A MANUAL FOR ADDING MODELS INTO HTTK

Marina V Evans and Celia Schacht

2024-02-14

Contents

Chapter 1

Introduction

The purpose of this document is to instruct a modeler on how to add a model into httk. This vignette will discuss the overall steps needed to create the httk package with new functions created by the user. Currently, httk contains the following models: pbtk, 3 compartment, 1 compartment, fetal, and gas. Each model is unique with varying tissue compartments and routes for exposure, depending on the model goal. For example, the gas model is used for open chamber inhalation of volatile organic compounds. The following vignette will explain the steps to incorporate a new model into httk. The case study for this new model is a closed chamber inhalation model. The following instructions will detail

1. Converting a model (in .model format) to C code with MCSIM and altering it to make it compatible in httk.

2. Creating the Modelinfo_MODELNAME file.

3. Creating the parameterize_MODELNAME file to generate parameters for your model.
4. Creating the solve_MODELNAME file to solve the model and generate time and concentration vectors.

5. Using devtools to add the model to httk and roxygen to add comments.

Adding a model to httk is done by adding functions and models to the httk repository, which can be downloaded from GitHub[Wambaugh, JF,Pearce RG, et al., 2017]. Once these are on your computer, you can use DevTools (discussed in section 5) to load it locally on your machine. The following vignette highlights

the steps to add a model to this local version of httk so that you can use httk's functionality locally.

1.1 McSim under RStudio

Chapter 1 will introduce the reader to the process needed to compile a .model file into C code using MCSim. There are multiple platforms for running PBPK models that can be converted into C for use in R. The USEPA has invested significant effort in using R to develop a generic, high-throughput PBTK model known as httk. Since internal dose prediction using PBPK models remains a major need for thousands of chemicals, the model has been named high throughput toxicokinetics or httk. The R package httk is free and available for anyone to use (https://github.com/USEPA/CompTox-ExpoCast-httk). Note that httk's PBTK models are written in C code which is why we have provided instructions on converting .model files into C code; if your model is already in C code, you may skip this step. MCSIM is a simulation package with a GNU C compiler that allows the user to add PBPK models into httk by first converting them into C. This user guide will show a step by step approach from start to finish.

1.2 Gas uptake models

A collection of legacy closed chamber PBPK models exists scattered over publications since the 1990s. Most of these models were written in ACSL, and some in Matlab. These original models can be converted into C and integrated into httk. The USEPA has invested significant effort in using R to develop a generic, high-throughput PBTK model knkown as httk. The goal is to help simplify modeling for risk assessors, and to have a tool that can be used for multiple chemicals. Since internal dose prediction using PBPK models remains a major need for thousands of chemicals, the model has been named high throughput toxicokinetics or httk. The package is free and available for anyone to use (https://github.com/USEPA/CompTox-ExpoCast-httk).

How does this tool change the way we can perform simulations, particularly when we may have legacy code that has been already used in risk assessment? Fortunately, there is a way to compile source code into C and use R functions to extract the necessary information to run httk.

There are three major steps needed to map information from the PBPK C code into httk:

- Get all model parameters (chemical specific and physiological)
- Provide initial conditions needed to solve the equations
- Provide a list of the ordinary differential equations (ODEs) that need to be solved.

1.2.1 Enabling MCSim

The first step involves enabling the use of a tool to translate a model defined in MCSim model language (a ".model" file) into the C language (a ".C" file). The code to complete this action was written by Dr. Dustin Kapruan and is in Step 8 and is an R interface to GNU MCSim. The following steps outline the necessary steps taken to make use of this tool. For additional information related to MCSim for R please see: https://www.gnu.org/software/mcsim/.

1) Download essential files which will be used to build the "mod" utility of GNU MCSim (explained more below). A compressed (ZIP) directory containing all necessary files and additional instructions can be accessed here: https://www.gnu.org/software/mcsim/mcsim_under_R.zip. Unzip the files and save them into a directory of your choice.

2) Install R (https://cran.r-project.org). We recommend using the default installation directory, `C:\Program Files\`.

3) (Optional, but recommended.) Install Rstudio (https://www.rstudio.com). Rstudio includes a code editor and debugging and visualization tools.

4) From R or Rstudio, install the package "deSolve" (which provides methods for integrating ordinary differential equations). If using command line R, use the "install.packages" method. If using RGui, choose "Install package(s)…" from the "Packages" menu, then follow the instructions. If using Rstudio, choose "Install packages…" from the "Tools" menu, then follow the instructions.

5) Install Rtools (https://cran.r-project.org/bin/windows/Rtools/). This should be installed in the default location within your directory. Note that installation of Rtools requires administrative privileges. The Rtools suite of tools includes a C compiler (gcc) that will be used to generate a DLL version of a given MCSim model that can be called by the R package "deSolve". NOTE: Users of ACSLX may experience issues after installing Rtools because a directory containing an alternate C compiler (gcc) will be added to the PATH environment variable. Changing the ordering of the directories listed in the PATH environment variable such that either the ACSLX or the Rtools directories are listed *first* (depending on which of these the user plans to run) should resolve any such problem. The relevant directories are listed in the *system* PATH environment variable (not the user-specific PATH environment variable), and administrative privileges are required to modify this PATH variable. ***THIS STEP IS NOT NECESSARY FOR MAC/UNIX USERS.

    a. Add the Rtools file path to the gcc compiler to your PATH environmental variable in R itself, if it is not already in your path. This can

be done with the R script provided below: add_mod_to_filepath.R You will not be able to create the "mod.exe" file in subsequent steps without modifying your R path to include the gcc compiler.

We are using the System functions to help us set the path in case the user does not have administrative privileges.# First, find the path to your Rtools folder containing gcc.exe. Everyone's path may be different depending on when you downloaded Rtools, but mine is "C:/rtools40/mingw64/bin". If you are having trouble locating this folder. Try searching for "gcc.exe" it should be in a folder called "bin". Windows also uses the / (forward slash) to specify paths. Please use this convention.

```
#-------------------------------------------------------------------------
# add_file_path.R
#
# This file contains code to assist in correctly running the building_mod.R code as pa
#
# Author: Annabel Meade, U.S. EPA and ORISE, January 2021
# Edited by Marina V Evans and Celia Schacht, US EPA , 05/16/2023
#-------------------------------------------------------------------------

# path_to_rtools_folder <-
 path_to_rtools_folder <-"C:/rtools40/mingw64/bin"

# Now get your PATH environmental variable in R. Look at the formatting, usually
#   the file paths are  seperated by ":" or ";". This will be important when you
#   add your path to your Rtools folder.
old_path <- Sys.getenv("PATH")

#Windows uses the ";" character as separator for different paths.

Sys.setenv(PATH=paste(old_path, path_to_rtools_folder, sep = ";"))
```

6. Now that "gcc" can be located on your PATH, the next step requires the user to build the "mod" utility. First, go into the config.h file (in the "mod" folder) and ensure that the line "#define HAVE_LIBSBML 1" is commented out. The more advanced user may go into the README.txt file in the MCSim_under_R folder and configure this differently.

Next, Within R, navigate to the directory to which you had saved the downloaded MCSim files from step 1. The folder name is "MCSim under R" and it contains three subfolders: "mod", "my_R_project", and "sim". Relocate to the folder "my_R_project" in the directory where installed these MCSim files using Session>Set Working. For example, for WINDOWS computers, we placed the folder in this location: "~/Documents/MCSim under R/my_R_project". It

may be helpful to use getwd() to confirm where the directory in which your current R session is running.

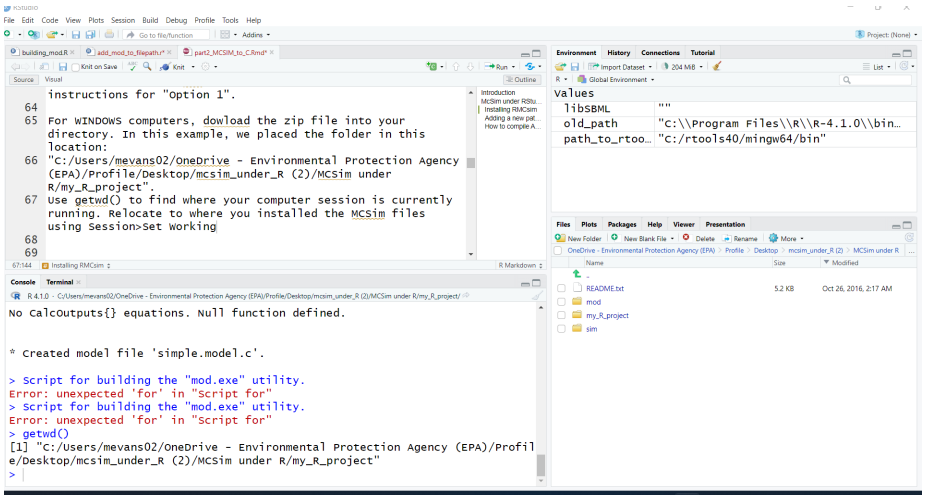


Figure 1.1: File structure after unzipped MCSim

Now it is time to build mod.exe. Open "building_mod.R" in the my_R_project folder (again, confirm your working directory is in alignment with the location of the file).Run the building_mod.R script to create mod.exe and make sure it compiles simple.model.c. The output in your console should look like:

Mod v6.1.0 - Model Generator for MCSim

MCSim and associated software comes with ABSOLUTELY NO WARRANTY; This is free software, and you are welcome to redistribute it under certain conditions; see the GNU General Public License.

- Created model file 'simple.model.c'.

a. If you are using a Mac/Unix, you have to modify the building_mod.R file by removing the .exe extension to create a Unix executable instead of a Microsoft Windows Executable. Replace this line of code:

system(paste("gcc -o ../mod/mod.exe ../mod/*.c", libSBML, sep = " "))

with this line of code:

system(paste("gcc -o ../mod/mod ../mod/*.c", libSBML, sep = " "))

7) Make a copy of mod.exe file that has just been created using building_mod.R, and put it somewhere you will be able to find it again. This

way, you do not have to keep moving mod.exe every time you want to use MCSim. Modify your PATH environment variable so that the directory containing "mod.exe" (created in Step 6 above) is listed in your PATH. In our example, we moved "mod.exe" into a folder called "R_Utility" with the path "~/Documents/R_Utility".

```
#-------------------------------------------------------------------------------
# Get your PATH in R. See comments above for more information.
old_path <- Sys.getenv("PATH")

# Get path to folder containing mod.exe
#folderpath_with_mod <- "Your file path goes here"  #See lines 101-105
folderpath_with_mod <- "~/Documents/R_Utility"
#Add the mod.exe path to existing path.
Sys.setenv(PATH=paste(old_path, folderpath_with_mod, sep = ";"))
#You can check for errors by using the Sys.getenv("PATH") command.
```

7 a) NOTE: Different versions of RTOOLS or operating systems may produce issues with bu

```
# Script for building the "mod.exe" utility- workaround.
#
# Run it once (or if you delete "mod.exe" by mistake).
# The mod directory with all the needed C files should be in the directory
# just above (or change the path accordingly).
# You may have to set gcc path explicitly too if Rtools are not well installed
#
#
# Modified by C.M. Schacht
# This script is used in the event that your system does not recognize the wildcard *
# used in building_mod.R
# Note: building_mod.R gives the following command: #system(paste("gcc -o ../mod/mod.e



libSBML = ""
# libSBML = "-lsbml" # uncomment and execute if you have and want to use libSBML.

our.files = c("../mod/strutil.c ../mod/modo.c ../mod/modiSBML2.c ../mod/modiSBML.c ../
system(paste("gcc -o ../mod/mod.exe ", our.files, " " , libSBML, sep = ""))
```

The next step requires you to check that it works by compiling the "simple.model" that is located in the My_R_project folder. Copy your newly created mod.exe file that is located in the mod folder into My_R_project folder and run the following code. If it works, you should see a .c file get generated named "simple.model.c"

```
#drag mod.exe to my_R_project
# you are done.
system2("mod.exe",args="-R simple.model simple.model.c")
```

