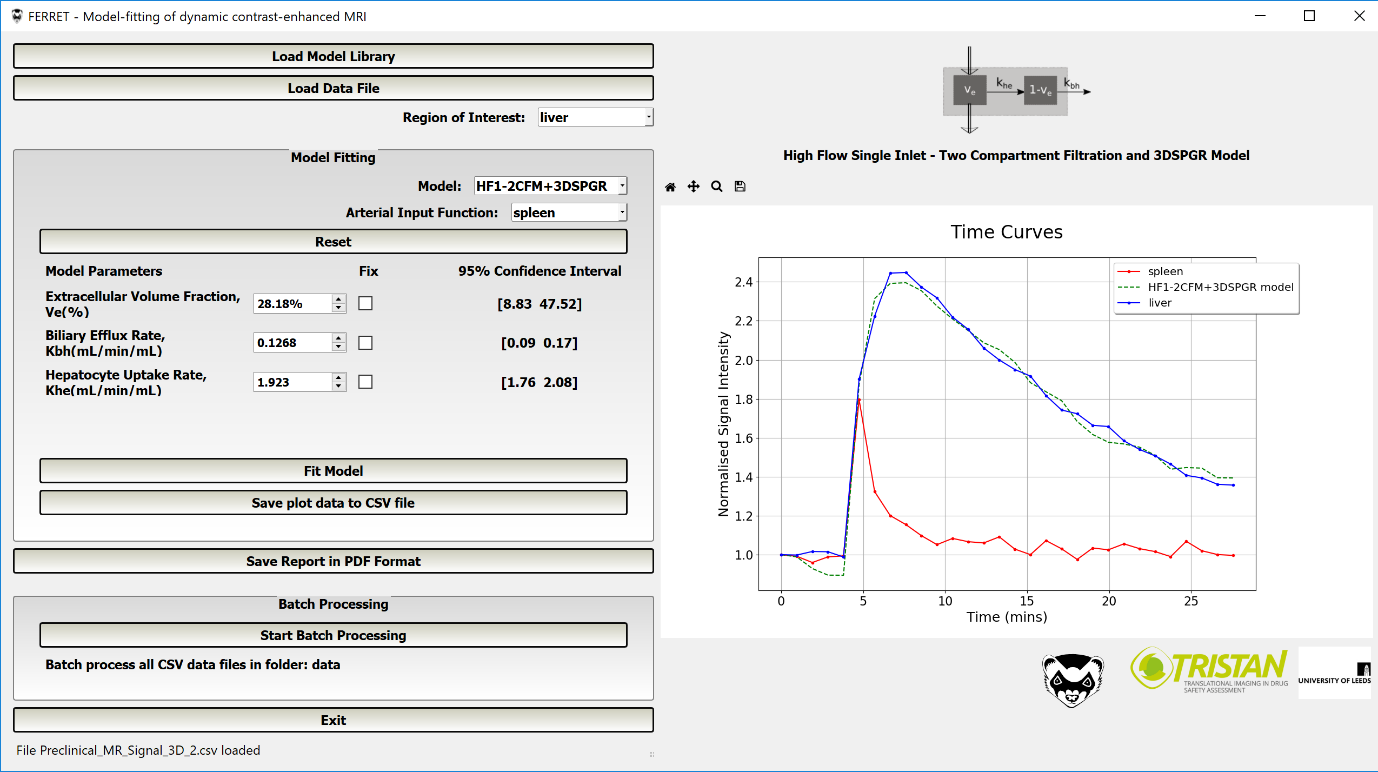
FERRET Developer Guide

# Introduction

The FERRET application allows the user to display an MR signal/time curve for a region of interest and fit a model to that curve. The values of the model parameters that achieve this fit are displayed on the left-hand side of the GUI. It is also possible for the user to manually adjust the values of these model parameters and observe how the shape of the model curve changes.

The diagram below shows a screenshot of the FERRET application with a plot of the MR signal pertaining to the concentration of a tracer compound in the liver (blue solid curve). The High Flow Single Inlet – Two Compartment Filtration and 3DSPGR model has been fitted to this curve (green dotted line). In the middle of the left-hand side of the screen, the three model parameters and their resultant values are displayed. 

