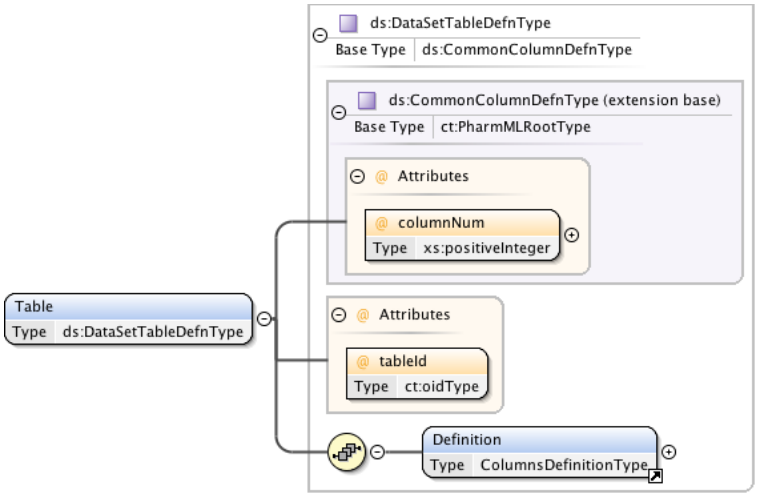
1.2.5 Element `ColumnsDefinitionType /Column`

Namespace	http://www.pharmml.org/2013/08/Dataset
Annotations	Defines a column in the dataset.
Diagram	
Type	ColumnDefnType

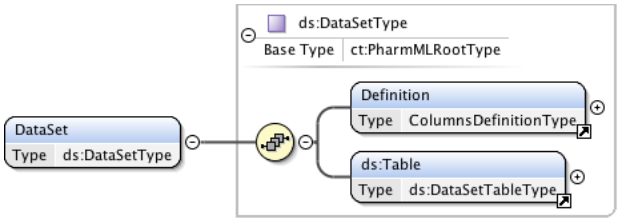
Attributes	QName	Type	Use			
	columnId	SymbolIdType	required			
		The name to give the column.				
	valueType	ColumnValueTypeType	required			
		The column number in the reosurce to use for this column.				

1.2.6 Element `ColumnsDefinitionType /Table`

Namespace	http://www.pharmml.org/2013/08/Dataset
Annotations	Defines a nested table.

Diagram	
Type	DataSetTableDefnType
Attributes	**QName** **Type** **Use** **tableId** oidType required The identifier of the nested table column.

1.2.7 Element `DataSet`

Namespace	http://www.pharmml.org/2013/08/Dataset
Annotations	This element specifies a dataset in PharmML. Its children define its structure and the data associate with it. More information about the dataset can be found in the Language Overview chapter in the PharmML specification.
Diagram	
Type	DataSetType

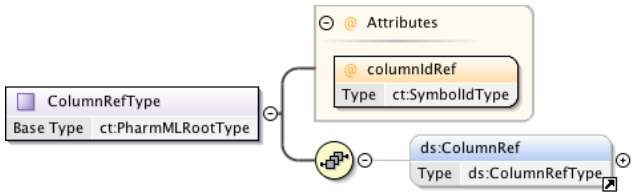
1.3 Simple Type(s)

1.3.1 Simple Type `ColumnValueTypeType`

Namespace	http://www.pharmml.org/2013/08/Dataset
Annotations	Type specifying the permitted column types.
Diagram	
Type	SymbolTypeType

1.4 Complex Type(s)

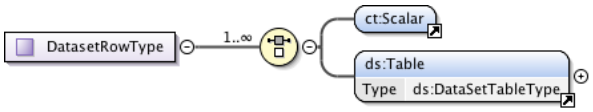
1.4.1 Complex Type `ColumnRefType`

Namespace	http://www.pharmml.org/2013/08/Dataset
Annotations	Type specified a reference to a column in a dataset.
Diagram	
Type	extension of PharmMLRootType

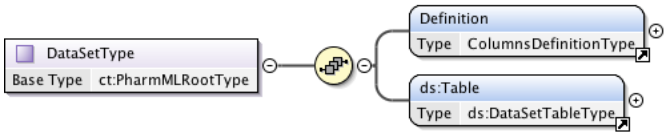
1.4.2 Complex Type `DataSetTableType`

Namespace	http://www.pharmml.org/2013/08/Dataset
Annotations	The type specifies the content of a dataset or nested table.
Diagram	

1.4.3 Complex Type `DatasetRowType`

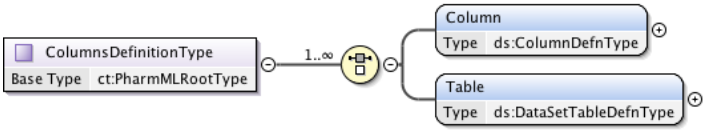
Namespace	http://www.pharmml.org/2013/08/Dataset
Annotations	This type specifies a row of values in the dataset. The row contains a cell which is a scalar value, null or a nested table.
Diagram	

1.4.4 Complex Type `DataSetType`

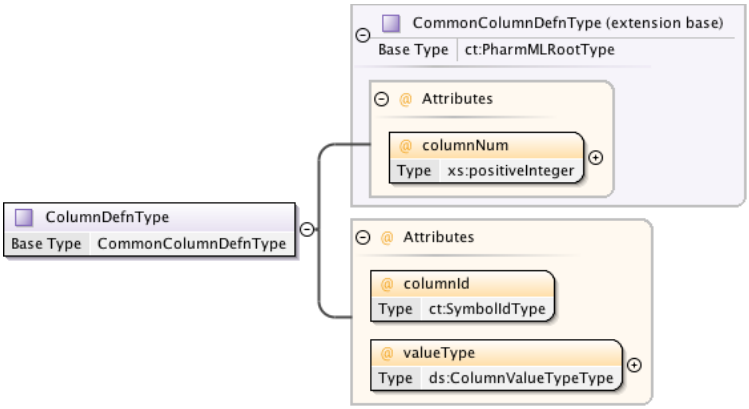
Namespace	http://www.pharmml.org/2013/08/Dataset
Annotations	The type specifying the dataset. The dataset is decribed in more detail in the Language Overview section of the specification.
Diagram	
Type	extension of PharmMLRootType

1.4.5 Complex Type `ColumnsDefinitionType`

Namespace	http://www.pharmml.org/2013/08/Dataset
Annotations	Type specifies all the columns in a dataset or nested table.

Diagram	
Type	extension of PharmMLRootType

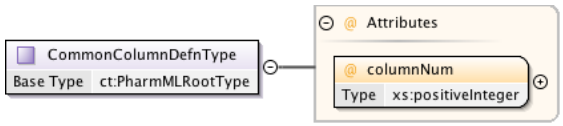
1.4.6 Complex Type `ColumnDefnType`

Namespace	http://www.pharmml.org/2013/08/Dataset
Diagram	
Type	extension of CommonColumnDefnType

Attributes

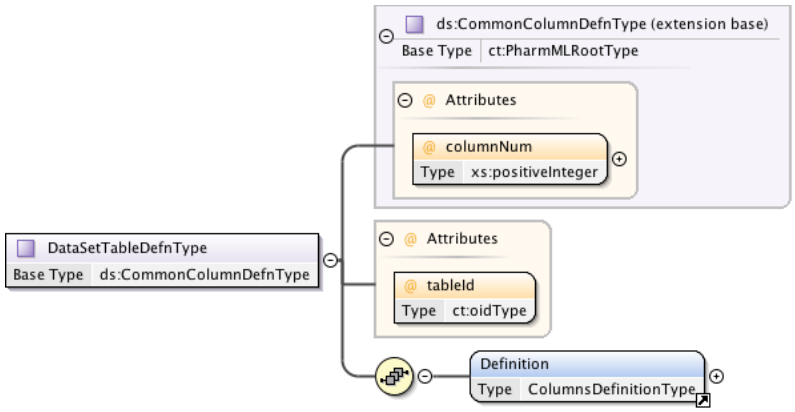
QName	Type	Use			
columnId	SymbolIdType	required			
	The name to give the column.				
valueType	ColumnValueTypeType	required			
	The column number in the reosurce to use for this column.				

1.4.7 Complex Type `CommonColumnDefnType`

Namespace	http://www.pharmml.org/2013/08/Dataset
Diagram	
Type	extension of PharmMLRootType

1.4.8 Complex Type `DataSetTableDefnType`

Namespace	http://www.pharmml.org/2013/08/Dataset
Annotations	Type that specifies the definition of a nested table.