1.2.2 How to compile ACSL code into C

8) We then use the RMCSim functionality to convert your code into C code. RMCSim uses the "compile_model" function to translate an MCSim model (.model) file into the C language (with a .c suffix). Once the subsequent steps have been performed, copy the code below into an R script and source it.

```
# RMCSim.R
#
# This file contains R functions for compiling, loading, and running an ODE
# model encoded in the GNU MCSim model specification language. The "mod.exe"
# utility must be available in the user's PATH.
#
# Author: Dustin Kapraun, U.S. EPA, November 2018.
#-------------------------------------------------------------------------------

# Load deSolve.
library(deSolve)

compile_model <- function(mName) {
# This function translates a model that has been defined in an MCSim model
# (".model") file into the C language (i.e., a ".c" file). It then compiles the
# model to create an object code (".o") file and a dynamic linked library
# (".dll") file, as well as an R script ("_inits.R") containing several R
# functions that can be used for initializing model states and parameters.
#
# Inputs:
#   mName: String containing the name of the MCSim model. Exclude the file name
#   suffix ".model". If the function is called from a working directory other
#   than the one containing the ".model" file, the full path of the ".model"
#   file should be provided.

    # Construct names of required files and objects from mName.
    model_file = paste(mName, ".model", sep="")
    c_file = paste(mName, "_model.c", sep="")
    dll_name = paste(mName, "_model", sep="")
    dll_file = paste(dll_name, .Platform$dynlib.ext, sep="")

    # Unload DLL if it has been loaded.
```

```
    if (is.loaded("derivs", PACKAGE=dll_name)) {
        dyn.unload(dll_file)
    }

    # Create a C model file (ending with ".c") and an R parameter
    # initialization file (ending with "_inits.R") from the GNU MCSim model
    # definition file (ending with ".model"). Using the "-R" option generates
    # code compatible with functions in the R deSolve package.
    system(paste("mod -R ", model_file, " ", c_file, sep = ""))

    # Compile the C model to obtain "mName_model.o" and "mName_model.dll".
    system(paste("R CMD SHLIB ", c_file, sep = ""))

}
```

In a different chapter, we are going to set fileName = "gas_closed_pbtk" to compile the closed inhalation file. As stated in the function comments, this file must be in the same location as the RMCSim.R file. By running the command below, a file named fileName_model.c should be created. Because the mod executable was added to the PATH, one should be able to use the code below (after running or sourcing compile_function from above) and create a .c file from their .model file (as long as the .model file is in the same location as compile_model).

```
fileName = "gas_closed_pbtk"
compile_model(fileName)
```

If the user had to resolve their issues with step 7a, they can simply compile their .model into .c files by ensuring their .model file is in the same folder as mod.exe and typing the following command:

```
model.name = "example.model"
system2("mod.exe",args=paste("-R ", model.name," ", model.name,".c",sep=""))
```

For additional context for MCSim under R software see: https://rpubs.com/Nanhung/MCSim_with_RStudio.

Chapter 2

Comparison between R modelinfo and c code

After generation of new C code, the first recommended task is to create the modelinfo file, which links the parameters, states, and outputs in the C code with httk. The format for this file is the same for each model and should be tailored to each new model. The name used for the inhalation model is modelinfo_gas_pbtk.R and can be found in your httk/R.

Within modelinfo, the model name that "model.list" is calling to should be modified to fit the model "gas_closed_pbtk" so that the items called to by model.list corresponding to the appropriate model. The parameterize and solve functions, both of which will be created for each unique model, are called in modelinfo and thus their names need to include the model accordingly.

The purpose of this chapter is to introduce the reader to the pbtk model structure as is currently coded. This httk option would allow the user to run simulations for a non-volatile chemical without the inhalation route. If the chemical is volatile, there is currently a gas_pbtk option within httk. The organs included in pbtk are: liver, gut, lung, kidney, arterial and venous blood, and rest of the body. The user lists the compartments that are to be un-lumped from the rest of the body compartment using $tissuelist. Default values for these compartments are included in a table within httk.

Let's introduce the structure and organs included in the pbtk model:

2.1 Modelinfo function

After generation of new C code, the first recommended task is to create the modelinfo file, which links the parameters, states, and outputs in the C code

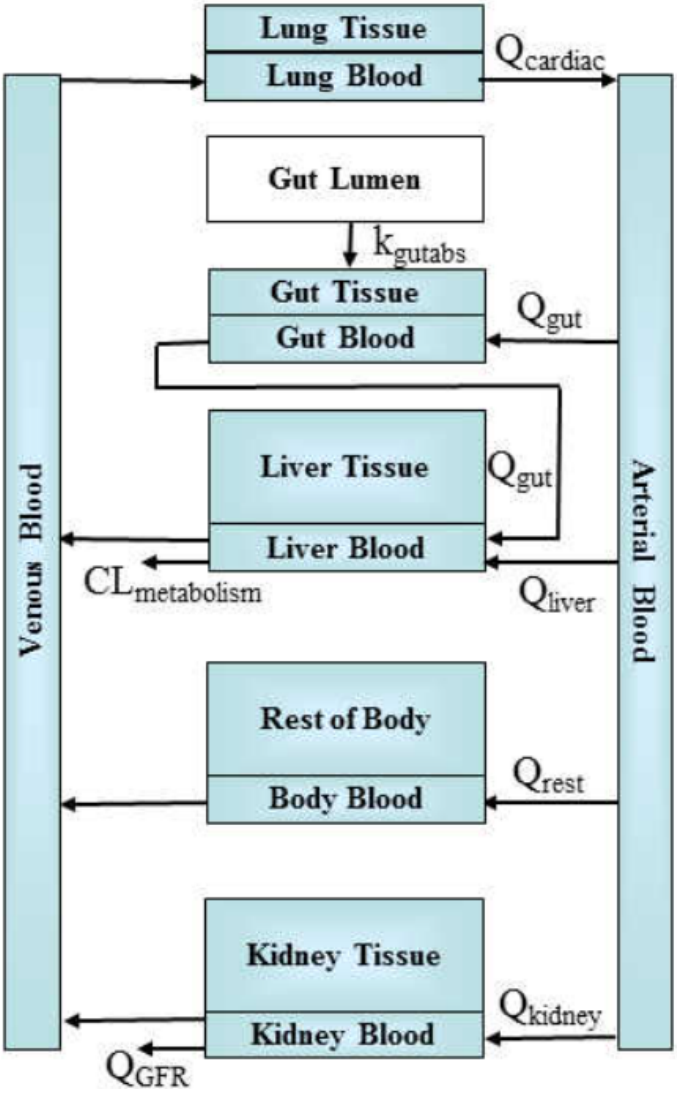


Figure 2.1: Structure and organs in pbtk model

with httk. The format for this file is the same for each model and should be tailored to each new model. The format used in this vignette tries to divide the page in columns so that you can compare C and R code directly. Some R code is included to link to other R functions, and will not have a C counterpart. In those cases, there will only be one column presented.

The function starts with comments for the parameters used in the R code.The next section is unique to modelinfo_pbtk.R and does not have a C code counterpart:

```
# Analytic expression for steady-state plasma concentration to be used by
# calc_analytic_css:
model.list[["pbtk"]]$analytic.css.func <- "calc_analytic_css_pbtk"

# What units does the analytic function return in calc_analytic_css:
model.list[["pbtk"]]$steady.state.units <- "mg/L"

# When calculating steady-state with calc_css, which compartment do we test?
# ("C" is preprended):
model.list[["pbtk"]]$steady.state.compartment <- "plasma"

# Function used for generating model parameters:
model.list[["pbtk"]]$parameterize.func <- "parameterize_pbtk"

# Function called for running the model:
model.list[["pbtk"]]$solve.func <- "solve_pbtk"

# Here are the tissues from tissue.data that are considered (for example,
# do we include placenta or not? Here, yes we do). They should correspond
# in name to the names present in the tissue.data object, if the parameters
# necessary for describing the tissue/compartment aren't going to be provided
# otherwise.
model.list[["pbtk"]]$alltissues=c(
  "adipose",
  "bone",
  "brain",
  "gut",
  "heart",
  "kidney",
  "liver",
  "lung",
  "muscle",
  "skin",
  "spleen",
  "red blood cells",
  "rest")
```

How to lump or un-lump existing tissues. The user/modeler will make these decisions.

```
# How the tissues from tissue.data are lumped together to form the model:
# PBTK model has liver, kidney, gut, and lung compartments that draw info
# from tissue.data; everything else from alltissues should be lumped.
model.list[["pbtk"]]$tissuelist=list(
                          liver=c("liver"),
                          kidney=c("kidney"),
                          lung=c("lung"),
                          gut=c("gut"))
```

A list of parameters included in the parameterize_pbtk related function. The parameterize function can be used before calling the solve_pbtk function. Only parameters listed here can be modified by changing their value.

```
# These are all the parameters returned by the R model parameterization function.
# Some of these parameters are not directly used to solve the model, but describe
# how other parameters were calculated:
model.list[["pbtk"]]$param.names <- c(
  "BW",
  "Clint",
  "Clmetabolismc",
  "Funbound.plasma",
  "Funbound.plasma.dist",
  "Funbound.plasma.adjustment",
  "Fgutabs",
  "Fhep.assay.correction",
  "hematocrit",
  "Kgut2pu",
  "kgutabs",
  "Kkidney2pu",
  "Kliver2pu",
  "Klung2pu",
  "Krbc2pu",
  "Krest2pu",
  "liver.density",
  "million.cells.per.gliver",
  "MW",
  "Pow",
  "pKa_Donor",
  "pKa_Accept",
  "MA",
  "Qcardiacc",
  "Qgfrc",
```

```
  "Qgutf",
  "Qkidneyf",
  "Qliverf",
  "Rblood2plasma",
  "Vartc",
  "Vgutc",
  "Vkidneyc",
  "Vliverc",
  "Vlungc",
  "Vrestc",
  "Vvenc")
```

Left panel is the C code for a generic pbtk model

```
#define BW parms[0]
#define Clmetabolismc parms[1]
#define hematocrit parms[2]
#define kgutabs parms[3]
#define Kkidney2pu parms[4]
#define Kliver2pu parms[5]
#define Krest2pu parms[6]
#define Kgut2pu parms[7]
#define Klung2pu parms[8]
#define Qcardiacc parms[9]
#define Qgfrc parms[10]
#define Qgutf parms[11]
#define Qkidneyf parms[12]
#define Qliverf parms[13]
#define Vartc parms[14]
#define Vgutc parms[15]
#define Vkidneyc parms[16]
#define Vliverc parms[17]
#define Vlungc parms[18]
#define Vrestc parms[19]
#define Vvenc parms[20]
#define Fraction_unbound_plasma parms[21]
#define Rblood2plasma parms[22]
#define Clmetabolism parms[23]
#define Qcardiac parms[24]
#define Qgfr parms[25]
#define Qgut parms[26]
#define Qkidney parms[27]
#define Qliver parms[28]
#define Qrest parms[29]
#define Vart parms[30]
```

```
#define Vgut parms[31]
#define Vkidney parms[32]
#define Vliver parms[33]
#define Vlung parms[34]
#define Vrest parms[35]
#define Vven parms[36]
```

Right panel is the R code from modelinfo_pbtk.R.

```
model.list[["pbtk"]]$Rtosolvermap <- list(
  BW="BW",
  Clmetabolismc="Clmetabolismc",
  hematocrit="hematocrit",
  kgutabs="kgutabs",
  Kkidney2pu="Kkidney2pu",
  Kliver2pu="Kliver2pu",
  Krest2pu="Krest2pu",
  Kgut2pu="Kgut2pu",
  Klung2pu="Klung2pu",
  Qcardiacc="Qcardiacc",
  Qgfrc="Qgfrc",
  Qgutf="Qgutf",
  Qkidneyf="Qkidneyf",
  Qliverf="Qliverf",
  Vartc="Vartc",
  Vgutc="Vgutc",
  Vkidneyc="Vkidneyc",
  Vliverc="Vliverc",
  Vlungc="Vlungc",
  Vrestc="Vrestc",
  Vvenc="Vvenc",
  Fraction_unbound_plasma="Funbound.plasma",
  Rblood2plasma="Rblood2plasma"
  #The rest of the parameters listed in the C code are calculated using the  getParmsp
)
```

List of state variables

```
    Agutlumen = 0.0,
    Agut = 0.0,
    Aliver = 0.0,
    Aven = 0.0,
    Alung = 0.0,
    Aart = 0.0,
```

```
    Arest = 0.0,
    Akidney = 0.0,
    Atubules = 0.0,
    Ametabolized = 0.0,
    AUC = 0.0
```

Same list of state variables must be copied.

```
model.list[["pbtk"]]$state.vars <- c(
    "Agutlumen",
    "Agut",
    "Aliver",
    "Aven",
    "Alung",
    "Aart",
    "Arest",
    "Akidney",
    "Atubules",
    "Ametabolized",
    "AUC"
    )
```