Clicking the **Load Model Library** button, allows the user to select and load a model library XML file into the FERRET application. A model library XML file describes a list of models in XML format. The list of models in the **Model:** dropdown list is created from information in the selected model library XML file. When a model is selected from this dropdown list, details of its parameters such as name, units and default value are retrieved from the model library XML file.

Clicking the **Load Data File** button, allows the user to select and load the input CSV file of time and MR signal data for several organs (or regions of interest). The lists of organs in the **Region of Interest**, **Arterial Input Function** and **Venous Input Function** (if displayed) are formed from the column headers in the CSV data file.

The FERRET application ‘ships’ with several liver models but the software has been designed to allow the user to add their own models. All changes are made to files in the subfolders within the **Developer** folder. **This involves three steps**:

1. Write a function in Python that contains the algorithm that executes the model. This function can be written in an existing model library python module containing model functions or a new module can be created for this purpose. Modules contain model functions must reside in the **Developer\ModelLibrary** folder.
2. Describe this function in XML format in an existing model library XML file or a new model library XML file can be created for this purpose. Model library XML files must reside in the **Developer\ModelConfiguration** folder.
3. This step is optional. Create a graphics file depicting a schematic representation of the new model and store it in the **Developer\ModelDiagrams** folder. The following file formats are supported:

| **Format** | **Description** |
| --- | --- |
| BMP | Windows Bitmap |
| GIF | Graphic Interchange Format (optional) |
| JPG | Joint Photographic Experts Group |
| JPEG | Joint Photographic Experts Group |
| PNG | Portable Network Graphics |
| PBM | Portable Bitmap |
| PGM | Portable Graymap |
| PPM | Portable Pixmap |
| XBM | X11 Bitmap |
| XPM | X11 Pixmap |

In the remainder of this document, the requirements for running the FERRET application are listed, an overview of the relevant features of the software structure and model library XML file is given and the above two stage process is explained in greater depth.

# Software Requirements

In addition to the 32 bit version of Python 3, the following Python packages must be installed on your computer (or virtual environment):

numpy

pyautogui

PyQt5

matplotlib

scipy

FPDF

openpyxl

lmfit

importlib

The source code can be downloaded from <https://github.com/IMI-TRISTAN/FERRET>

# Brief overview of the software structure.

This application was written in Python 3 using the Microsoft Visual Studio IDE. The file TRISTAN-Model-Fitting.pyproj is a Visual Studio project file and may not be required in your IDE.

In the working directory of the application:

1. The Python module **FERRET.py** is the start-up file.
2. In the folder **Developer\ModelLibrary**, the Python module **ModelFunctions.py** contains functions that run models. Each model has its own function. **A model can have up to a maximum of 5 parameters.**
3. The folder **CoreModules** contains Python modules that supply the core functionality of FERRET. It contains the following modules:
   1. **ExcelWriter.py**
   2. **ExceptionHandling.py**
   3. **ModelFunctionsHelper.py**
   4. **ModelFunctionsHelper.py**
   5. **PDFWriter.py**
   6. **StyleSheet.py**
   7. **MathsTools.py**
   8. **XMLReader.py**
4. The Python module **MathsTools.py** contains mathematical routines used by the model functions. If your model(s) require additional mathematical routines, you could either,
5. Add them to Tools.py
6. Or create another module and add them to this module. This module would have to be imported into the module where the model function(s) using these mathematical routines are coded; e.g.,

import MyMathsTools as mymathstools

1. In the folder **Developer\ModelLibrary**, the Python module **Model\_Function\_Module\_Template.py** contains a template for a model function to which model logic can be added. This module can be renamed with a more meaningful name.
2. The folder, **Developer\ModelDiagrams**, contains image files that show a schematic representation of a given model. When a model is selected, if there is a corresponding image in the **Developer\ModelDiagrams** folder it is displayed on the GUI. Otherwise the string ‘**No image available for this model**’ is displayed on the GUI. Below this image the full name of the model is displayed.