Diagram	
Type	extension of CommonColumnDefnType
Attributes	**QName** / **Type** / **Use** **tableId** / oidType / required The identifier of the nested table column.

2 Namespace: ""

2.1 Attribute(s)

2.1.1 Attribute `ColumnRefType /@columnIdRef`

Namespace	No namespace
Annotations	Refers to a column in a dataset. This can be a column or a nested table.
Type	SymbolIdType

2.1.2 Attribute `CommonColumnDefnType /@columnNum`

Namespace	No namespace
Annotations	The column number in the reosurce to use for this column.
Type	xs:positiveInteger

2.1.3 Attribute `ColumnDefnType /@columnId`

Namespace	No namespace
Annotations	The name to give the column.
Type	SymbolIdType

2.1.4 Attribute `ColumnDefnType /@valueType`

Namespace	No namespace
Annotations	The column number in the reosurce to use for this column.
Type	ColumnValueTypeType

2.1.5 Attribute `DataSetTableDefnType /@tableId`

Namespace	No namespace
Annotations	The identifier of the nested table column.
Type	oidType

11.7 Maths

1 Namespace: "http://www.pharmml.org/2013/03/Maths"

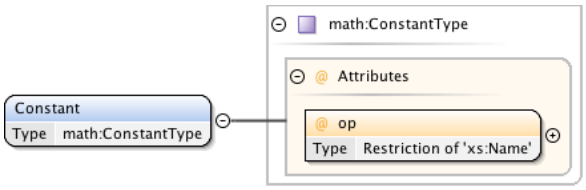
1.1 Schema(s)

1.1.1 Main schema `maths.xsd`

Namespace	http://www.pharmml.org/2013/03/Maths

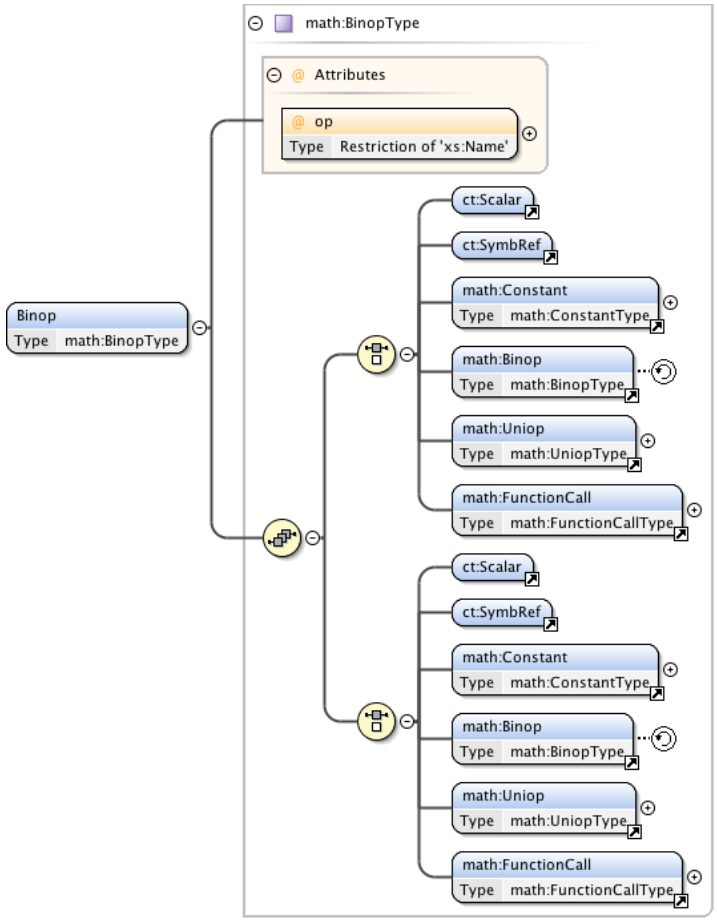
1.2 Element(s)

1.2.1 Element `math:Constant`

Namespace	http://www.pharmml.org/2013/03/Maths
Annotations	A constant symbol.
Diagram	
Type	math:ConstantType
Attributes	**QName** / **Type** / **Use** **op** / restriction of xs:Name / required The type of constant.

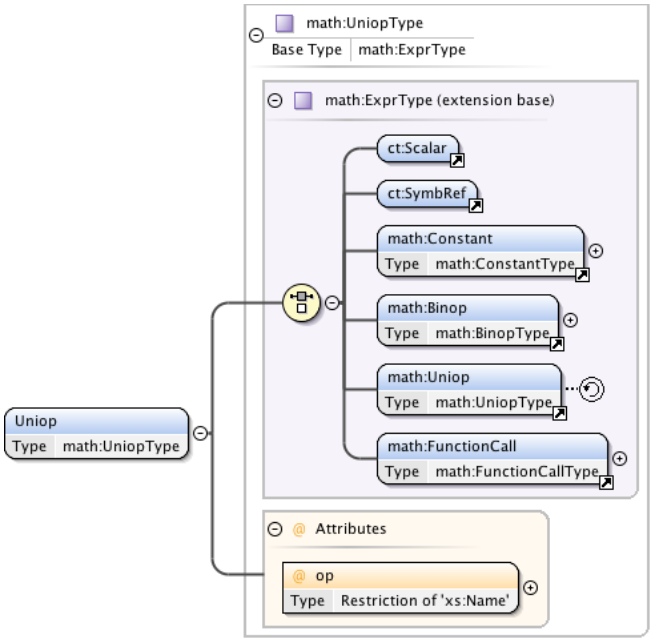
1.2.2 Element `math:Binop`

Namespace	http://www.pharmml.org/2013/03/Maths
Annotations	A binary operator.

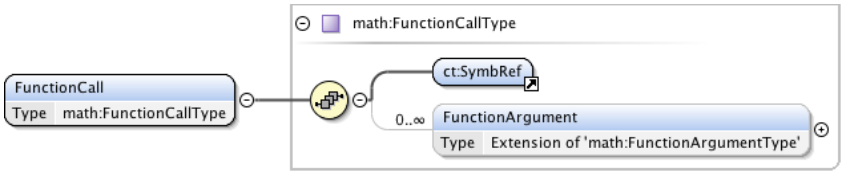
Diagram	
Type	math:BinopType
Attributes	**QName** / **Type** / **Use** **op** / restriction of xs:Name / required The binary operator type. See the specification for a more detailed description.

1.2.3 Element `math:Uniop`

Namespace	http://www.pharmml.org/2013/03/Maths
Annotations	A unary operator.

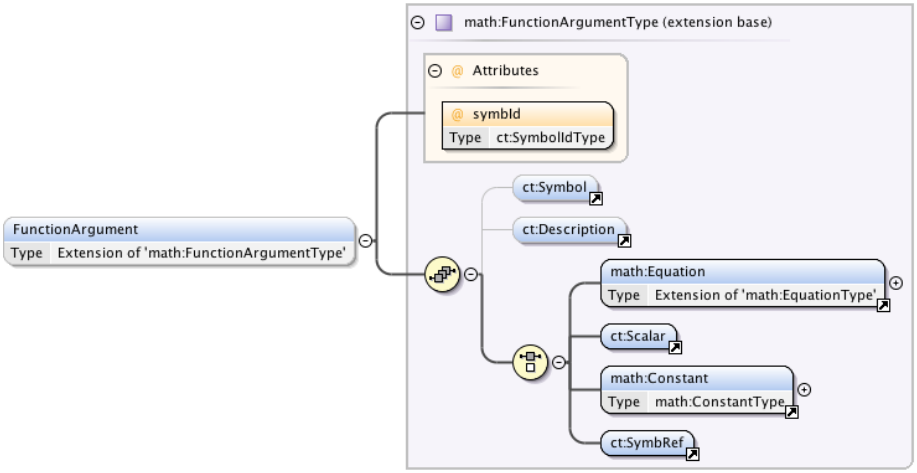
Diagram	
Type	math:UniopType
Attributes	**QName** / **Type** / **Use** **op** / restriction of xs:Name / required The operator. More detail in the specification.