Initialization of parameters in C code - function name getParmspbtk must be the same in C and R code. These are named by the user.

```
void getParmspbtk (double *inParms, double *out, int *nout) {
/*----- Model scaling */

  int i;

  for (i = 0; i < *nout; i++) {
    parms[i] = inParms[i];
  }


  kgutabs = kgutabs * 24 ;
  Clmetabolism = Clmetabolismc * 24 * BW ;
  Qcardiac = Qcardiacc * 24 * pow ( BW , 0.75 ) ;
  Qgfr = Qgfrc * pow ( BW , 0.75 ) * 24 ;
  Qgut = Qcardiac * Qgutf ;
  Qkidney = Qcardiac * Qkidneyf ;
  Qliver = Qcardiac * Qliverf ;
  Qrest = Qcardiac - ( Qgut + Qkidney + Qliver ) ;
  Vart = Vartc * BW ;
  Vgut = Vgutc * BW ;
```

```
  Vkidney = Vkidneyc * BW ;
  Vliver = Vliverc * BW ;
  Vlung = Vlungc * BW ;
  Vrest = Vrestc * BW ;
  Vven = Vvenc * BW ;

  for (i = 0; i < *nout; i++) {
    out[i] = parms[i];
  }
  }
```

Initialization of parameters in C code - function name getParmspbtk must be the same in C and R code. These are named by the user.

```
# This function translates the R model parameters into the compiled model
# parameters:
model.list[["pbtk"]]$compiled.parameters.init <- "getParmspbtk"

# This needs to be a global variable so that R CMD check --as-cran can test
# the code (the HTTK package does not use this):
compiled_parameters_init <- "getParmspbtk"
```

List of compiled parameter names from C code.

```
/* 37 Parameters:
     BW = 70,
     Clmetabolismc = 0.203,
     hematocrit = 0.44,
     kgutabs = 1,
     Kkidney2pu = 0,
     Kliver2pu = 0,
     Krest2pu = 0,
     Kgut2pu = 0,
     Klung2pu = 0,
     Qcardiacc = 4.8,
     Qgfrc = 0.108,
     Qgutf = 0.205,
     Qkidneyf = 0.221,
     Qliverf = 0.0536,
     Vartc = 0.0487,
     Vgutc = 0.0158,
     Vkidneyc = 0.00119,
     Vliverc = 0.02448,
     Vlungc = 0.00723,
     Vrestc = 0.77654,
```

```
    Vvenc = 0.0487,
    Fraction_unbound_plasma = 0.0682,
    Rblood2plasma = 0.0,
    Clmetabolism = 0.0,
    Qcardiac = 0.0,
    Qgfr = 0.0,
    Qgut = 0.0,
    Qkidney = 0.0,
    Qliver = 0.0,
    Qrest = 0.0,
    Vart = 0.0,
    Vgut = 0.0,
    Vkidney = 0.0,
    Vliver = 0.0,
    Vlung = 0.0,
    Vrest = 0.0,
    Vven = 0.0,
*/
```

List of compiled parameter names in R:

```
model.list[["pbtk"]]$compiled.param.names <- c(
  "BW",
  "Clmetabolismc",
  "hematocrit",
  "kgutabs",
  "Kkidney2pu",
  "Kliver2pu",
  "Krest2pu",
  "Kgut2pu",
  "Klung2pu",
  "Qcardiacc",
  "Qgfrc",
  "Qgutf",
  "Qkidneyf",
  "Qliverf",
  "Vartc",
  "Vgutc",
  "Vkidneyc",
  "Vliverc",
  "Vlungc",
  "Vrestc",
  "Vvenc",
  "Fraction_unbound_plasma",
  "Rblood2plasma",
```

```
  "Clmetabolism",
  "Qcardiac",
  "Qgfr",
  "Qgut",
  "Qkidney",
  "Qliver",
  "Qrest",
  "Vart",
  "Vgut",
  "Vkidney",
  "Vliver",
  "Vlung",
  "Vrest",
  "Vven"
  )
```

Initialization of parameters in the C code to values at start of simulations.

```
/*----- Initializers */
void initmodpbtk (void (* odeparms)(int *, double *))
{
  int N=37;
  odeparms(&N, parms);
}
```

The name of this function must be the same as used in the C code, initmodpbtk

```
# This function initializes the state vector for the compiled model:
model.list[["pbtk"]]$compiled.init.func <- "initmodpbtk"
```

The ODE equations are listed in the C code and are changed to outputs in the R code. The name of the derivative function must be the same in C and R codes, derivspbtk.

```
/*----- Dynamics section */

void derivspbtk (int *neq, double *pdTime, double *y, double *ydot, double *yout, int *
{

  yout[ID_Cgut] = y[ID_Agut] / Vgut ;

  yout[ID_Cliver] = y[ID_Aliver] / Vliver ;

  yout[ID_Cven] = y[ID_Aven] / Vven ;
```

```
  yout[ID_Clung] = y[ID_Alung] / Vlung ;

  yout[ID_Cart] = y[ID_Aart] / Vart ;

  yout[ID_Crest] = y[ID_Arest] / Vrest ;

  yout[ID_Ckidney] = y[ID_Akidney] / Vkidney ;

  yout[ID_Cplasma] = y[ID_Aven] / Vven / Rblood2plasma ;

  yout[ID_Aplasma] = y[ID_Aven] / Rblood2plasma * ( 1 - hematocrit ) ;

  ydot[ID_Agutlumen] = - kgutabs * y[ID_Agutlumen] ;

  ydot[ID_Agut] = kgutabs * y[ID_Agutlumen] + Qgut * ( y[ID_Aart] / Vart - y[ID_Agut] / Vgut * Rb

  ydot[ID_Aliver] = Qliver * y[ID_Aart] / Vart + Qgut * y[ID_Agut] / Vgut * Rblood2plasma / Kgut2

  ydot[ID_Aven] = ( ( Qliver + Qgut ) * y[ID_Aliver] / Vliver / Kliver2pu + Qkidney * y[ID_Akidne

  ydot[ID_Alung] = Qcardiac * ( y[ID_Aven] / Vven - y[ID_Alung] / Vlung * Rblood2plasma / Klung2p

  ydot[ID_Aart] = Qcardiac * ( y[ID_Alung] / Vlung * Rblood2plasma / Klung2pu / Fraction_unbound_

  ydot[ID_Arest] = Qrest * ( y[ID_Aart] / Vart - y[ID_Arest] / Vrest * Rblood2plasma / Krest2pu /

  ydot[ID_Akidney] = Qkidney * y[ID_Aart] / Vart - Qkidney * y[ID_Akidney] / Vkidney / Kkidney2pu

  ydot[ID_Atubules] = Qgfr * y[ID_Aart] / Vart / Rblood2plasma * Fraction_unbound_plasma ;

  ydot[ID_Ametabolized] = Clmetabolism * y[ID_Aliver] / Vliver / Kliver2pu ;

  ydot[ID_AUC] = y[ID_Aven] / Vven / Rblood2plasma ;

} /* derivs */
```

The “derivspbtk” function in R calls the outputs of all the derivatives.The name of the derivative function must be the same in C and R codes, derivspbtk.

```
model.list[["pbtk"]]$derivative.func <- "derivspbtk"
```

Next are the list of output functions from the C code. They must be copied in the exact order from the C to the R code (enclosed in “ “).

```
 9 Outputs:
    "Cgut",
    "Cliver",
    "Cven",
    "Clung",
    "Cart",
    "Crest",
    "Ckidney",
    "Cplasma",
    "Aplasma",
```

Next are the list of output functions from the C code. They must be copied in the exact order from the C to the R code (enclosed in " ").

```
model.list[["pbtk"]]$derivative.output.names <- c(
  "Cgut",
  "Cliver",
  "Cven",
  "Clung",
  "Cart",
  "Crest",
  "Ckidney",
  "Cplasma",
  "Aplasma"
  )
```

Next come a series of code that was designed for httk and is written in R. There is no C counterpart to this section. It consists of a list of variables to be plotted, units used for variables and routes of exposure.

```
model.list[["pbtk"]]$default.monitor.vars <- c(
  "Cgut",
  "Cliver",
  "Cven",
  "Clung",
  "Cart",
  "Crest",
  "Ckidney",
  "Cplasma",
  "Atubules",
  "Ametabolized",
  "AUC"
  )
```

```
# Allowable units assigned to dosing input:
model.list[["pbtk"]]$allowed.units.input <- list(
       "oral" = c('umol','mg','mg/kg'),
       "iv" = c('umol','mg','mg/kg'))

# Allowable units assigned to entries in the output columns of the ode system
model.list[["pbtk"]]$allowed.units.output <- list(
       "oral" = c('uM','mg/l','umol','mg','uM*days','mg/L*days'),
       "iv" = c('uM','mg/l','umol','mg','uM*days','mg/L*days'))

model.list[["pbtk"]]$routes <- list(
  "oral" = list(
# We need to know which compartment gets the dose
    "entry.compartment" = "Agutlumen",
# desolve events can take the values "add" to add dose C1 <- C1 + dose,
# "replace" to change the value C1 <- dose
# or "multiply" to change the value to C1 <- C1*dose
    "dose.type" = "add"),
  "iv" = list(
    "entry.compartment" = "Aven",
    "dose.type" = "add")
  )
```

The remaining R code has functions that are independent of the compiled code and are used to call httk specific functions.

```
model.list[["pbtk"]]$compartment.units <- c(
    "Agutlumen"="umol",
    "Agut"="umol",
    "Aliver"="umol",
    "Aven"="umol",
    "Alung"="umol",
    "Aart"="umol",
    "Arest"="umol",
    "Akidney"="umol",
    "Atubules"="umol",
    "Ametabolized"="umol",
    "Cgut"="uM",
    "Cliver"="uM",
    "Cven"="uM",
    "Clung"="uM",
    "Cart"="uM",
    "Crest"="uM",
    "Ckidney"="uM",
    "Cplasma"="uM",
```

```
    "Aplasma"="umol",
    "AUC"="uM*days"
  )


#Parameters needed to make a prediction (this is used by get_cheminfo):
model.list[["pbtk"]]$required.params <- c(
  "Clint",
  "Funbound.plasma",
  "Pow",
  "pKa_Donor",
  "pKa_Accept",
  "MW"
  )

# Function for calculating Clmetabolismc after Clint is varied:
model.list[["pbtk"]]$propagateuv.func <- "propagate_invitrouv_pbtk"

# If httk-pop is enabled:
# Function for converting httk-pop physiology to model parameters:
model.list[["pbtk"]]$convert.httkpop.func <- NULL

# We want all the standard physiological calculations performed:
model.list[["pbtk"]]$calc.standard.httkpop2httk <- TRUE

# These are the model parameters that are impacted by httk-pop:
model.list[["pbtk"]]$httkpop.params <- c(
  "BW",
  "Fgutabs",
  "hematocrit",
  "liver.density",
  "million.cells.per.gliver",
  "Qcardiacc",
  "Qgfrc",
  "Qgutf",
  "Qkidneyf",
  "Qliverf",
  "Rblood2plasma",
  "Vartc",
  "Vgutc",
  "Vkidneyc",
  "Vliverc",
  "Vlungc",
  "Vrestc",
  "Vvenc")
```

```
# Do we need to recalculate partition coefficients when doing Monte Carlo?
model.list[["pbtk"]]$calcpc <- TRUE


# Do we need to recalculate first pass metabolism when doing Monte Carlo?
model.list[["pbtk"]]$firstpass <- FALSE

# Do we ignore the Fups where the value was below the limit of detection?
model.list[["pbtk"]]$exclude.fup.zero <- TRUE

# These are the parameter names needed to describe steady-state dosing:
model.list[["pbtk"]]$css.dosing.params <- c("hourly.dose")

# Filter out volatile compounds with Henry's Law Constant Threshold
model.list[["pbtk"]]$log.henry.threshold <- c(-4.5)

# Filter out compounds belonging to select chemical classes
model.list[["pbtk"]]$chem.class.filt <- c("PFAS")
```

2.2 Modifications to our model

If you want to modify this basic structure, you need to understand the components you are starting with. For example, the structure can be changed by un-lumping additional organs from the rest of the body and making them into separate organs. Another example of change includes adding metabolic complexity by changing linear clearance to add multiple enzymatic pathways. The rest of the document will show you the basics present in the pbtk model and how to modify the code to include closed chamber inhalation by adding an additional ODE to the existing code. For a closed chamber system, the PBTK model needs an additional equation describing mass balance occurring inside the chamber:

where, Qp is the ventilation rate in L/hr Cinh is the inhaled chemical concentration in ppm (which is converted to mg/L), and Cex is the exhaled chemical concentration in ppm (which is converted to mg/L).

In addition to the chamber equation, the closed inhalation PBTK model has a different structure than PBTK.

The changes included: Adipose tissue is a separate organ from the Rest of the Body compartment. Slowly perfused consists of the sum of muscle,skin, and bone compartments. Default time units in httk are days, while default units in closed chamber are hours.The PBTK model's new"rest of the body" will include the remaining tissues not used in slowly perfused.