# Brief overview of the Model Library XML file structure.

1. The root node of a Model Library XML file is <models></models>.
2. The section of the XML file enclosed by the XML tags <constants></constants> contains a collection of <constant></constant> nodes that describe operational constants relating to the MRI scanner on which the signal data where obtained. These constants are used to convert MR signals to concentrations and calculated concentrations back to MR signals. It may be necessary to adjust the values of these constants or add/remove constants depending on the MR scanner used to obtain the signal data. These constants are passed into a model function via the constantsString parameter as a string representation of a Python dictionary; e.g., in the case of the data belowthis would be **“{'TR': '0.013', 'baseline': '1', 'FA': '20', 'r1': '5.5', 'R10a': '0.74575', 'R10t': '1.3203'}”**. If you are going to test your own model function in a test stub, then it could be called with,

constantsString = “{'TR': '0.013', 'baseline': '1', 'FA': '20', 'r1': '5.5', 'R10a': '0.74575', 'R10t': '1.3203'}”

<constants>

<constant>

<!--Repetition time of dynamic SPGR sequence in seconds -->

<name>TR</name>

<value>0.013</value>

</constant>

<constant>

<!-- Number of baseline scans -->

<name>baseline</name>

<value>5</value>

</constant>

<constant>

<!-- Degrees-->

<name>FA</name>

<value>20</value>

</constant>

<constant>

<!-- Hz/mM-->

<name>r1</name>

<value>5.5</value>

</constant>

<constant>

<!-- Hz-->

<name>R10a</name>

<value>0.74575</value>

</constant>

<constant>

<!-- Hz-->

<name>R10t</name>

<value>1.3203</value>

</constant>

</constants>

1. A model must have a long name, which is a full description of the model and a short name, which is a code-like abbreviation. The short name is displayed in a drop-down list of model names on the application GUI.
2. The description of each model is enclosed by the <model></model> tags.
3. Each model must have a long and a short name and they are enclosed by the following XML tags.

<model>

<name>

<short>***short name***</short>

<long>***long name***</long>

</name>

*Other tags removed for brevity*

</model>

When the model library XML file is loaded, the short name for each model is compiled into a list and displayed in the **Model:** dropdown list.

1. The folder **Developer\ModelConfiguration** contains the model library XML files. Related models can be grouped together in the same model library XML file. For example, several liver models could be described in the same XML file, with the name LiverModels.xml. Clicking the **Load Model Library** button shows the contents of the **config** directory in an open file dialog. Selecting a particular model library XML file loads the models it describes into the application.
2. The folder **Developer\ModelConfiguration** contains an empty template model library XML file called **Model\_Library\_Template.xml** to which one or more model descriptions can be added. This XML file can be renamed with a more meaningful name.

# Adding a model to the FERRET model fitting application.

Incorporating a new model into the FERRET application will be demonstrated by describing the steps required to add a single inlet liver model, with the long name **High Flow Single Inlet - Two Compartment Filtration and 3DSPGR Model** and short name **HF1-2CFM+3DSPGR** to this application**.**

## 1. Coding the model.

1. In **Developer\ModelLibrary,** rename the Python module **Model\_Function\_Module\_Template.py** to **LiverModels.py**.
2. Open **LiverModels.py** and scroll down to the module function template definition

def Model\_Function\_Template(xData2DArray, param1, param2, param3, param4, param5, constantsString):

Rename the function to **HighFlowSingleInletGadoxetate3DSPGR**

def **HighFlowSingleInletGadoxetate3DSPGR**(xData2DArray, param1, param2, param3, param4, param5, constantsString):

1. In **LiverModels.py**, in the function **HighFlowSingleInletGadoxetate3DSPGR**, replace the following comment block with the logic of your model.

###########################################

#

# Add code here to calculate concentration, ct

#

##############################################

In the case of the **High Flow Single Inlet - Two Compartment Filtration and 3DSPGR Model,** this logic is

# Correct for spleen Ve

ve\_spleen = 0.43

ce = ca/ve\_spleen

Th = (1-Ve)/Kbh

ct = Ve\*ce + Khe\*Th\* /

tools.expconv(Th,t,ce,'HighFlowSingleInletGadoxetate3DSPGR')

In the above code, the model parameters are:

* Kbh - Biliary Efflux Rate, units mL/min/mL.
* Khe - Hepatocyte Uptake Rate, units mL/min/mL .
* Ve - Extracellular Volume Fraction, units %.