1.2.4 Element `math:FunctionCall`

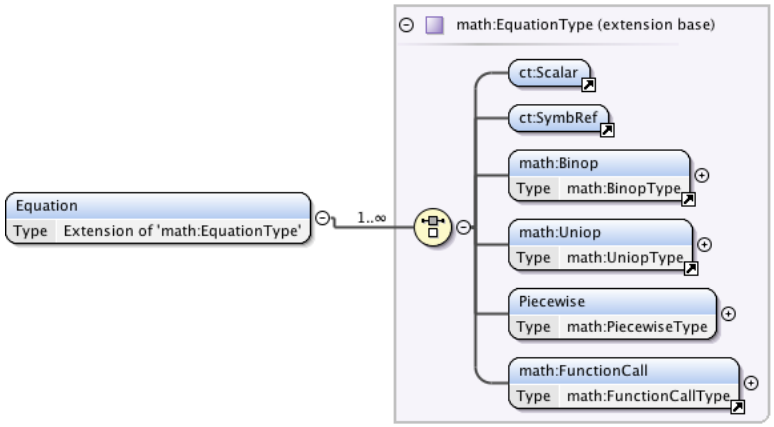
Namespace	http://www.pharmml.org/2013/03/Maths
Annotations	A function call.
Diagram	
Type	math:FunctionCallType

1.2.5 Element `math:FunctionCallType /math:FunctionArgument`

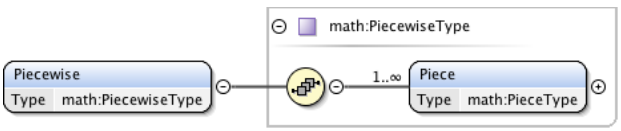
Namespace	http://www.pharmml.org/2013/03/Maths
Annotations	An argument of the function.

Diagram	
Type	extension of math:FunctionArgumentType
Attributes	**QName** / **symbId**: **Type** SymbolIdType; **Use** required; The symbold ID of the argument.

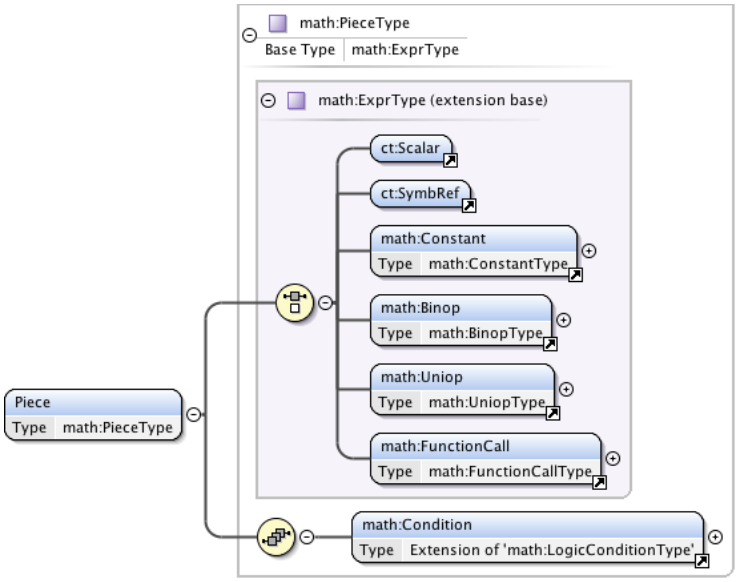
1.2.6 Element `math:Equation`

Namespace	http://www.pharmml.org/2013/03/Maths
Annotations	A mathematical expression that is evaluated to a scalar type.
Diagram	
Type	extension of math:EquationType

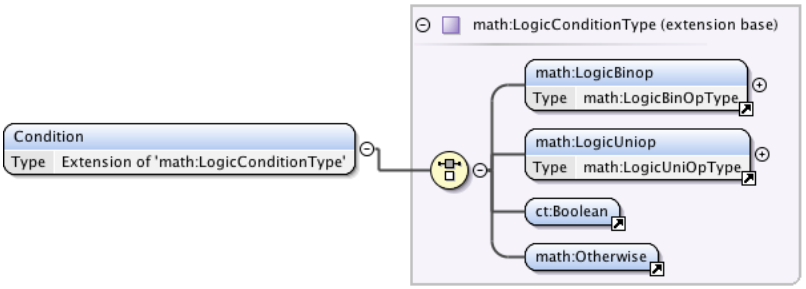
1.2.7 Element `math:EquationType /math:Piecewise`

Namespace	http://www.pharmml.org/2013/03/Maths
Annotations	Defines a piecewise expression.
Diagram	
Type	math:PiecewiseType

1.2.8 Element `math:PiecewiseType /math:Piece`

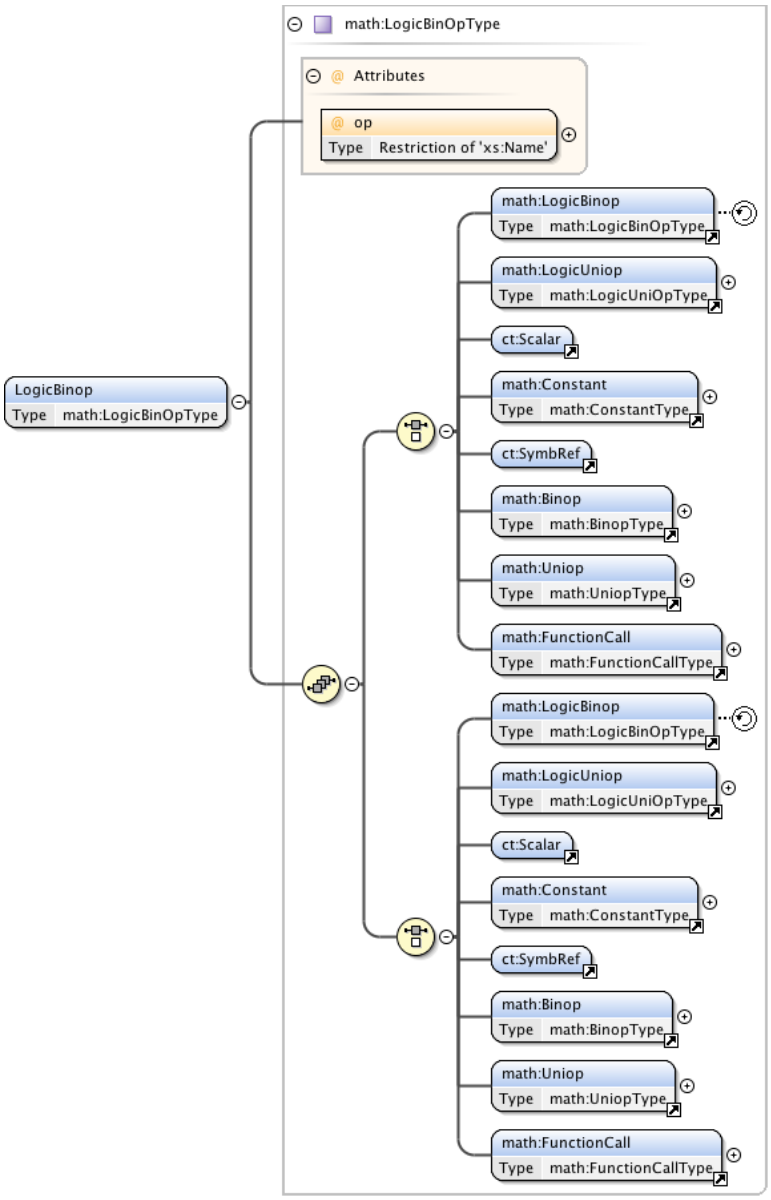
Namespace	http://www.pharmml.org/2013/03/Maths
Annotations	Defines a piece in the piecewise function.
Diagram	
Type	math:PieceType

1.2.9 Element `math:Condition`

Namespace	http://www.pharmml.org/2013/03/Maths
Annotations	A condition defined by a logical expression. Can be evaluated to True or False.
Diagram	
Type	extension of math:LogicConditionType

1.2.10 Element `math:LogicBinop`

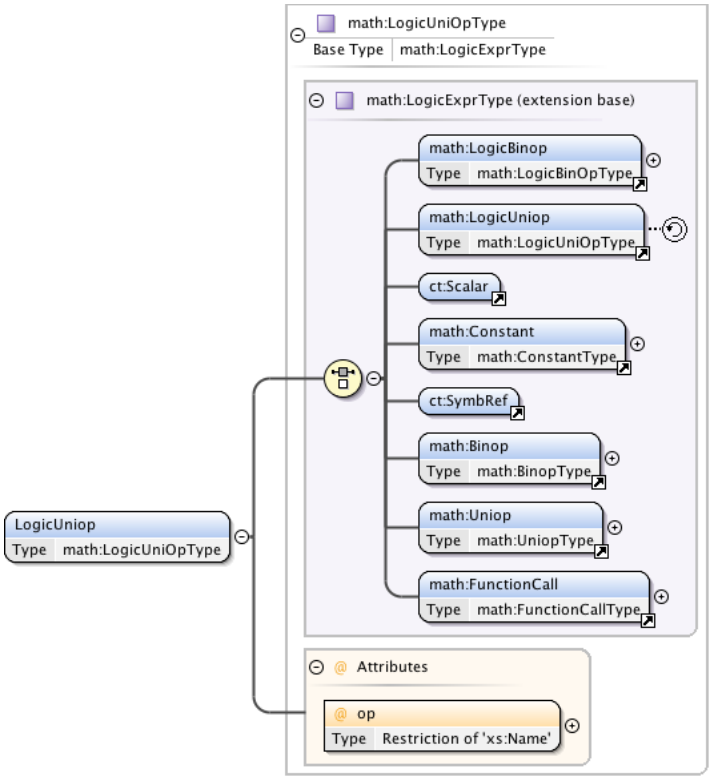
Namespace	http://www.pharmml.org/2013/03/Maths
Annotations	A logical binary operator used in logical expressions.