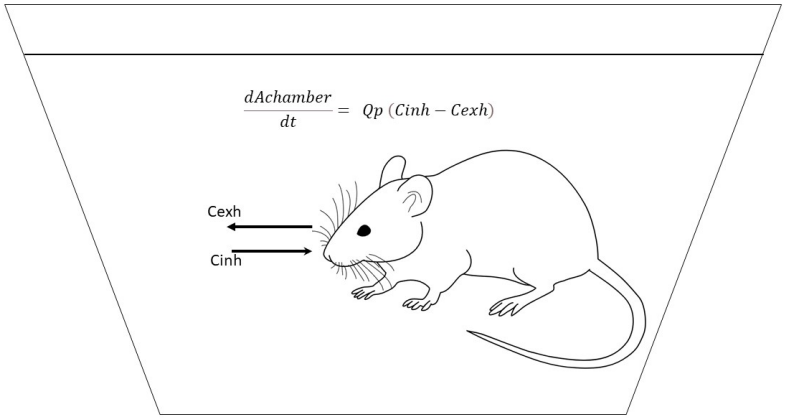


Figure 2.2: Closed chamber inhalation and equation

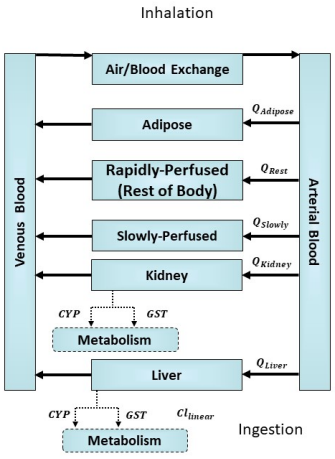


Figure 2.3: Closed chamber inhalation model schematic

First, choose a name for the modelinfo function that is descriptive. We selected modelinfo_gas_closed_pbtk for this example, where name=gas_closed_pbtk.The user needs to create a corresponding parameterize_name and solve_name functions.

```
model.list[["gas_closed_pbtk"]]$parameterize.func <- "parameterize_gas_closed_pbtk"

model.list[["gas_closed_pbtk"]]$solve.func <- "solve_gas_closed_pbtk"
```

All of the tissues are listed below. Because our gas_closed_pbtk model only include some of these tissues, we must include the "tissuelist", which lists the explicit tissues we need for the model. Closed inhalation has lumped parameters in addition to those listed in the original rest of the body. One example is slowly perfused tissue, the sum of muscle, bone, and skin. The code below shows how to lump these tissues in a vector. The other individual tissues in the model (adipose, liver, kidney) are left separate. The model's "richly perfused tissue" will be considered "rest" of the body and include the remaining tissues not used.

```
model.list[["gas_closed_pbtk"]]$alltissues=c( "adipose", "bone",
"brain",
"gut",
"heart",
"kidney",
"liver",
"lung",
"muscle", "skin",
"spleen",
"red blood cells", "rest")

model.list[["gas_closed_pbtk"]]$tissuelist=list(
liver=c("liver"),
kidney=c("kidney"),
adipose=c("adipose"),
slowly =c("muscle","skin","bone") )
```

Next, we need to describe all model parameters. In some cases there are common parameters to both inhalation models (such as blood flow and tissue volumes). The additional parameters will have a naming convention that is different from those normally used in httk. This is resolved in the Rtosolvermap code by listing all parameters with their associated variable names. However, some parameters are model-specific and are not found by other httk functions. These model-specific parameters will have their own naming conventions. Please note that any of the "common" parameters must have the httk naming convention, while the model-specific parameter names can have the names used in the .model code.

Below is a list of parameters (param.names) which are returned by the R model parameterize function (which will be discussed in the next section). Some of these parameters are not directly used to solve the equations,but describe how other parameters were calculated. In param.names, the parameter names which are "common" should follow the httk naming convention while the model-specific parameter names can remain the same. Include only initial parameters; do not include "calculated parameters" or parameters that are calculated from other parameters within the .C file. An example for a .model specific parameter is KRESYN, a constant used to describe suicide inhibition and not usually included in httk. The same name (KRESYN) can be used when adding this parameter to httk.

```
model.list[["gas_closed_pbtk"]]$param.names <- c(
   "BW",
  "Fgutabs",
  "Kadipose2air",
  "Kblood2air",
  "Kkidney2air",
  "Kliver2air",
  "Krest2air",
  "MW",
  "Pow",
  "Qadiposef" ,
  "Qalvc",
  "Qcardiacc",
  "Qkidneyf",
  "Qrestf",
  "Qslowlyf",
  "Vadiposec" ,
  "Vkidneyc",
  "Vliverc",
  "Vrestc",
  "RATS",
  "VCHC",
  "Qslowlyf",
  "Qrestf",
  "VMAXC",
  "KM",
  "VMRATIO",
  "KLOSS",
  "KRESYN",
  "KAS"
  )
```

Left panel is the C code for a closed inhalation pbtk model C file. The parameter names are the original names assigned in the .model file.

```
/* Parameters */
static double parms[52];

#define BW parms[0]
#define RATS parms[1]
#define VCHC parms[2]
#define QCC parms[3]
#define QPC parms[4]
#define QFC parms[5]
#define QLC parms[6]
#define QKC parms[7]
#define QSC parms[8]
#define QRC parms[9]
#define VFC parms[10]
#define VLC parms[11]
#define VKC parms[12]
#define VRC parms[13]
#define VSC parms[14]
#define PFA parms[15]
#define PLA parms[16]
#define PKCTXA parms[17]
#define PKMEDA parms[18]
#define PSA parms[19]
#define PRA parms[20]
#define PB parms[21]
#define VMAXC parms[22]
#define KM parms[23]
#define VMRATIO parms[24]
#define KLOSS parms[25]
#define KRESYN parms[26]
#define KAS parms[27]
#define MW parms[28]
#define QC parms[29]
#define QP parms[30]
#define QF parms[31]
#define QL parms[32]
#define QKCTX parms[33]
#define QKMED parms[34]
#define QS parms[35]
#define QR parms[36]
#define VF parms[37]
#define VL parms[38]
#define VKCTX parms[39]
#define VKMED parms[40]
#define VS parms[41]
```

```
#define VR parms[42]
#define PF parms[43]
#define PL parms[44]
#define PKCTX parms[45]
#define PKMED parms[46]
#define PS parms[47]
#define PR parms[48]
#define VMAX parms[49]
#define VMAXK parms[50]
#define VCH parms[51]
```

The R list uses the C code names on the left(=) and the httk parameter names on the right. Httk parameters are enclosed with " ".

```
model.list[["gas_closed_pbtk"]]$Rtosolvermap<- list(
  BW = "BW",
  RATS = "RATS",
  VCHC = "VCHC",
  QCC = "Qcardiacc",
  QPC = "Qalvc",
  QFC = "Qadiposef",
  QLC = "Qtotal.liverf",
  QKC = "Qkidneyf",
  QSC = "Qslowlyf",
  QRC="Qrestf",
  VFC = "Vadiposec",
  VLC = "Vliverc",
  VKC = "Vkidneyc",
  VRC = "Vrestc",
  VSC = "Vslowlyc",
  PFA = "Kadipose2air",
  PLA = "Kliver2air",
  PKCTXA = "Kkidney2air",
  PKMEDA = "Kkidney2air",
  PSA = "Kslowly2air",
  PRA = "Krest2air",
  PB = "Kblood2air",
  VMAXC = "VMAXC",
  KM = "KM",
  VMRATIO = "VMRATIO",
  KLOSS = "KLOSS",
  KRESYN = "KRESYN",
  KAS = "KAS",
  MW = "MW"
)
```

Next step is to list the state variables in the same order as listed in the c code.

```
  19 States:
    ACH = 0.0,
    AX = 0.0,
    AINH = 0.0,
    AF = 0.0,
    AS = 0.0,
    AR = 0.0,
    AKMED = 0.0,
    ORALDR = 0.0,
    AINORAL = 0.0,
    AST = 0.0,
    AL = 0.0,
    AML = 0.0,
    VMAXT = 0.0,
    AKCTX = 0.0,
    AMK = 0.0,
    VMAXKT = 0.0,
    AUC = 0.0,
    AUCA = 0.0,
    CIN = 0.0,
```

List of state variables in r code using same order as in c file.

```
model.list[["gas_closed_pbtk"]]$state.vars <- c(
  "ACH",
  "AX",
  "AINH",
  "AF",
  "AS",
  "AR",
  "AKMED",
  "ORALDR",
  "AINORAL",
  "AST",
  "AL",
  "AML",
  "VMAXT",
  "AKCTX",
  "AMK",
  "VMAXKT",
  "AUC",
  "AUCA",
  "CIN"
    )
```

Initialization of parameters in the C code to values at start of the simulations. The calculation of additional parameters is also included in this section.

```
void getParms_gas_closed_pbtk (double *inParms, double *out, int *nout) {
/*----- Model scaling */

  int i;

  for (i = 0; i < *nout; i++) {
    parms[i] = inParms[i];
  }


  QC = QCC * pow ( BW , 0.74 ) *24;
  QP = QPC * pow ( BW , 0.74 )  *24;

  QF = QFC * QC ;
  QL = QLC * QC ;
  QKCTX = 0.9 * QKC * QC ;
  QKMED = 0.1 * QKC * QC ;
  QS = QSC * QC ;
  QR = QRC * QC ;

  VF = VFC * BW ;
  VL = VLC * BW ;
  VKCTX = 0.7 * VKC * BW ;
  VKMED = 0.3 * VKC * BW ;
  VS = VSC * BW ;
  VR = VRC * BW ;
  PF = PFA / PB ;
  PL = PLA / PB ;
  PKCTX = PKCTXA / PB ;
  PKMED = PKMEDA / PB ;
  PS = PSA / PB ;
  PR = PRA / PB ;
  VMAX = VMAXC * pow ( BW , 0.7 )  ;
  VMAXK = VMAX * VMRATIO ;
  VCH = ( VCHC ? VCHC - BW * RATS : 0 ) ;
  VCH = ( VCH < 0 ? 0 : VCH ) ;

  for (i = 0; i < *nout; i++) {
    out[i] = parms[i];
  }
  }
```

Initialization of parameters in C code - function name in his example, getParmspbtk must be the same in C and R code. These function names are selected by the user.

```
model.list[["gas_closed_pbtk"]]$compiled.parameters.init <- "getParms_gas_closed_pbtk"

compiled_parameters_init <- "getParms_gas_closed_pbtk"
```

Initializing the parameters in the C code:

```
Next, we set the function to initialize the state vectors for the compiled model.
   52 Parameters:
     BW = 0.202,
     RATS = 1,
     VCHC = 3.8,
     QCC = 14,
     QPC = 14,
     QFC = 0.09,
     QLC = 0.25,
     QKC = 0.141,
     QSC = 0.15,
     QRC = 0.369,
     VFC = 0.07,
     VLC = 0.04,
     VKC = 0.0073,
     VRC = 0.0427,
     VSC = 0.7615,
     PFA = 352,
     PLA = 17.7,
     PKCTXA = 11.0,
     PKMEDA = 11.0,
     PSA = 17.7,
     PRA = 17.7,
     PB = 17.7,
     VMAXC = 5.218,
     KM = 0.12,
     VMRATIO = 0.008,
     KLOSS = 0.0,
     KRESYN = 0.0,
     KAS = 0.083,
     MW = 119.38,
     QC = 0.0,
     QP = 0.0,
     QF = 0.0,
     QL = 0.0,
```

```
    QKCTX = 0.0,
    QKMED = 0.0,
    QS = 0.0,
    QR = 0.0,
    VF = 0.0,
    VL = 0.0,
    VKCTX = 0.0,
    VKMED = 0.0,
    VS = 0.0,
    VR = 0.0,
    PF = 0.0,
    PL = 0.0,
    PKCTX = 0.0,
    PKMED = 0.0,
    PS = 0.0,
    PR = 0.0,
    VMAX = 0.0,
    VMAXK = 0.0,
    VCH = 0.0,
*/
```

Initializing the parameters in the R code:

```
model.list[["gas_closed_pbtk"]]$compiled.param.names <- c(
  "BW",
  "RATS",
  "VCHC",
  "QCC",
  "QPC",
  "QFC",
  "QLC",
  "QKC",
  "QSC",
  "QRC",
  "VFC",
  "VLC",
  "VKC",
  "VRC",
  "VSC",
  "PFA",
  "PLA",
  "PKCTXA",
  "PKMEDA",
  "PSA",
  "PRA",
```

```
  "PB",
  "VMAXC",
  "KM",
  "VMRATIO",
  "KLOSS",
  "KRESYN",
  "KAS",
  "MW",
  "QC",
  "QP",
  "QF",
  "QL",
  "QKCTX",
  "QKMED",
  "QS",
  "QR",
  "VF",
  "VL",
  "VKCTX",
  "VKMED",
  "VS",
  "VR",
  "PF",
  "PL",
  "PKCTX",
  "PKMED",
  "PS",
  "PR",
  "VMAX",
  "VMAXK",
  "VCH"
)
```