1. Add the above model parameters to the model function definition and delete the parameter placeholders ‘param4’ and ‘param5’. The model function definition should now look like,

def HighFlowSingleInletGadoxetate3DSPGR(xData2DArray, **Ve, Kbh, Khe**,

constantsString):

1. The complete model function will look like (added code is highlighted in bold red),

def HighFlowSingleInletGadoxetate3DSPGR\_Rat(xData2DArray, **Ve, Kbh, Khe**,

constantsString):

"""This function contains the algorithm for calculating

how the MR signal from a 3D scan varies with time using the

High Flow Single Inlet Two Compartment Gadoxetate Model model.

Input Parameters

----------------

xData2DArray - time and AIF concentration 1D arrays

stacked into one 2D array.

Ve - Plasma Volume Fraction (decimal fraction).

Khe - Hepatocyte Uptake Rate (mL/min/mL)

Kbh - Biliary Efflux Rate (mL/min/mL)

constantsString - String representation of a dictionary

of constant name:value pairs used to convert concentrations

predicted by this model to MR signal values.

Returns

-------

St\_rel - list of calculated MR signals at each of the

time points in array 'time'.

"""

try:

exceptionHandler.modelFunctionInfoLogger()

t = xData2DArray[:,0]

Sa = xData2DArray[:,1]

# Unpack SPGR model constants from

# a string representation of a dictionary

# of constants and their values

constantsDict = eval(constantsString)

TR, baseline, FA, r1, R10a, R10t = \

float(constantsDict['TR']), \

int(constantsDict['baseline']),\

float(constantsDict['FA']), float(constantsDict['r1']), \

float(constantsDict['R10a']), float(constantsDict['R10t'])

# Convert to concentrations

# n\_jobs set to 1 to turn off parallel processing

# because parallel processing caused a segmentation

# fault in the compiled version of this application.

# This is not a problem in the uncompiled script

R1a = [Parallel(n\_jobs=1)(delayed(fsolve)

(tools.spgr3d\_func, x0=0,

args = (FA, TR, R10a, baseline, Sa[p]))

for p in np.arange(0,len(t)))]

R1a = np.squeeze(R1a)

ca = (R1a - R10a)/r1

**# Correct for spleen Ve**

**ve\_spleen = 0.43**

**ce = ca/ve\_spleen**

**Th = (1-Ve)/Kbh**

**ct = Ve\*ce + Khe\*Th\*tools.expconv(Th,t,ce,'HighFlowSingleInletGadoxetate3DSPGR\_Rat')**

# Convert to signal

St\_rel = tools.spgr3d\_func\_inv(r1, FA, TR, R10t, ct)

return(St\_rel) #Returns tissue signal relative to the baseline St/St\_baseline

except ZeroDivisionError as zde:

exceptionHandler.handleDivByZeroException(zde)

except Exception as e:

exceptionHandler.handleGeneralException(e)

## 2. Describing the model in the model library XML file.

1. In the **Developer\ModelConfiguration** folder, rename **Model\_Library\_Template.xml** to **LiverModels.xml**.
2. Open LiverModels.xml, scroll down to the section enclosed by the <model></model> tags and make the following changes:
   1. The <name></name> tags encapsulate the short and long names of the model. Between the <short></short> tags insert the short model name and between the <long></long> tags insert the long model name.
   2. Between the <module></module> tags insert the name of the module containing the coded liver model, **LiverModels,** without the .py extension.
   3. Between the <inlet\_type></inlet\_type> tags insert the word **single.** If the model were dual inlet, the word **dual** would be inserted here.

<model>

<name>

<short>**HF1-2CFM+3DSPGR**</short>

<long>**High Flow Single Inlet - Two Compartment Filtration and 3DSPGR Model**</long>

</name>

<!--Name of the module containing the function-->

<module>**LiverModels**</module>

<!--Name of the function implementing this model-->

<function></function>

<!--Name of the image describing schematically this model-->

<image></image>

<inlet\_type>**single**</inlet\_type>

*Other tags removed for brevity*

</model>

1. Between the <function></function> tags insert the name of model function, **HighFlowSingleInletGadoxetate3DSPGR**.

<model>

<name>

<short>**HF1-2CFM+3DSPGR**</short>

<long>**High Flow Single Inlet-Two Compartment Filtration and 3DSPGR Model**</long>

</name>

<!--Name of the module containing the function-->

<module>**LiverModels**</module>

<!--Name of the function implementing this model-->

<function>**HighFlowSingleInletGadoxetate3DSPGR**</function>

<!--Name of the image describing schematically this model-->

<image></image>

<inlet\_type>**single**</inlet\_type>

*Other tags removed for brevity*

</model>

1. In the images folder, **HighFlowSingleInletTwoCompartmentGadoxetateModel.png** is a schematic representation of the High Flow Single Inlet - Two Compartment Filtration and 3DSPGR Model. Between the <image></image> tags insert the name of this image file with its .png file extension. If an image file with a schematic representation of your model is not available leave the <image></image> tags unchanged.