Diagram	
Type	math:LogicBinOpType

Attributes	QName	Type	Use			
	op	restriction of xs:Name	required			
		The logical binary operator type.				

1.2.11 Element `math:LogicUniop`

Namespace	http://www.pharmml.org/2013/03/Maths
Annotations	A logical unary operator used in logical expressions.

Diagram	
Type	math:LogicUniOpType
Attributes	**QName** / **Type** / **Use**: **op** / restriction of xs:Name / required The unary operator type.

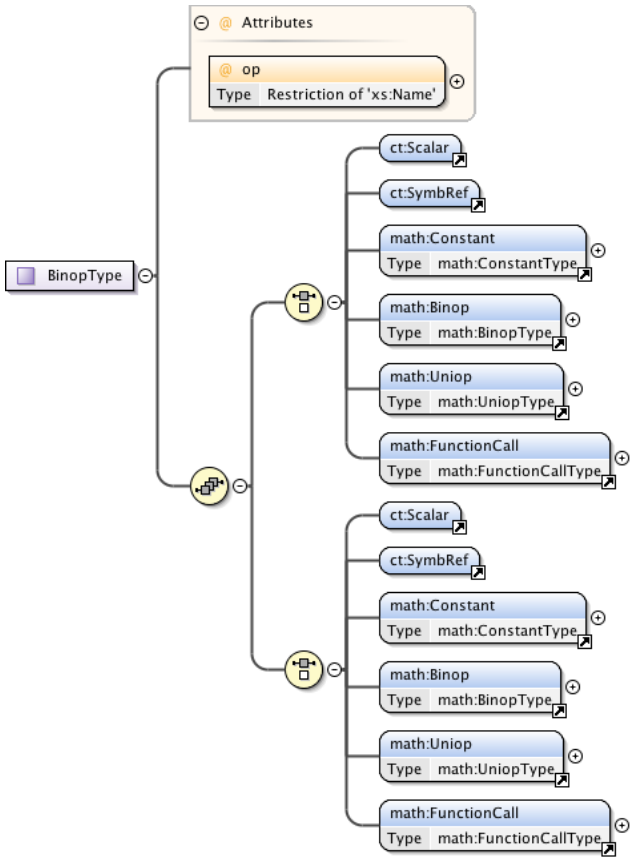
1.2.12 Element `math:Otherwise`

Namespace	http://www.pharmml.org/2013/03/Maths
Annotations	The otherwise case in a piecewise function.
Diagram	

1.3 Complex Type(s)

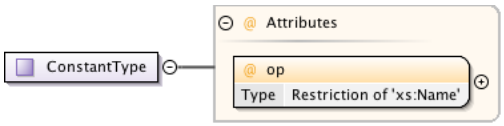
1.3.1 Complex Type `math:BinopType`

Namespace	http://www.pharmml.org/2013/03/Maths
Annotations	A binary operator describing a numerical operation. Takes two operands (as you would expect).

Diagram	

Attributes	QName	Type	Use			
	op	restriction of xs:Name	required			
		The binary operator type. See the specification for a more detailed description.				

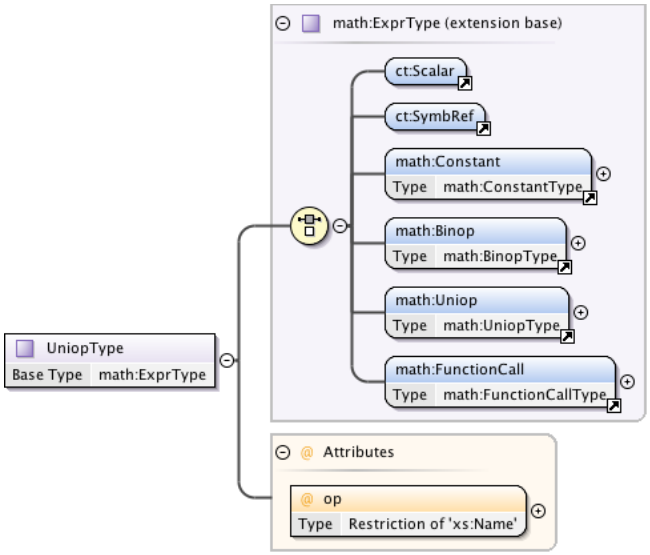
1.3.2 Complex Type `math:ConstantType`

Namespace	http://www.pharmml.org/2013/03/Maths
Annotations	The schema type defining a mathematical constant.
Diagram	

Attributes	QName	Type	Use			
	op	restriction of xs:Name	required			
		The type of constant.				

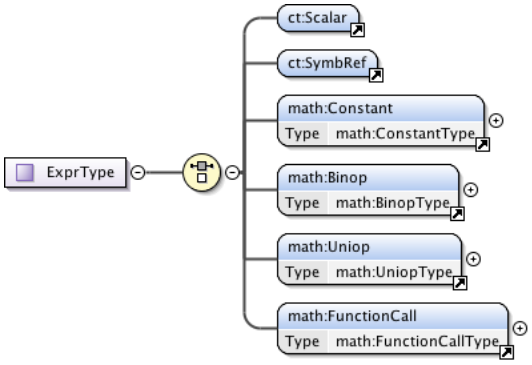
1.3.3 Complex Type `math:UniopType`

Namespace	http://www.pharmml.org/2013/03/Maths
Annotations	The unary operator type. Takes one operator.

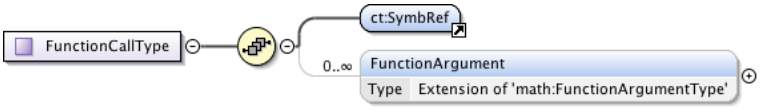
Diagram	
Type	extension of math:ExprType

Attributes	QName	Type	Use			
	op	restriction of xs:Name	required			
		The operator. More detail in the specification.				

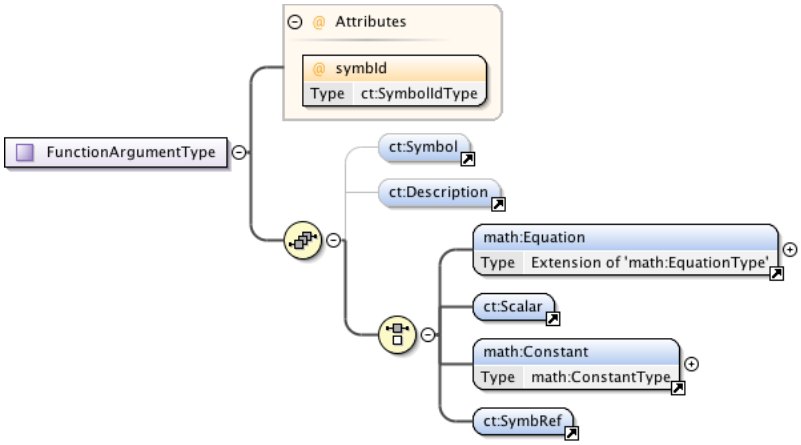
1.3.4 Complex Type `math:ExprType`

Namespace	http://www.pharmml.org/2013/03/Maths
Annotations	The schema type defining a mathematical expression.
Diagram	

1.3.5 Complex Type `math:FunctionCallType`

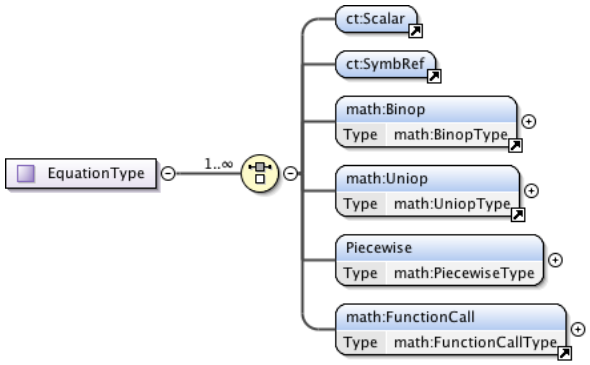
Namespace	http://www.pharmml.org/2013/03/Maths
Annotations	A type defining a function call.
Diagram	

1.3.6 Complex Type `math:FunctionArgumentType`

Namespace	http://www.pharmml.org/2013/03/Maths
Annotations	A type defining an argument of a function being called.
Diagram	

Attributes	QName	Type	Use			
	symbId	SymbolIdType	required			
		The symbold ID of the argument.				