Initialization model in c code must have same name as in the R code.

```
/*----- Initializers */
void initmod_gas_closed_pbtk (void (* odeparms)(int *, double *))
{
  int N=52;
  odeparms(&N, parms);
}
```

Initialization code in R must contain the model name: _gas_closed_pbtk

```
model.list[["gas_closed_pbtk"]]$compiled.init.func <-"initmod_gas_closed_pbtk"
```

The derivative section is calculated by calling the C code listing the derivative equations. This section is compiled using c code for speed.

```
void derivs_gas_closed_pbtk (int *neq, double *pdTime, double *y, double *ydot, double
{
  /* local */ double QTOT;
  /* local */ double RMK;
  /* local */ double RML;

  yout[ID_CF] = y[ID_AF] / VF ;

  yout[ID_CVF] = yout[ID_CF] / PF ;

  yout[ID_CS] = y[ID_AS] / VS ;

  yout[ID_CVS] = yout[ID_CS] / PS ;

  yout[ID_CR] = y[ID_AR] / VR ;

  yout[ID_CKMED] = y[ID_AKMED] / VKMED ;

  yout[ID_CVKMED] = yout[ID_CKMED] / PKMED ;

  yout[ID_CVR] = yout[ID_CR] / PR ;

  yout[ID_CL] = y[ID_AL] / VL ;

  yout[ID_CVL] = yout[ID_CL] / PL ;

  yout[ID_CKCTX] = y[ID_AKCTX] / VKCTX ;

  yout[ID_CVKCTX] = yout[ID_CKCTX] / PKCTX ;

  yout[ID_CV] = ( QF * yout[ID_CVF] + QL * yout[ID_CVL] + QKMED * yout[ID_CVKMED] + QK

  yout[ID_CI] = ( VCH ? y[ID_ACH] / VCH : y[ID_CIN] ) ;

  yout[ID_CA] = ( QP * yout[ID_CI] + QC * yout[ID_CV] ) / ( QC + ( QP / PB ) ) ;

  yout[ID_CX] = yout[ID_CA] / PB ;

  yout[ID_AXC] = y[ID_AX] * RATS ;
```

```
yout[ID_CK] = ( y[ID_AKCTX] + y[ID_AKMED] ) / ( VKMED + VKCTX ) ;

yout[ID_FAT_PRED] = yout[ID_CF] * 0.9 ;

yout[ID_AINHC] = y[ID_AINH] * RATS ;

yout[ID_AMET] = y[ID_AML] + y[ID_AMK] ;

yout[ID_AOUT] = yout[ID_AMET] + y[ID_AX] ;

yout[ID_AIN] = y[ID_AINORAL] + y[ID_AINH] ;

yout[ID_ATOT] = y[ID_AF] + y[ID_AL] + y[ID_AKCTX] + y[ID_AKMED] + y[ID_AR] + y[ID_AS] + y[ID_AS

yout[ID_MASSBALANCE] = yout[ID_AIN] - ( yout[ID_ATOT] + yout[ID_AOUT] ) ;

QTOT = QF + QL + QKCTX + QKMED + QR + QS ;

yout[ID_FLOWBALANCE] = QC - QTOT ;

ydot[ID_ACH] = RATS * QP * ( yout[ID_CX] - yout[ID_CI] ) -0.0298 * y[ID_ACH] ;

ydot[ID_AX] = QP * yout[ID_CX] ;

ydot[ID_AINH] = QP * yout[ID_CI] ;

ydot[ID_AF] = QF * ( yout[ID_CA] - yout[ID_CVF] ) ;

ydot[ID_AS] = QS * ( yout[ID_CA] - yout[ID_CVS] ) ;

ydot[ID_AR] = QR * ( yout[ID_CA] - yout[ID_CVR] ) ;

ydot[ID_AKMED] = QKMED * ( yout[ID_CA] - yout[ID_CVKMED] ) ;

ydot[ID_ORALDR] = 0 ;

ydot[ID_AINORAL] = y[ID_ORALDR] ;

ydot[ID_AST] = y[ID_ORALDR] - KAS * y[ID_AST] ;

RMK = y[ID_VMAXKT] * yout[ID_CVKCTX] / ( KM + yout[ID_CVKCTX] ) *24;

ydot[ID_VMAXKT] = KRESYN * ( VMAXK - y[ID_VMAXKT] ) - KLOSS * ( y[ID_VMAXKT] / VKCTX ) * RMK ;

ydot[ID_AKCTX] = QKCTX * ( yout[ID_CA] - yout[ID_CVKCTX] ) - RMK ;
```

```
  ydot[ID_AMK] = RMK ;

  RML = y[ID_VMAXT] * yout[ID_CVL] / ( KM + yout[ID_CVL] ) *24;

  ydot[ID_VMAXT] = KRESYN * ( VMAX  - y[ID_VMAXT] ) - KLOSS * ( y[ID_VMAXT] / VL ) * RM

  ydot[ID_AL] = ( KAS * y[ID_AST] ) + QL * ( yout[ID_CA] - yout[ID_CVL] ) - RML ;

  ydot[ID_AML] = RML ;

  ydot[ID_AUC] = yout[ID_CV]*24 ;

  ydot[ID_AUCA] = yout[ID_CA]*24 ;

  ydot[ID_CIN] = 0.0 ;

} /* derivs */
```

The R code function calling the derivatives in C code must contain the same function name.

```
model.list[["gas_closed_pbtk"]]$derivative.func <- "derivs_gas_closed_pbtk"
```

The list of outputs in the C code must be entered in the same order inside R.

```
   26 Outputs:
    "CF",
    "CVF",
    "CS",
    "CVS",
    "CR",
    "CKMED",
    "CVKMED",
    "CVR",
    "CL",
    "CVL",
    "CKCTX",
    "CVKCTX",
    "CK",
    "CV",
    "CA",
    "CI",
    "CX",
    "AXC",
```

```
    "AINHC",
    "AMET",
    "AOUT",
    "AIN",
    "ATOT",
    "MASSBALANCE",
    "FLOWBALANCE",
    "FAT_PRED",
```

The output listed in the R function must be the same as the list in the C code.

```
model.list[["gas_closed_pbtk"]]$derivative.output.names <- c(
  "CF",
  "CVF",
  "CS",
  "CVS",
  "CR",
  "CKMED",
  "CVKMED",
  "CVR",
  "CL",
  "CVL",
  "CKCTX",
  "CVKCTX",
  "CK",
  "CV",
  "CA",
  "CI",
  "CX",
  "AXC",
  "AINHC",
  "AMET",
  "AOUT",
  "AIN",
  "ATOT",
  "MASSBALANCE",
  "FLOWBALANCE",
 "FAT_PRED"
  )
```

Next comes a series of code that was designed for httk and is written in R. There is no C counterpart to this section. All additional parameters and conversion units have been added in the following sections.

```
#list of variables to be monitored (plotted). This list should be able to be
#constructed from states and outputs.
model.list[["gas_closed_pbtk"]]$default.monitor.vars <- c(
  "CF",
  "CVF",
  "CS",
  "CVS",
  "CR",
  "CKMED",
  "CVKMED",
  "CVR",
  "CL",
  "CVL",
  "CKCTX",
  "CVKCTX",
  "CK",
  "CV",
  "CA",
  "CI",
  "CX",
  "AXC",
  "AINHC",
  "AMET",
  "AOUT",
  "AIN",
  "ATOT",
  "MASSBALANCE",
   "FLOWBALANCE",
  "FAT_PRED",
  "ACH",
  "AX",
  "AINH",
  "AF",
  "AS",
  "AR",
  "AKMED",
  "ORALDR",
  "AINORAL",
  "AST",
 "AL",
 "AML",
 "VMAXT",
 "AKCTX",
 "AMK",
 "VMAXKT",
```

```
  "AUC",
  "AUCA",
  "CIN"
  )

# Allowable units assigned to dosing input:
model.list[["gas_closed_pbtk"]]$allowed.units.input <- list(
    "oral" = c('umol','mg','mg/kg'),
    "iv" = c('umol','mg','mg/kg'),
    "inhalation" = c('ppmv','mg/L','mg/m^3','uM','umol','mg'))

# Allowable units assigned to entries in the output columns of the ode system
model.list[["gas_closed_pbtk"]]$allowed.units.output <- list(
       "oral" = c('uM','mg/L','ppmv','umol','mg','uM*days',
                  'mg/L*days','mg/m^3','mg/m^3*days'),
       "iv" = c('uM','mg/L','ppmv','umol','mg','uM*days','mg/L*days',
                'mg/m^3','mg/m^3*days'),
       "inhalation" = c('uM','mg/L','ppmv','umol','mg','uM*days','mg/L*days',
                        'mg/m^3','mg/m^3*days','mg/days','L/days'))

# Actual (intrinsic) units assigned to each of the time dependent
# variables of the model system including state variables and any transformed
# outputs (for example, concentrations calculated from amounts.)
# AUC values should also be included.
model.list[["gas_closed_pbtk"]]$compartment.units <- c( #missing States: Ainh and Aexh - AM, 1/21
  "CF"="mg/L",
  "CVF"="mg/L",
  "CS"= "mg/L",
  "CVS" = "mg/L",
  "CR" = "mg/L",
  "CKMED" = "mg/L",
  "CVKMED" ="mg/L",
  "CVR" = "mg/L",
  "CL" = "mg/L",
  "CVL" ="mg/L",
  "CKCTX"= "mg/L",
  "CVKCTX"= "mg/L",
  "CK"="mg/L",
  "CV" ="mg/L",
  "CA" ="mg/L",
  "CI" = "mg/L",
  "CX" = "mg/L",
  "AXC" = "mg",
  "AINHC" = "mg",
  "AMET" = "mg",
```

```
  "AOUT" = "mg",
  "AIN" = "mg",
  "ATOT" ="mg",
  "MASSBALANCE" = "mg",
  "FLOWBALANCE" = "mg", #L/day
  "FAT_PRED"="mg/L",
  "ACH" = "mg",
  "AX" = "mg",
  "AINH"="mg",
  "AF" ="mg",
  "AS" ="mg",
  "AR" ="mg",
  "AKMED"="mg",
  "ORALDR"="mg",
   "AINORAL"="mg",
   "AST" ="mg",
  "AL" ="mg",
  "AML"="mg",
  "VMAXT"="mg", #should be mg/d
  "AKCTX" ="mg",
   "AMK"="mg",
  "VMAXKT"= "mg", #should be mg/d
  "AUC"= "mg/L*days",
  "AUCA"="mg/L*days",
  "CIN"="mg/L"
  )

# These parameters specify the exposure scenario simulated by the model:
model.list[["gas_closed_pbtk"]]$dosing.params <- c(
  "initial.dose",
  "daily.dose",
  "doses.per.day",
  "dosing.matrix"
  )

model.list[["gas_closed_pbtk"]]$routes <- list(
  "oral" = list(
# We need to know which compartment gets the dose
    "entry.compartment" = "AST",
# desolve events can take the values "add" to add dose C1 <- C1 + dose,
# "replace" to change the value C1 <- dose
# or "multiply" to change the value to C1 <- C1*dose
    "dose.type" = "replace"),
  "inhalation" = list(
    "entry.compartment" = "ACH",
```

```
    "dose.type" = "replace")
  )
```

::: Rest of the R code is unaltered from the original httk package.

```
# #Parameters needed to make a prediction (this is used by get_cheminfo):
model.list[["gas_closed_pbtk"]]$required.params <- c(
  "Pow",
  "MW",
  "logHenry"
   )

# Do we ignore the Fups where the value was below the limit of detection?
model.list[["gas_closed_pbtk"]]$exclude.fup.zero <- TRUE
```

2.3 Modifications to init.c file

Once the .c file has been modified, it must be incorporated into httk. Move the .c file into the httk/src folder. Within httk/src is a file named init.c. This file must be modified as well.The names of all the new _closed_gas_model functions have to be included in the init.c file. Please look at the highlighted statements for examples.

#include <R.h>

#include <Rinternals.h>

#include <stdlib.h> // for NULL

#include <R_ext/Rdynload.h>

/* FIXME:

Check these declarations against the C/Fortran source code.