1.3.7 Complex Type `math:EquationType`

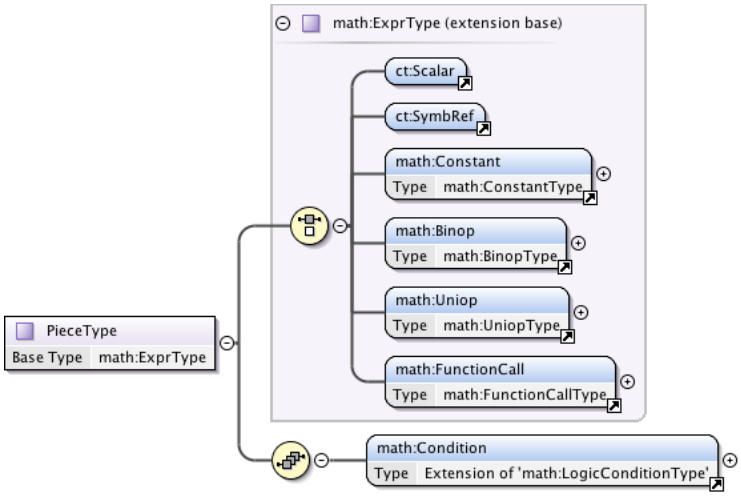
Namespace	http://www.pharmml.org/2013/03/Maths
Annotations	Complex Type that defines a mathematical equation.
Diagram	

1.3.8 Complex Type `math:PiecewiseType`

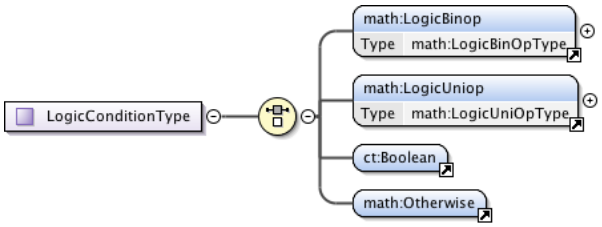
Namespace	http://www.pharmml.org/2013/03/Maths
Annotations	The schema type defining a piecewise function.
Diagram	

1.3.9 Complex Type `math:PieceType`

Namespace	http://www.pharmml.org/2013/03/Maths
Annotations	The schema type defining a `piece' in a piecewise function.

Diagram	
Type	extension of math:ExprType

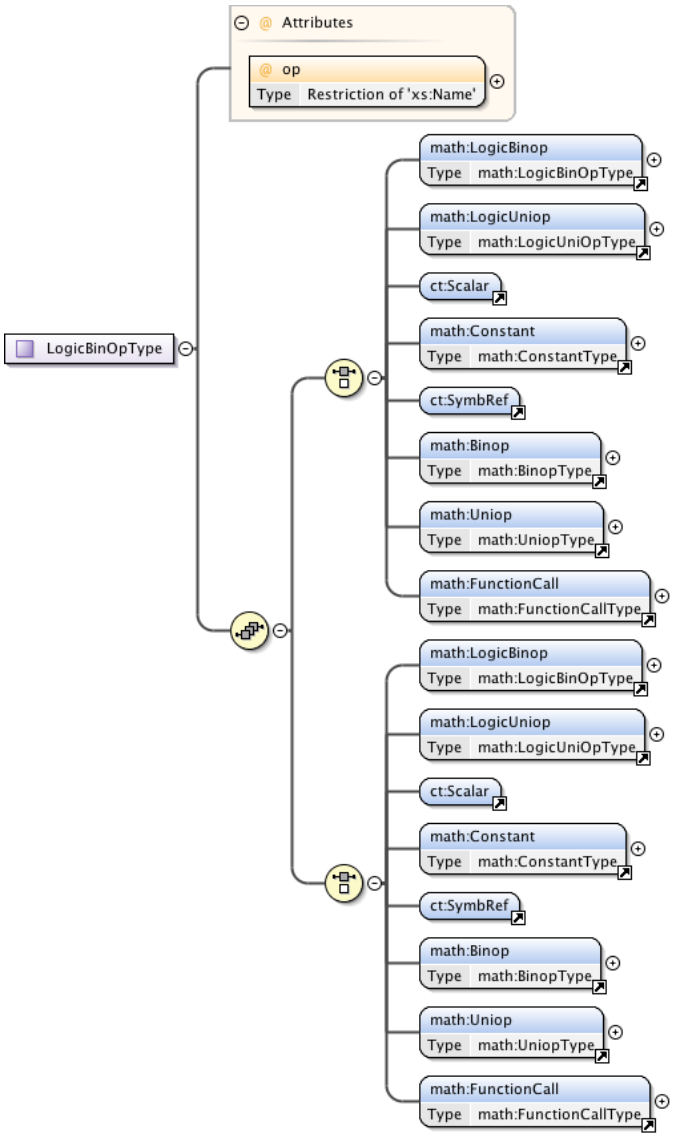
1.3.10 Complex Type `math:LogicConditionType`

Namespace	http://www.pharmml.org/2013/03/Maths
Annotations	The schema type defining logical condition.
Diagram	

1.3.11 Complex Type `math:LogicBinOpType`

Namespace	http://www.pharmml.org/2013/03/Maths
Annotations	The schema type defining a binary logical operator.

Diagram

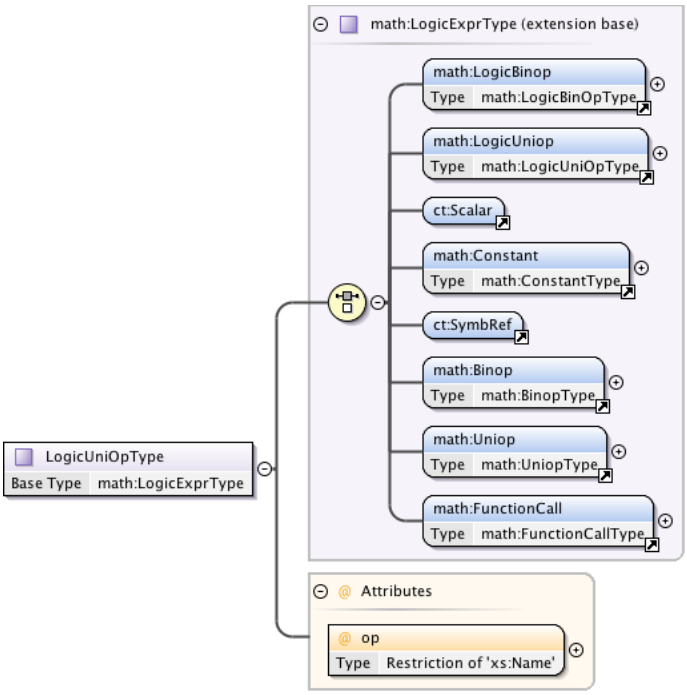


Attributes

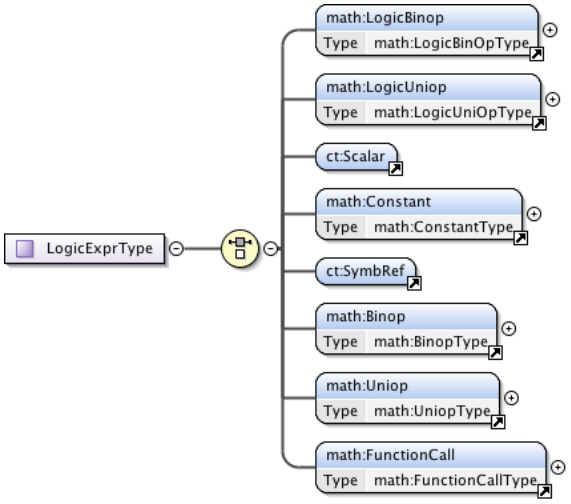
QName	Type	Use			
op	restriction of xs:Name	required			
	The logical binary operator type.				

1.3.12 Complex Type `math:LogicUniOpType`

Namespace	http://www.pharmml.org/2013/03/Maths

<table>
<tr><td>Diagram</td><td> 
</td></tr>
<tr><td>Type</td><td>extension of math:LogicExprType</td></tr>
<tr><td>Attributes</td><td>
<table>
<tr><th>QName</th><th>Type</th><th>Use</th><th></th><th></th><th></th></tr>
<tr><td>op</td><td>restriction of xs:Name</td><td>required</td><td></td><td></td><td></td></tr>
<tr><td></td><td colspan="5">The unary operator type.</td></tr>
</table>
</td></tr>
</table>

1.3.13 Complex Type `math:LogicExprType`

<table>
<tr><td>Namespace</td><td>http://www.pharmml.org/2013/03/Maths</td></tr>
<tr><td>Annotations</td><td>A logical expression.</td></tr>
<tr><td>Diagram</td><td> 
</td></tr>
</table>

2 Namespace: ""

2.1 Attribute(s)

2.1.1 Attribute `math:ConstantType /@op`

Namespace	No namespace
Annotations	The type of constant.
Type	restriction of xs:Name
Facets	enumeration `notanumber` enumeration `pi` enumeration `exponentiale` enumeration `infinity`

2.1.2 Attribute `math:LogicUniOpType /@op`

Namespace	No namespace
Annotations	The unary operator type.
Type	restriction of xs:Name
Facets	enumeration `isDefined` enumeration `not`

2.1.3 Attribute `math:LogicBinOpType /@op`

Namespace	No namespace
Annotations	The logical binary operator type.
Type	restriction of xs:Name
Facets	enumeration `lt` enumeration `leq` enumeration `gt` enumeration `geq` enumeration `eq` enumeration `neq` enumeration `and` enumeration `or` enumeration `xor`

2.1.4 Attribute `math:FunctionArgumentType /@symbId`

Namespace	No namespace
Annotations	The symbold ID of the argument.
Type	SymbolIdType

2.1.5 Attribute `math:UniopType /@op`

Namespace	No namespace
Annotations	The operator. More detail in the specification.
Type	restriction of xs:Name

Facets	enumeration	exp
	enumeration	log
	enumeration	minus
	enumeration	factorial
	enumeration	sin
	enumeration	cos
	enumeration	tan
	enumeration	sec
	enumeration	csc
	enumeration	cot
	enumeration	sinh
	enumeration	cosh
	enumeration	tanh
	enumeration	sech
	enumeration	csch
	enumeration	coth
	enumeration	arcsin
	enumeration	arccos
	enumeration	arctan
	enumeration	arcsec
	enumeration	arccsc
	enumeration	arccot
	enumeration	arcsinh
	enumeration	arccosh
	enumeration	arctanh
	enumeration	arcsech
	enumeration	arccsch
	enumeration	arccoth
	enumeration	floor
	enumeration	abs
	enumeration	ceiling
	enumeration	logistic
	enumeration	logit
	enumeration	probit

2.1.6 Attribute `math:BinopType /@op`

Namespace	No namespace
Annotations	The binary operator type. See the specification for a more detailed description.
Type	restriction of xs:Name
Facets	enumeration plus
	enumeration minus
	enumeration times
	enumeration divide
	enumeration power
	enumeration logx
	enumeration root

CHAPTER 12

Validation of PharmML

12.1 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.

12.2 Rule Identification

The rule numbers use in this chapter are *not* consistent with those in the previous specification. Because so many rules changed since the last version of PharmML, particularly in the trial design section, that we decided to start afresh. However, our intention is that the rule numbers used here will be persistent[1]. Certainly within major releases of PharmML. In practice this means is a rule becomes obsolete then it will not be reused and if it changes significantly in substance then again it will be discarded and a new rule created in it's place. This means that rule numbers are not sequential and will have gaps in number over time.

12.3 Namespaces and Scopes

12.3.1 Defining Symbols and Objects

The namespaces and scopes used in PharmML are shown in figure 12.1. By namespace we mean essentially a dictionary of names, in which each name must be unique within its given scope. As you can see. 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 6.10). The symbols can be classified as follows:

[1] Keeping rule numbers persistent will help users and developers as they move between different versions of PharmML. If validation reports an error with the same number in different versions of PharmML then you can be sure that it is the same rule.

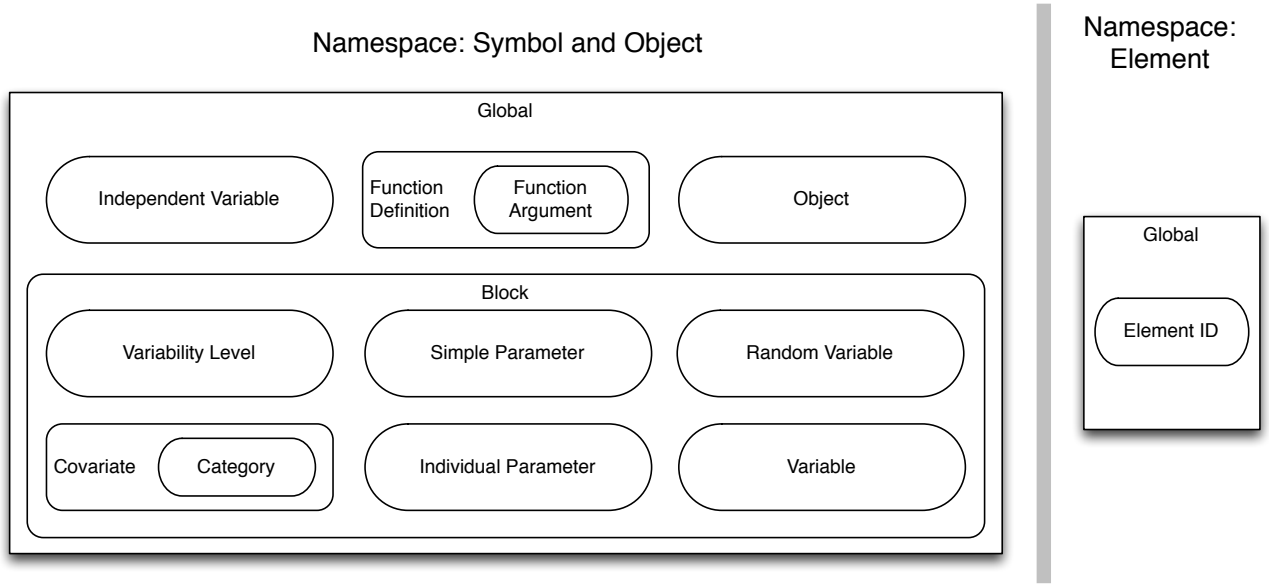


Figure 12.1: An overview of the scopes and namespaces used in PharmML. The class of symbols within the scope are shown as lozenges, symbols that also define a scope are rounded rectangles and the global scope is shown as rectangles. So for example, the function argument is a class of symbol that is scoped by the function definition which in turn belongs to the global scope.

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 as you would expect. 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.

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 belong to one of the classes described above.*

S2 *The scope and namespace of a symbol is determined by its class.* The scope and namespace of each class is described in figure 12.1.

S3 *Symbols must be unique with their scopes.* Duplicate symbols are not permitted within a given namespace.

12.3.2 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:

S4 *References to symbols and objects must resolved.* Dangling references are not permitted.

S5 *The resolved symbol must be compatible PharmML component referencing it.* By this we mean that an `<ArmRef>` which should match an arm definition should not point to an Epoch definition. Compatibility is defined in the table below.

S6 *A `<SymbRef>` element must only reference symbols that are compatible with its parent element.* Compatibility is defined by table 12.1.

S7 *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 12.2.

Table 12.1: 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. Where more than one element is possible each option is separated by the '|' symbol.