*/

/* .C calls */

extern void getParmspbtk(double *, double *, int *);

extern void getParms1comp(double *, double *, int *);

extern void getParms3comp(double *, double *, int *);

extern void getParms_gas_pbtk(double *, double *, int *);

extern void getParms_gas_closed_pbtk(double *, double *, int *);}

extern void getParmsfetal_pbtk(double *, double *, int *);

```
extern void initmodpbtk(void *);

extern void initmod1comp(void *);

extern void initmod3comp(void *);

extern void initmod_gas_pbtk(void *);

extern void initmod_gas_closed_pbtk(void *);}

extern void initmodfetal_pbtk(void *);

extern void derivspbtk(int *, double *, double *, double *, double *, int *);

extern void derivs1comp(int *, double *, double *, double *, double *, int *);

extern void derivs3comp(int *, double *, double *, double *, double *, int *);

extern void derivs_gas_pbtk(int *, double *, double *, double *, double *, int
*);

extern void derivs_gas_closed_pbtk(int *, double *, double *, double *, double
*, int *);}

extern void derivsfetal_pbtk(int *, double *, double *, double *, double *, int
*);

extern void jacpbtk(int *, double *, double *, int *, int *, double *, int *, double
*, int *);

extern void jac1comp(int *, double *, double *, int *, int *, double *, int *,
double *, int *);

extern void jac3comp(int *, double *, double *, int *, int *, double *, int *,
double *, int *);

extern void jac_gas_pbtk(int *, double *, double *, int *, int *, double *, int
*, double *, int *);

extern void jac_gas_closed_pbtk(int *, double *, double *, int *, int *, double
*, int *, double *, int *);}

extern void jacfetal_pbtk(int *, double *, double *, int *, int *, double *, int *,
double *, int *);

extern void eventpbtk(int *, double *, double *);

extern void event1comp(int *, double *, double *);

extern void event3comp(int *, double *, double *);

extern void event_gas_pbtk(int *, double *, double *);

extern void event_gas_closed_pbtk(int *, double *, double *);}

extern void eventfetal_pbtk(int *, double *, double *);

extern void rootpbtk (int *, double *, double *, int *, double *, double *, int *);
```

extern void root1comp (int *, double *, double *, int *, double *, double *, int *);

extern void root3comp (int *, double *, double *, int *, double *, double *, int *);

extern void root_gas_pbtk (int *, double *, double *, int *, double *, double *, int *);

extern void root_gas_closed_pbtk (int *, double *, double *, int *, double *, double *, int *);}

extern void rootfetal_pbtk (int *, double *, double *, int *, double *, double *, int *);

static const R_CMethodDef CEntries[] = {

{“getParmspbtk”, (DL_FUNC) &getParmspbtk, 3},

{“getParms1comp”, (DL_FUNC) &getParms1comp, 3},

{“getParms3comp”, (DL_FUNC) &getParms3comp, 3},

{“getParms_gas_pbtk”, (DL_FUNC) &getParms_gas_pbtk, 3},

{“getParms_gas_closed_pbtk”, (DL_FUNC) &getParms_gas_closed_pbtk, 3},}

{“getParmsfetal_pbtk”, (DL_FUNC) &getParmsfetal_pbtk, 3},

// {“getParms”, (DL_FUNC) &getParms, 4},

{“initmodpbtk”, (DL_FUNC) &initmodpbtk, 1},

{“initmod1comp”, (DL_FUNC) &initmod1comp, 1},

{“initmod3comp”, (DL_FUNC) &initmod3comp, 1},

{“initmod_gas_pbtk”, (DL_FUNC) &initmod_gas_pbtk, 1},

{“initmod_gas_closed_pbtk”, (DL_FUNC) &initmod_gas_closed_pbtk, 1},}

{“initmodfetal_pbtk”, (DL_FUNC) &initmodfetal_pbtk, 1},

{“derivspbtk”, (DL_FUNC) &derivspbtk, 6},

{“derivs1comp”, (DL_FUNC) &derivs1comp, 6},

{“derivs3comp”, (DL_FUNC) &derivs3comp, 6},

{“derivs_gas_pbtk”, (DL_FUNC) &derivs_gas_pbtk, 6},

{“derivs_gas_closed_pbtk”, (DL_FUNC) &derivs_gas_closed_pbtk, 6},}

{“derivsfetal_pbtk”, (DL_FUNC) &derivsfetal_pbtk, 6},

{“jacpbtk”, (DL_FUNC) &jacpbtk, 9},

{“jac1comp”, (DL_FUNC) &jac1comp, 9},

```
{“jac3comp”, (DL_FUNC) &jac3comp, 9},
{“jac_gas_pbtk”, (DL_FUNC) &jac_gas_pbtk, 9},
{“jac_gas_closed_pbtk”, (DL_FUNC) &jac_gas_closed_pbtk, 9},}
{“jacfetal_pbtk”, (DL_FUNC) &jacfetal_pbtk, 9},
{“eventpbtk”, (DL_FUNC) &eventpbtk, 3},
{“event1comp”, (DL_FUNC) &event1comp, 3},
{“event3comp”, (DL_FUNC) &event3comp, 3},
{“event_gas_pbtk”, (DL_FUNC) &event_gas_pbtk, 3},
{“event_gas_closed_pbtk”, (DL_FUNC) &event_gas_closed_pbtk, 3},
{“eventfetal_pbtk”, (DL_FUNC) &eventfetal_pbtk, 3},
{“rootpbtk”, (DL_FUNC) &rootpbtk, 7},
{“root1comp”, (DL_FUNC) &root1comp, 7},
{“root3comp”, (DL_FUNC) &root3comp, 7},
{“root_gas_pbtk”, (DL_FUNC) &root_gas_pbtk, 7},
{“root_gas_closed_pbtk”, (DL_FUNC) &root_gas_closed_pbtk, 7},}
{“rootfetal_pbtk”, (DL_FUNC) &rootfetal_pbtk, 7},
{NULL, NULL, 0}
};
void R_init_httk(DllInfo *dll)
{
R_registerRoutines(dll, CEntries, NULL, NULL, NULL);
R_useDynamicSymbols(dll, TRUE);
}
///*----- Initializers */
//void getParms(void *parmsInit(double *, double *, int *), double *inParms,
double *out, int *nout)
//{
// parmsInit(inParms, out, nout);
//}
```

Chapter 3

Parameterizing a New Model

The function parameterize_MODELNAME() is used to generate all parameters for the model. The function call requires the chemical name, CAS, or dtxsid, the species default.to.human (which generates human parameters if another species' parameters are unavailable), the tissue list from modelinfo_MODELNAME(), and any other options specific to the new model.

Many parameters are accessible within httk. These parameters follow a naming convention. For example, fractional flow rates start with Q, fractional volumes begin with V, and partition coefficients begin with K. Below is a list of commonly used parameters and their httk naming conventions.

Parameter Name	Description
BW	Body weight (kg)
hematocrit	Percent volume of red blood cells in the blood.
	Blood/flow rates
Qcardiacc	Cardiac Output, L/h/kg BW^3/4
Qgfrc	Glomerular Filtration Rate, L/h/kg BW^3/4, volume of fluid filtered from kidney and excreted.
	Fraction of cardiac output flowing to...
Qgutf	the gut.
Qkidneyf	the kidneys.
Qadiposef	adipose tissue.
Qbrainf	brain tissue.
Qbonef	bone.
Qlungf	the lungs.

Parameter Name	Description
Qmusclef	muscle.
Qskinf	the skin.
Qspleenf	the spleen.
Qheartf	the heart.
Qrestf	the rest of the body.
	Volume of the tissue per kg body weight (L/kg BW); i.e.fractional tissue volumes of...
Vgutc	the gut.
Vkidneyc	the kidneys.
Vliverc	the liver.
Vadiposec	adipose tissue.
Vbrainc	brain tissue.
Vbonec	bone.
Vlungc	the lungs.
Vmusclec	muscle.
Vskinc	the skin.
Vspleenc	the spleen.
Vheartc	the heart.
Vrestc	the rest of the body.
Vvenc	Volume of the veins per kg body weight, L/kg BW.
Vartc	Volume of the arteries per kg body weight, L/kg BW.
	Tissue to plasma partition coefficients (Ratio of concentration of chemical in tissue to unbound concentration in plasma.)
Kgut2pu	the gut.
Kkidney2pu	the kidneys.
Kliver2pu	the liver.
Kadipose2pu	adipose tissue.
Kbrain2pu	brain tissue.
Kbone2pu	bone.
Klung2pu	the lungs.
Kmuscle2pu	muscle.
Kskin2pu	the skin.
Kspleen2pu	the spleen.
Kheartf	the heart.
Krest2pu	the rest of the body.
Krbc2pu	red blood cells.
	Gut parameters

Parameter Name	Description
kgutabs	Rate that chemical enters the gut from gutlumen, 1/h, (defaults to 2.18 1/h from Wambaugh et al. (2018); not currently generated within httk)
Fabsgut	Fraction of the oral dose absorbed, i.e. the fraction of the dose that enters the gutlumen.
	Other chemical-specific parameters
Clmetabolismc	Hepatic Clearance (L/h/kg BW).
Rblood2plasma	The ratio of the concentration of the chemical in the blood to the concentration in the plasma from available_rblood2plasma.
Funbound.plasma	Fraction of plasma that is not bound.
MW	Molecular Weight (g/mol).

3.1 Core httk Data Sets

A main component of "httk" to enable modularity are the parameter data tables which are used to avoid hard coding model parameter values. Instead, parameter values should be stored in data tables within "httk", then retrieved with the appropriate 'core' functions where necessary. Several key data files are provided within the "httk" R package for parameterizing and evaluating the generic HTTK models. These tables are stored in the Tables.RData file in the "httk/data" sub-directory. The following data tables are automatically loaded to your local R session when the "httk" package is called from the local R library via "library(httk)":

Data tables	Description
chem.physical_and_invitro.data	Chemical-specific values for physico-chemical properties and in vitro measurements.
physiology.data	Data describing species-specific physiology.
tissue.data	Tissue composition data for Schmitt's method of partition coefficient prediction.
chem.invivo.PK.data	Chemical concentration vs. time data (CvTdb) for evaluating model predictions.

3.2 Core functions for model parameterization of generic TK models

While most models should provide and use their own function for generating and retrieving chemical-specific model parameters (parameterize_MODELNAME().R) they may rely on the function parameterize_pbtk(). The parameterize_pbtk() function can be customized to create a model specific function by specifying which tissues are lumped together into various compartments. For example, parameterize_3comp() makes use of parameterize_pbtk() to obtain all necessary parameters and sets any unnecessary parameters to NULL.

Most of the in vitro measured data are from human tissues. However, for certain calculations a user may want to use these values with non-human (for example, rat, mouse, etc.) physiologies, the argument "default.to.human" is provided in many of parameterization functions to allow this option. In cases where "default.to.human" is set to TRUE the human values are used in lieu of the non-human species values.

There are several generic functions for calculating key aspects of ADME (absorption, distribution, metabolism, and excretion) available for use by any model; one might need certain physio-chemical properties (for example, MW, pKa_Donor, Pka_Accept, and logP) to calculate other parameters within the parameterize_MODELNAME() function. These can be obtained by calling to the get_physchem_param function. This function and other core functions are described below.

Functions	Description
get_physchem_param	Retrieve a chemical-specific physico-chemical property from chem.physical_and_invitro.data table.Handles missing data with appropriate warning/error messages.
get_invitroPK_param	Retrieve a chemical- and species-specific in vitro-measured parameter from chem.physical_and_invitro.data table.
calc_hep_clearance	Calculate the hepatic clearance (L/h/kg BW) to a TK clearance (µL/min/106 hepatocytes) based on the in vitro clearance for various models.