Reference Parent	Target
VariableAssignment	SimpleParameter | CovariateModel/Covariate | RandomVariable | IndividualParameter|Variable | DerivativeVariable
ParentLevel	VariabilityModel/Level[a]
PopulationParameter	SimpleParameter
LinearCovariate/Covariate	CovariateModel/Covariate
FixedEffect	SimpleParameter
GeneralCovariate	SimpleParameter | CovariateModel/Covariate

continues on next page

[a]The correct target is also affected by rule M7.

continued from previous page

Reference Parent	Target
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[a]
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

[a] The choice of valid target is governed by rule D11.

Table 12.2: 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 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. Where more than one element is possible each option is separated by the 'l' symbol.

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

12.4 Type System

12.4.1 Types

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

S8 *PharmML has a type system and all symbols and elements, if they have a type must conform it.* The types are specified in table 12.3.

S9 *Symbol classes have a type.* The types are specified in table 12.4.

S10 *Some XML elements have a type.* The types are specified in table 12.5.

S11 *Elements 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 12.5.

S12 *Literal values have a type.* The types of literal values are specified in table 12.6.

Table 12.3: 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.

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 12.4: Each symbol class has one or more types that it can be assigned to. If a type is defined as "explicit" then this means that the type is specified as part of the definition of that symbol.

Symbol class	Implicit Type
Independent Variable	real
Function Definition	explicit
Function Argument	explicit
Object	void
Block	void
Variability Level	void
Covariate	continuous: real categorical: id
Simple Parameter	real

continues on next page

continued from previous page

Symbol class	Implicit Type
Individual Parameter	real
Random Variable	real
Variable	explicit
Element ID	void

Table 12.5: 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.

Element	Type
PopulationParameter	real
FixedEffect	real
GeneralCovariate	real
GaussianModel/RandomEffects	real
RandomVariable1	real
RandomVariable2	real
CorrelationCoefficient	real
Covariance	real
DerivativeVariable/IndependentVariable	real
InitialCondition	real
ObservationModel/General	real
ObservationModel/Standard/Output	real
ErrorModel	real
RandomError	real
DoseAmount	real
SteadyState/EndTime	real
SteadyState/Interval	real
DosingTimes	real

continues on next page

continued from previous page

Element	Type
Duration	real
Rate	real
SimulationStep/Observations/Continuous	real
Property	real, int, string, boolean or array

Table 12.6: In common with other computational languages PharmML provides a mechanism to define literal values. In all cases these literals has a type.

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/>`

12.5 Common Constructs

12.5.1 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:

S13 *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.

S14 *A symbol can be assigned to only once.* See section 6.3.2.

S15 *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).

12.5.2 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:

S16 *A column reference must always to 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.

S17 *A mapping between a symbol or object to 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 12.7) must be the same as the type specified in the column definition of the dataset.

Table 12.7: 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.

Mapping Element	Column Type	Target of Mapping	Notes
Population/IndividualMapping	id		Defines the id of the mapping.
ArmMapping	id	Arm	
CovariateMapping	id	ModelDefinition/Covariate	if categorical covariate
CovariateMapping	real	ModelDefinition/Covariate	if continuous covariate
DemographicMapping	scalar	Demographic	
IVDependentMapping	table		
IndependentVariableMapping	real	IndependentVariable	
EpochMapping	id	Epoch	
IndividualRef	id		Must map to an id found in an instantiated Population.
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	
ObjectiveDataSet/IndividualMapping	id		Must map to an id found in an instantiated Population.
VariableMapping	real	Variable | DerivativeVariable	

12.5.3 Array Literal Types

Symbols of array type cannot be define 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. PharmML provides two ways to do this. The `<Sequence>` element specifies a uniform sequence of numerical values and the `<Vector>` defines an ordered list of scalar values. 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.

12.6 Dataset

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

DS1 *Columns are ordered. The order is specified by the* `columnNum` *attribute.*

DS2 *Columns must be numbered sequentially from 1, with no gaps in the sequence.*

DS3 *Each dataset must have one or more columns assigned a s a unique key.* If more than 1 column then the combination of column values together defines the key.

DS4 *Rows with a identical key values are forbidden.* The rows in the dataset must be unique with respect its unique key.

DS5 *Key columns cannot define a nested table.*

DS6 *Each cell must contain a value that is type compatible with the column definition.*

DS7 *All cells in a column must have the same type.*

DS8 *Each row must define a cell for each column.*

DS9 *A cell that is in a column defining a nested table, must instantiate a nested table.* If a column defines a nested table then that data in the table must be described using an `<Table>` element. The nested table is a properly constituted dataset and these consistency rules apply to it.

DS10 *A dataset has a unique key.* The keys of each dataset used in PharmML are defined in table 12.8.

Table 12.8: Unique keys for datasets used in PharmML.

Parent Element	Nesting Element	Key
Population	N/A	IndividualTemplate/IndividualMapping

continues on next page

continued from previous page

Parent Element	Nesting Element	Key
Population	IVDependentMapping	IndependentVariableMapping or EpochMapping
IndividualDosing	N/A	IndividualRef and DosingTime
ObjectiveDataSet	N/A	ObjectiveDataSet/IndividualMapping and VariableMapping[a]
		ObjectiveDataSet/IndividualMapping and EpochMapping

[a]The variable mapping *must* refer to the independent variable. The key here is individual ID and time.

12.7 Maths

As described in more detail in section 6.7 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.

12.7.1 Numerical Operators

T1 *The operands of the binary numerical operators have specified semantics.* The semantics are defined in table 12.9.

T2 *The operands of the unary numerical operators have specified semantics.* The semantics are defined in table 12.10.

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

Table 12.9: Numerical binary operator semantics

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)

Table 12.10: Numerical unary operator semantics

Operator	Definition

continues on next page

continued from previous page

Operator	Definition
exp	Exponential function e^x
log	Natural logarithm: $\log(x)$
minus	Negation: $-x$
factorial	Factorial: $x!$
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: $\operatorname{sech}(x)$
csch	Hyperbolic cosecant function: $\operatorname{csch}(x)$
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: $\operatorname{arcsec}(x)$
arccsc	Inverse Cosecant function: $\operatorname{arccsc}(x)$
arccot	Inverse Cotangent function: $\operatorname{arccot}(x)$
arcsinh	Inverse Hyperbolic sine function: $\operatorname{arcsinh}(x)$
arccosh	Inverse Hyperbolic cosine function: $\operatorname{arccosh}(x)$
arctanh	Inverse Hyperbolic tangent function: $\operatorname{arctanh}(x)$
arcsech	Inverse Hyperbolic secant function: $\operatorname{arcsech}(x)$
arccsch	Inverse Hyperbolic cosecant function: $\operatorname{arccsch}(x)$
arccoth	Inverse Hyperbolic cotangent function: $\operatorname{arccoth}(x)$

continues on next page

continued from previous page

Operator	Definition
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: $probit(x)$

12.7.2 Logical Operator

T4 *The operands of the binary logical operators have specified semantics.* The semantics are defined in table 12.11.

T5 *The operands of the unary logical operators have specified semantics.* The semantics are defined in table 12.12.

T6 *The logical binary operators take operands of a specified type.* The types are defined in table 12.11.

T7 *The logical unary operators take operands of a specified type.* The types are defined in table 12.12.

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

Table 12.11: Logical binary operator semantics and expected types.

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

continues on next page

continued from previous page

Operator	Definition	Operand 1	Operand 2
	$\neq$	boolean	boolean
and	Boolean AND	boolean	boolean
or	Boolean OR	boolean	boolean
xor	Boolean XOR	boolean	boolean

Logical Unary Operators

Table 12.12: Logical unary operator semantics

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

12.7.3 Constants

T9 *All mathematical constants have type real.*

T10 *The constants have specified semantics.* The semantics are defined in table 12.13.

Table 12.13: Numerical constant semantics

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

[a] See `http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=4610935`

12.8 Global rules

These rules apply to the whole PharmML document.

G1 *The model is assumed to be initialised at* $t = 0$ t is the symbol for the independent variable — defined in the `<IndependentVariable>` element. Because the model is not initialised it follows that estimation and simulation steps with values of $t < 0$ are invalid.

G2 *The default independent variable symbols id* `t` This is the value used when no `<IndependentVariable>` element is defined in a PharmML document.

12.9 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 ast most one category must be assigned a value.

M2 *Categories of a categorical covariate must equal 1* If all categories are assigned a value then they must sum to 1. If one cat is unassigned then it has a probability of 1-sum of all other categories.

M4 *Correctly defined covariance matrix* If both random effects being correlated are Normal then we can calculate the diagonal of the covariance. We only need to calculate the off-diagonal elements. Otherwise the full upper triangular matrix must be defined.

M5 *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.

M6 *Random effect correlations must be unique* A correlation between identical random effects at the same variability level of the model is forbidden.

M7 *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.

12.10 Trial Design

D1 *Demographic values of 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 *Dosing times cannot be less than zero.* See rule G1.

D6 *Epoch periods cannot be less than zero.* See rule G1.

D8 *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.

D9 *A trial structure must be complete.* A `<Structure>` element must contain at least one each of Epoch, Arm, Cell, Segment and Activity.

D10 *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.

D11 *Dosing times relative to epoch.* Dosing times are relative to the start time of the epoch to which they are assigned.

D12 *If dosing type is* `target` *then the dosing variable must be a derivative variable.*

D13 *If dosing type is* `dose` *then the dosing variable must be a non-derivative variable.*

D14 *Multiple dosing of type* `target`. If dosing is of type `target` then the dose is added to the dosing variable at the dosing time.

D15 *Multiple dosing of type* `dose`. If dosing is of type `dose` then the structural model to which the dose is applied must be algebraic. This structural model, C, must then be summed over each dose and time-point to give the output variable, C_{sum}, as follows:

$$C_{sum}(t) = \sum_{i=1}^{n} C(D_i, t_{D_i}, t)$$

12.11 Modelling Step

L1 *A trial design section must be defined if a simulation step is defined.*

L2 *A trial design section must be defined for an estimation step is defined.*

L3 *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.

L4 *Times used in a modelling step cannot occur before the initial time of the model.* In practise this means that negative times are prohibited.

Part III

Appendix

APPENDIX **A**

PharmML Version 0.1.0

This appendix includes information about the previous version of PharmML. The language specification evolves over time and involves many individuals who's participation in the development of the language also changes between releases. In this chapter we record those who contributed to version 0.1.0 of PharmML and provide information about what has changed.

A.1 Acknowledgements

> Nothing is original. Steal from anywhere that resonates with inspiration or fuels your imagination. Select only things to steal from that speak directly to your soul. If you do this, your work (and theft) will be authentic. Authenticity is invaluable; originality is non-existent. And don't bother concealing your thievery – celebrate it if you feel like it. In any case, always remember what Jean-Luc Godard said: "It's not where you take things from – it's where you take them to."
> *Jim Jarmusch*

In developing PharmML we are indebted to the many individuals within the DDMoRe project and other colleagues who contributed to the development of the standard and this specification document that describes it. While we have had help and assistance from many people we would like to highlight the contributions of some key individuals. Marc Lavielle has helped us by the clarity with which he has described the mathematical underpinnings of the type of pharmacometric model described here. He has patiently answered our many questions and has always been quick to review documents. Mats Karlsson and Lutz Harnisch have provided us with support throughout the project and Lutz in particular has been an active and challenging contributor at our workshops. Andrea Mari has provided us with many helpful suggestions for the design of PharmML and his ideas about modularity have influenced the design you see here. France Mentré has been a valuable presence during workshops and has provided a calm reason which has helped make the PharmML workshops very productive. Emmanuelle Comets assisted us greatly in the development if the variability model used in PharmML. Her deep insight made something that was opaque to us become very clear and simple. Nick Holford provided advice and feedback on multiple occasions and countless use cases indispensable in the process of the language development. Mike Smith has been an enthusiastic contributor to this work and has always been keen to help us during workshops. His help teaching us NONMEM and helping understand real-world modelling scenarios has certainly informed this work. Paolo Magni provided expertise in many aspects of population modelling and statistics and Roberto Bizzotto consulted us on various occasions regarding NMTRAN. Duncan Edwards provided a number of excellent ideas on the trial design model. And finally Andy Hooker, who was very helpful in the early stages of the development of PharmML, but was unavailable over the past few months. His sharp eye spotted some limitations in early prototypes and so saved us from ourselves!

Besides these people there are have many more people who have attended workshops and/or provided us with input otherwise. They are: Chris Franklin, Eric Blaudez, Ivan Matthews, Jonathan Chard, Joost N. Kok, Kaelig Chatel, Mateusz Rogalski, Mihai Glonţ, Natallia Kokash, Richard Kaye, Simon Thomas, Nadia Terranova, Vijayalakshmi Chelliah.

Some of our colleagues at EMBL-EBI have been helpful with the technical aspects of the standard. Sarah Keating's work on SBML and libSBML provided us with "the voice of experience". Camille Laibe and Sarala Wimalaratne helped us develop our annotation strategy for PharmML. Pierre Grenon helped us develop the structural model ontology and also helped us develop the standard library approach described here. Finally SM and MS would like to thank Henning Hermjackob for his support during recent re-organisations at the EBI and in helping us attend important workshops and meeting vital for our work.

We would like to thank the administrative team at Interface Europe for their help in organising workshops and the smooth running of the DDMoRe project. And not least Wendy Aartsen of Leiden University for her continued support and good humor in organising meetings, preparing reports and generally shielding us from administration and helping us do science!

The DDMoRe project and consequently this work, is funded by the Innovative Medicines Initiative (IMI), a large-scale public-private partnership between the European Union and the pharmaceutical industry association EFPIA. We would like to gratefully acknowledge their support.

A.2 Changes from version 0.1.0 to 0.2.0

Change	Description
Refactored XML design	Replaced many attributes with elements and increased reused of global elements - in particular from the CommonTypes schema.
Introduced object identifiers (OIDs).	
Renamed symbol referencing element to SymbRef	
Refactored variable definitions	Derivatives now defined with the initial conditions. Improved definition of initial conditions in specification.
Redesigned database.	Now defines data in XML. Has nested tables and cannot refer to external files.
Introduced integer and real types	Before there was only a scalar type, now we make a distinction.
Complete redesign of Trial Design definition.	Now define the trial design explicitly. Cannot do it through data. The trial structure borrows from CDISC.
Simplified the estimation step.	It now only maps objective data to the model and describes the estimation operations.

continues on next page

continued from previous page

Change	Description
Simplified the simulation step	Simulations cannot now driven by a datafile and the definition of repetitions is removed. This feature may be reintroduced in the future.
Extended the definition of the Parameter Model	Implicit parameter models are now supported as is a Gaussian parameter model with a non-linear covariate model.
Extended the residual error model	Can support much more complex error models including those commonly defined in NONMEM.
Removed external inclusions.	To simplify this version of the language all imports of external resources have been removed. This simplifies validation and the aim is to reintroduce this later.
Added element identifier	To facilitate referencing by external resources all elements can have an optional id attribute.
Pre-release UncertML 3.0 incorporated.	Probability distributions are now defined using an externally maintained resource.

A.3 Changes from version 0.2.0 to 0.2.1

Change	Description
Fixes to specification	Fixed errors and typos in the text.
Expanded examples	Added a steady state example.
Additional schema documentation.	Expanded the schema documentation.
Minor schema refactoring.	Refactored the trialDesign.xsd to replace anonymous types with complex types and so comply with our design guidelines.
Added the id type.	We realised when writing version 0.2.0 that we needed a type for identifiers. This made it to the validation section but was omitted in the XML Schema definition and the examples. This is now rectified.

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., 1992] Beal, S. L., Boeckmann, A. J., and Sheiner, L. B. (November 1992). NONMEM Users Guide Part VI, PREDPP Guide. Technical report, NONMEM Project Group, University of San Francisco.

[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 SDM-XML Technical Committee, 2011] CDISC SDM-XML Technical Committee (2011). CDISC Study Design Model in XML (SDM-XML), Version 1.0. Technical report, Clinical Data Interchange Standards Consortium, Inc.

[Chan et al., 2005] Chan, P. L. S., Nutt, J. G., and Holford, N. H. G. (2005). Importance of within subject variation in levodopa pharmacokinetics: a 4 year cohort study in parkinson's disease. *J Pharmacokinet Pharmacodyn*, 32(3-4):307–31.

[DDMoRe/Copenhagen meeting, 2013] DDMoRe/Copenhagen meeting (2013). MML/Technical meeting.

[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, 2012] Lavielle, M. (June 25, 2012). Mathematical description of mixed effects models. Technical report, INRIA Saclay.

[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.

[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.

[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.

[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 and Moodie, 2013] Swat, M. J. and Moodie, S. (2013). Use Cases with Matlab Implementation. Technical report, EMBL-EBI.

[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://http://sedml.org/specifications.html`.