Functions	Description
calc_hep_bioavailability	This function predicts "first-pass metabolism" of orally absorbed chemicals by the liver. It is only needed for models that do not include explicit routing of blood from the gut to the liver. Requires both the intrinsic clearance and the blood:plasma ratio.
available_rblood2plasma	Retrieve or estimate the chemical-specific blood:plasma ratio.
parameterize_pbtk	Generate a chemical- and species-specific set of model parameters, including tissue:plasma partition coefficients and organ volumes and flows (from table physiology.data) for an arbitrary tissue lumping scheme (tissues must be described in table tissue.data) predictions.
predict_partitioning_schmitt	Model to predict the partition coefficient from tissue to unbound plasma partition for every tissue available, using the Schmitt model.
calc_ionization	Calculate the various charged ionic species (neutral, negative, positive, Zwitter) according to the hydrogen donor and acceptor pKa's and the Henderson-Hasslebach equation.
calc_dow	Calculate the octonal:water distribution coefficient as a function of ionization .
calc_ma	Calculate the lipid bilayer membrane affinity
calc_kair	Calculate the water:air, blood:air, and mucus:air partition coefficients.
calc_fup_correction	Calculate the lipid binding correction for the fup in vitro assay.
calc_hep_fu	Calculate the fraction of unbound chemical in a hepatocyte incubation assay.
get_fabsgut	Retrieve or calculate fraction of chemical absorbed from the gut

3.3 Creating the parameterize_MODELNAME() function

The parameterize function is created with arguments. Chemical identifiers, species, and the list of tissues delineated in the modelinfo_MODELNAME file ("tissuelist") are required arguments; any others that are included can be specific to the model, although there are many inputs that are commonly found in other functions, such as default.to.human (which allows for human values to be substituted for missing species data) and suppress.messages which suppress any information or warning messages. Below is an example of arguments for the parameterize_closed_gas_pbtk() model. Notice that some parameters specific to the closed gas model (and not generic in httk) are either set to NULL or to some scalar value; these will explained more in the following paragraphs.

```
parameterize_gas_closed_pbtk <- function(chem.cas=NULL,
                                chem.name=NULL,
                                dtxsid=NULL,
                                species="Rat",
                                default.to.human=FALSE,
                                tissuelist=list(
                                  liver=c("liver"),
                                  kidney=c("kidney"),
                                  adipose=c("adipose"),
                                  slowly =c("muscle","skin","bone")),
                                suppress.messages=FALSE,
                                VMAXC=NULL,
                                KM = NULL,
                                RATS=NULL,
                                VMRATIO=0.008,
                                KLOSS=NULL,
                                KRESYN=NULL,
                                KAS=NULL,
                                VCHC=NULL,
                                ...)
```

Parameters are generated in the body of the function and set to the approproate parameter names (as discussed in the modelinfo_MODELNAME() file). This function can generate the parameters that are explicitly used to solve the model (compiled.param.names) and any additional parameters that are not used in the model but are used to generate the explicit parameters (param.names).

For example, one might need certain physio-chemical properties (for example, MW and logP) to calculate other parameters within the parameterize_MODELNAME() function. These can be obtained by calling to the get_physchem_param function and setting the returned values to appropriate

parameter names. Likewise, the ratio of chemical in the blood to plasma can be generated with the function "available_rblood2plasma()"

```
  MW <- get_physchem_param("MW",chem.cas=chem.cas) #g/mol

  # Octanol:water partition coefficient
  Pow <- 10^get_physchem_param(
    "logP",
    chem.cas=chem.cas)

# Ratio of blood to plasma
 Rblood2plasma=available_rblood2plasma(chem.cas=chem.cas,
      species=species,
      adjusted.Funbound.plasma=adjusted.Funbound.plasma,
      suppress.messages=suppress.messages)
```

Physiological parameters such as body weight, fractional flow rates, and fractional tissue volumes can all be extracted from the table "this.phys.data". Note the units of cardiac output (Qcardiacc, mL/min/kg BW^(3/4)) and convert to L/h/kgBW^(3/4).

```
  BW <- this.phys.data["Average BW"]

Qcardiacc = this.phys.data["Cardiac Output"]/1000*60
```

Now, any parameters that model-specific and not found in httk can be added to the parameterize function. These can be set to NULL in the arguments of the function or they can be set to some scalar value if the model developer requires them to be set to a default value. An example of how to execute this is discussed below.

In the closed gas model, the following parameters are unique to the model but not to httk: RATS, KLOSS, KRESYN, VHCH. RATS is the number of subjects being exposed to the chemical; it must be an integer of value 1 or larger. KLOSS and KRESYN are the second order rate constant for enzyme destruction and enzyme resynthesis rate, respectively. These are set to a default value of 0.0. KAS is the stomach absorption rate and is set to 0.0 because it is unused. VCHC is the volume of the closed chamber. If it is not input by the user, then it is set for them. We choose a default value of 15.2 L per kg BW to mimic the experiment described in [Sasso et al., 2013] for which a single rat was in a 3.8 L chamber. For multiple subjects, the volume of the chamber is calculated as 15.2 $\times$ BW $\times$ numberSubjects. If the users input VCHC themselves, we check if VCHC $\leq$ BW $\times$ numberSubjects; if it is, then the .c model defaults to make volume of air in chamber to 0, which would return 0 values for the closed chamber experiment (because it is an open

chamber experiment). If the user wishes for an open chamber experiment, then they must manually change the initial values for the CIN state (concentration inhaled).

In the following code we set default values for these parameters if they are not specified by the user in the parameterize_closed_gas function call.

```
if (is.null(RATS)){
    RATS=1
    }else if (RATS>0 & RATS==round(RATS)){
      RATS=RATS
    }else if (RATS < 0 | RATS !=round(RATS)){
      stop("Number of subjects (RATS), must be a positive integer")
    }


  if (is.null(KLOSS)){
    KLOSS=0.0
  }else{
    KLOSS=KLOSS
  }

  if (is.null(KRESYN)){
    KRESYN=0.0
  }else{
    KRESYN=KRESYN
  }

  if (is.null(KAS)){
    KAS=0.0
  }else{
    KAS=KAS
  }

  # If Volume of chamber is not specified, set volume to be comparable with Sasso's
  # experiment (3.8 L for 1 Rat); a volume of 15.2 L per kg BW.
  if (is.null(VCHC)){
    VCHC=15.2 * RATS * outlist$BW
  }else{
    VCHC=VCHC
    if(VCHC <= outlist$BW * RATS){
      warning("VCHC was set such that volume of air in chamber is set to 0 and will not
              open chamber experiments require the initial condition CIN")
    }
  }
```

Metabolic parameters VMRATIO (Vmax ratio for liver to kidney), VMAXC (maximum metabolic rate), and KM (affinity constant) are also unique to the closed gas model and not generated in httk.
VMRATIO is the ratio of VMAXC_liver and VMAXC_kidney. This ratio exists for VOCs that have split metabolism between liver and kidney, and example being chloroform. For the purpose of this example, VMRATIO is set to a default value of 0.008 in the arguments. Last, VMAXC (mg/kg/h^.7) and KM (mg/L), the metabolic parameters are needed, which are known for chloroform. The user must input VMAXC; without this, parameterize_closed_gas_pbtk will produce an error and the solvemodel function will not run. If the user knows VMAXC but not KM, a default value of KM=1 will be generated.

```
VMATIO = VMRATIO

 if(is.null(VMAXC))
   stop('Cannot compute solutions without VMAXC. Please specify a value for VMAXC')

 if(is.null(KM) & !is.null(VMAXC)){
   warning("Defaulting to KM=1")
   outlist <- c(outlist, list(
     KM=1,
     VMAXC=VMAXC))
 }
 if(!is.null(KM) & !is.null(VMAXC)){
   outlist<-c(outlist, list(
     VMRATIO = VMRATIO,
     VMAXC=VMAXC,
     KM=KM
   ))
 }
```

3.4 Adding New Data or Functions to httk

3.4.1 Creating Functions

In the event that the model builder must generate parameter values that are not already in httk, they can do so by creating their own function and calling to it inside the parameterize_MODELNAME() file. A function is a block of code which runs when it is called. Data, or parameters, can be passed into the function and it can return other data. The function() keyword must be used to create a function. In the example below, the created function is called "new_function" and will return a default value of 1 unless otherwise specified by the user. To return values, use the return() function. Once the function is created, it can be stored in httk/R

```
new_function <- function(param = 1){
  print(paste("This function returns the value param = ",param))
  return(param)
}

new_function()
new_function(param = 3)
```

3.5 Loading New Data Tables

Many data tables are stored in the Tables.RData file in the "httk/data" subdirectory and will be automatically loaded to your local R session when httk is loaded. New data is periodically incorporated into httk over time. In the event that a new model requires a new set of data to be included, one can find the script (load_package_data_tables.R) within the Git repository the sub-directory "datatables" which is the source file needed for generating the Tables.RData file.

Data files should be stored in the "datatables" subdirectory so that they can easily be called; these can be .RData files or other data files such as excel. In the following example, we will explain how to load a new data table using some data stored in an .RData file called "Example_new_data.RData". Suppose "Example_new_data.RData" contains a dataframe of information called "ExampleData_dataframe". To include this data, modify the load_package_data_tables.R file using the write.table function. At the bottom of the file is a save() function that lists all of the tables; include the name of the new tables here. Now the file can be run; this will generate two new important files: Tables.RData and sysdata.rda. Move Tables.RData to httk/data and sysdata.rda to httk/R.

```
load("Example_new_data.RData")
# Write to text so Git can track changes:
write.table(ExampleData_dataframe,file = "ExampleData_dataframe.txt",quote = F,sep = "
```

Chapter 4

Solving the New Model

4.1 Solve function

Recall that each new model should have at least three main components: the modelinfo_MODELNAME.R, parameterize_MODELNAME.R, and solve_MODELNAME.R files. This section will focus on the solve_MODELNAME.R component. This function is used to forward solve the PBTK model from the .c file, using the parameters from parameterize_MODELNAME.R. The solve function, or solve_MODELNAME.R, is a nested function that the user calls to when solving the closed gas pbtk model. Within this function is another function, solve_model, which is a shared function called to by all models.

Below is an example of the arguments for the function solve_gas_closed_pbtk. Note that these arguments contain arguments from the parameterize function as well so that a user can specify parameters. The default time ("days") that the simulation runs is 10 days; the number of time steps per hour ("tsteps") and the default is 4.

```
solve_gas_closed_pbtk <- function(chem.name = NULL,
                            chem.cas = NULL,
                            dtxsid = NULL,
                            parameters=NULL,
                            times=NULL,
                            days=10,
                            tsteps = 4, #tsteps is number of steps per hour
                            daily.dose = NULL,
                            doses.per.day = NULL,
                            dose = NULL,
                            dosing.matrix = NULL,
```

```
                        initial.values=NULL,
                        plots=FALSE,
                        suppress.messages=FALSE,
                        species="Rat",
                        route="inhalation",
                        iv.dose=FALSE,
                        input.units = "mg",#ACH units in mg
                        output.units=NULL,
                        method="lsoda",rtol=1e-8,atol=1e-12,
                        default.to.human=FALSE,
                        monitor.vars=NULL,
                        RATS=NULL,
                        VCHC=NULL,
                        VMRATIO=0.008,
                        KLOSS=NULL,
                        KRESYN=NULL,
                        KAS=NULL,
                        KM = NULL,
                        VMAXC=NULL,
                        ...)
```

Within the solve_MODELNAME() function, the model builder will call to the unifying solve_model function, which is already in httk and used by all solve_MODELNAME() functions. This portion remains somewhat generic, however specific arguments for the new model must be configured; "model" is set to "gas_closed_pbtk", and relevant arguments are included in parameterize.arg.list. Before this function is called, other necessary specifications can be made, such as checking that the chemical name, CAS number, or DTXSID exist or ensuring that the correct route of exposure is selected.

```
#Now make call to solve_model with gas model specific arguments configured
  out <- solve_model(
    chem.name = chem.name,
    chem.cas = chem.cas,
    dtxsid=dtxsid,
    times=times,
    parameters=parameters,
    model="gas_closed_pbtk",
    route=route,
    dose=dose,
    dosing=list(
      initial.dose=dose,
      dosing.matrix=dosing.matrix,
      daily.dose=daily.dose,
      doses.per.day=doses.per.day
```

```
    ),
  days=days,
  tsteps = tsteps, # tsteps is number of steps per hour
  initial.values=initial.values,
  plots=plots,
  monitor.vars=monitor.vars,
  suppress.messages=suppress.messages,
  species=species,
  input.units=input.units,
  output.units=output.units,
  method=method,rtol=rtol,atol=atol,
  parameterize.arg.list = list(
    default.to.human=default.to.human,
    KM = KM,
    RATS=RATS,
    VCHC=VCHC,
    VMRATIO=VMRATIO,
    KLOSS=KLOSS,
    KRESYN=KRESYN,
    KAS=KAS,
    VMAXC=VMAXC),
  ...)

return(out)
```

Chapter 5

Putting it all together into httk

The first step is to create a project where all the old and new functions can be stored. This can be done in RStudio. In general, you will download httk from the CRAN or GitHub repository, with all the child folders necessary for its creation. The example used here will show you how to create a new project, and the steps leading to adding documentation using roxygen2.

To create a new project use the New Project window towards the top right corner in RStudio:

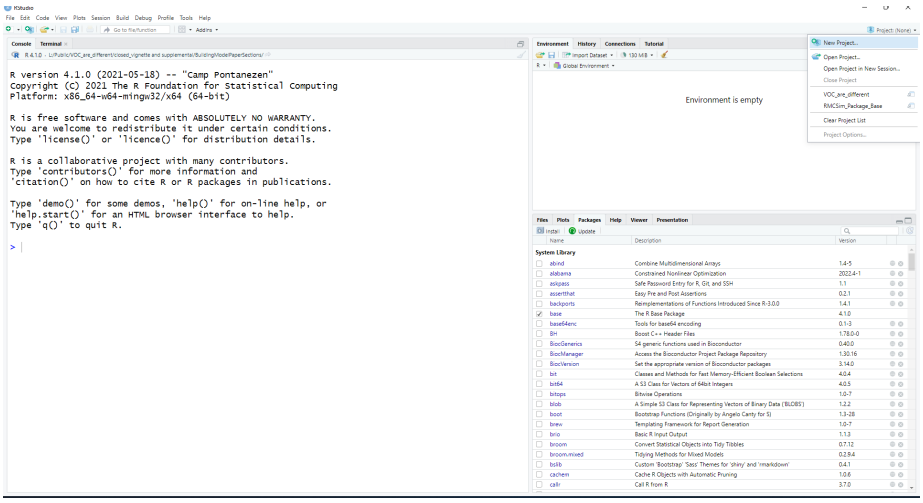


Figure 5.1: Start a new Project

Proceed with the Menu and enter a Path where you want the package to reside.

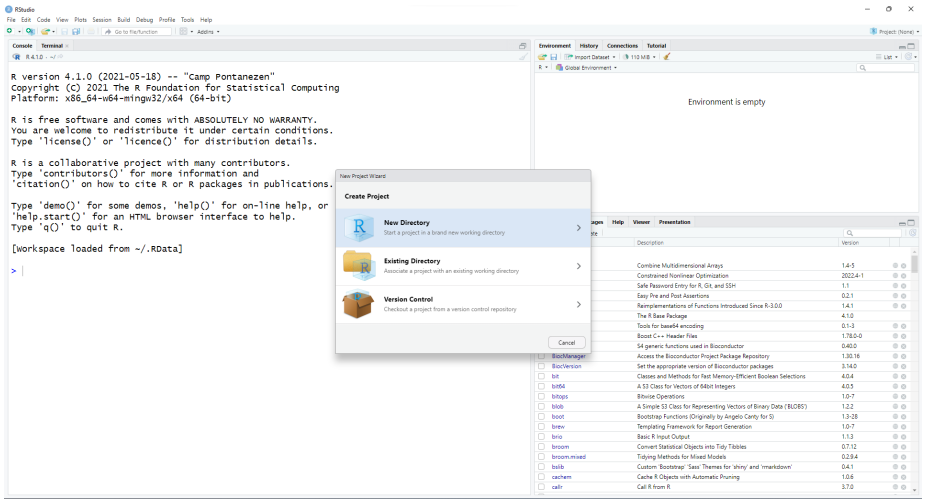


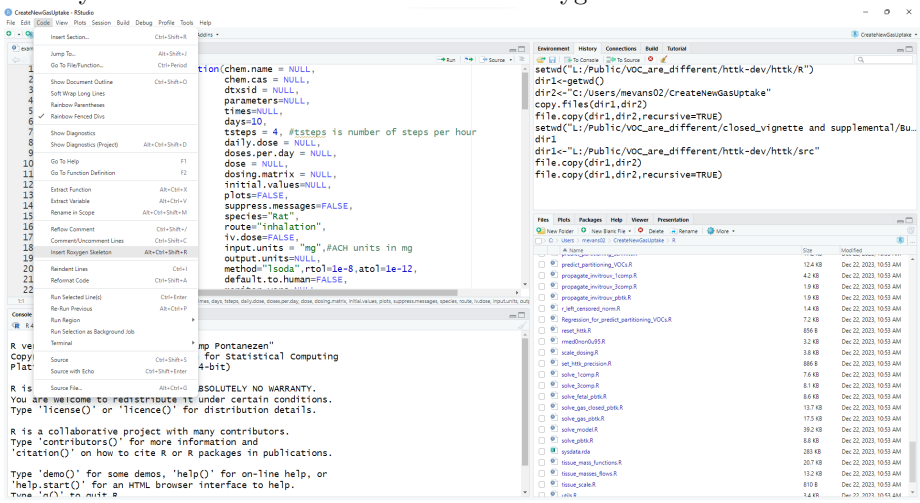
Figure 5.2: Create a New Directory

Scroll down to find the option to open with devtools - a package that has a nice set of functions dedicated to the creation of packages.

One of several files that needs editing is the DESCRPTION file, since the default DESCRIPTION file is empty. Use any editor of your choice to enter the name, authors and related information about the package into this file. We have provided an edited version as an example for the closed inhalation module.

The DESCRIPTION file can be found in the httk directory.

After modification of the DESCRIPTION file you can use roxygen2 to add comments and document your new code. Click on Code>Insert Roxygen as shown in the next slide.



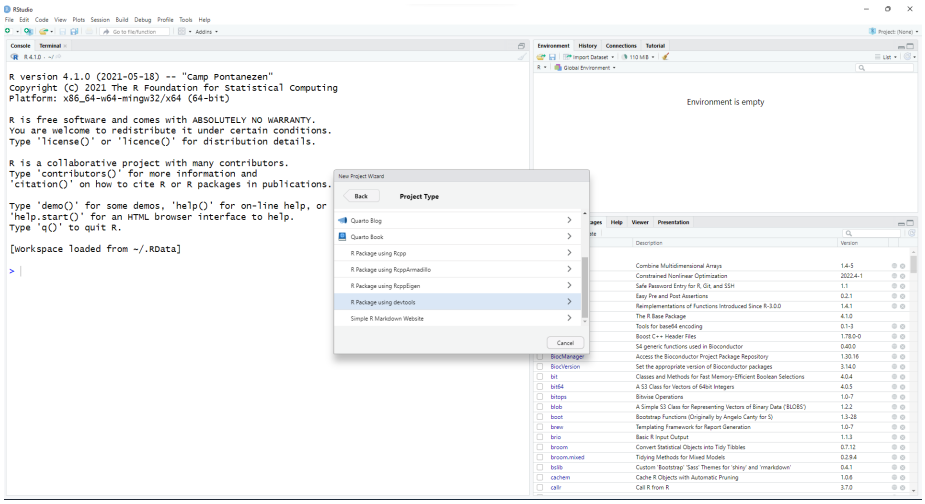


Figure 5.3: Include complete Path

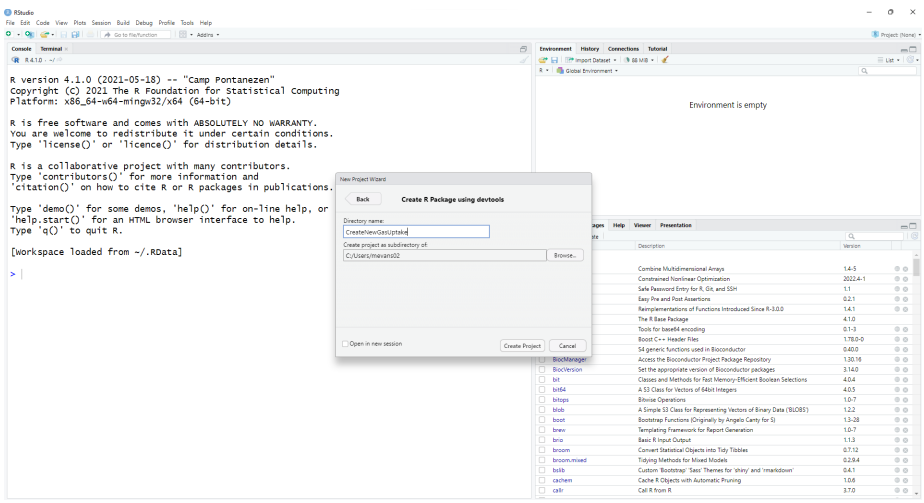


Figure 5.4: Make sure it is not a nested directory

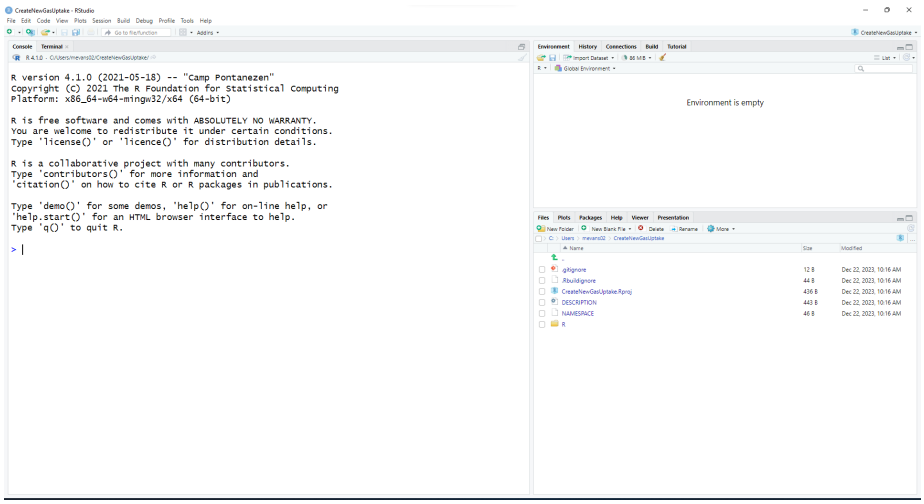
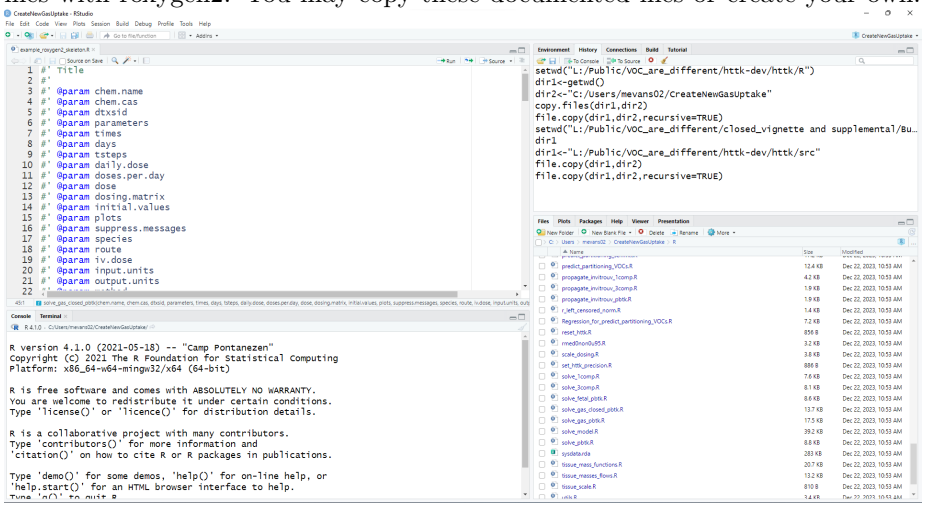


Figure 5.5: The new project and associated files will appear in the new directory

A skeleton text file is created for your new .R file. At this point, you can insert comment describing the new parameters. We have updated all our new function files with roxygen2. You may copy these documented files or create your own.



All new function files generated for the closed gas uptake addition have been edited to include comments using roxygen2. Please go to the CRAN repository to obtain the latest httk version with examples.

The final step is to use the package devtools to build and compile your new function.

```
devtools::load_all("C:/Users/mevans02/CreateNewGasUptake")
i Loading CreateNewGasUptake
Re-compiling CreateNewGasUptake
-  installing *source* package 'CreateNewGasUptake' ... (1.3s)
   ** using staged installation
   ** libs
   /mingw64/bin/gcc -shared -s -static-libgcc -o CreateNewGasUptake.dll tmp.def gas_closed_pbtk.c
   installing to C:/Users/mevans02/AppData/Local/Temp/RtmpABpKb3/devtools_install_54f0ae23f24/00L
-  DONE (CreateNewGasUptake)
```

For httk, you will need to specify the complete path where all the R functions are located. All the related closed inhalation files can be found in the .R folder. In our case, the path is: path<-“L:/Public/VOC_are_different/httk-dev/httk” load_all(path) i Loading httk

The new function will be integrated into httk.

Bibliography

Alan F Sasso, Paul M Schlosser, Gregory L Kedderis, Mary Beth Genter, John E Snawder, Zheng Li, Susan Rieth, and John C Lipscomb. Application of an updated physiologically based pharmacokinetic model for chloroform to evaluate cyp2e1-mediated renal toxicity in rats and mice. *toxicological sciences*, 131(2):360–374, 2013.

Wambaugh, JF,Pearce RG, et al. *High-throughput toxicokinetics (httk) R Package*. US Environmental Protection Agency, RTP, North Carolina, USA, 2017. URL https://github.com/USEPA/CompTox-ExpoCast-httk.