<model>

<name>

<short>**HF1-2CFM+3DSPGR**</short>a

<long>**High Flow Single Inlet-Two Compartment Filtration and 3DSPGR Model**</long>

</name>

<!--Name of the module containing the function-->

<module>**LiverModels**</module>

<!--Name of the function implementing this model-->

<function>**HighFlowSingleInletGadoxetate3DSPGR**</function>

<!--Name of the image describing schematically this model-->

<image>**HighFlowSingleInletTwoCompartmentGadoxetateModel.png**</image>

<inlet\_type>**single**</inlet\_type>

*Other tags removed for brevity*

1. Between the <model></model> tags, scroll down to the section enclosed by the <parameters></parameters> tags. Nested within these tags, the <parameter></parameter> tags enclose further tags that describe how each model parameter is displayed in the GUI. Each parameter must be setup in the XML file in the same order in which they are listed in their model function definition. Therefore, we will setup Ve first, then Kbh and finally Khe.
2. Firstly define the short and long names of Ve; thus,

<parameters>

<parameter>

<name>

<short>Ve</short>

<long>Extracellular Volume Fraction</long>

</name>

1. Extracellular volume fraction has units % and has a default value of 23.0

<units>%</units>

<default>23.0</default>

1. On the GUI, model parameters are displayed in a spinbox and their values can be incremented/decremented by up/down arrows by an amount specified between the <step></step> tags. The number of decimal places displayed in the spinbox is defined between the <precision></precision> tags. For extracellular volume fraction, define a step of 1.0 and display to 2 decimal places.

<step>1.0</step>

<precision>2</precision>

1. Between the <display\_value></display\_value> tags it is possible to set a lower and an upper limit to the value of the model parameter displayed in the spinbox. For extracellular volume fraction, define the minimum value as 0.01 and the maximum value as 99.99; thus,

<display\_value>

<min>0.01</min>

<max>99.99</max>

</display\_value>

1. When fitting the concentration/time curve generated by the model to the concentration/time curve in the region of interest, it is possible to constrain the value of the model parameter between a lower and an upper value. In the case of extracellular volume fraction, there are no constraints. So leave the following XML fragment unchanged.

<constraints>

<lower></lower>

<upper></upper>

</constraints>

1. This section of the XML file should now look like this,

<parameters>

<parameter>

<name>

<short>Ve</short>

<long>Extracellular Volume Fraction</long>

</name>

<units>%</units>

<default>23.0</default>

<!--When parameter spinbox arrows are clicked,

the value of the parameter is incremented/decremented by

the value of step-->

<step>1.0</step>

<!--precision is the number of decimal places displayed

in the parameter spinbox-->

<precision>2</precision>

<!--Lower and upper display value in the spinbox -->

<display\_value>

<min>0.01</min>

<max>99.99</max>

</display\_value>

<!--Constraints applied during model fitting-->

<constraints>

<lower></lower>

<upper></upper>

</constraints>

</parameter>

</parameters>

1. Likewise, set up the remaining two model parameters with the following data within their own set of <parameter></parameter> tags within the <parameters></parameters> tags.

|  |  |
| --- | --- |
| Short name | Kbh |
| Long name | Biliary Efflux Rate |
| Units | mL/min/mL |
| Default value | 0.0918 |
| Step | 0.01 |
| Precision | 4 |
| Minimum display value | 0.01 |
| Maximum display value | 100.0 |
| Lower constraint | None |
| Upper constraint | None |

|  |  |
| --- | --- |
| Short name | Khe |
| Long name | Hepatocyte Uptake Rate |
| Units | mL/min/mL |
| Default value | 2.358 |
| Step | 0.1 |
| Precision | 3 |
| Minimum display value | 0.0001 |
| Maximum display value | 100.0 |
| Lower constraint | None |
| Upper constraint | None |

1. The model **High Flow Single Inlet - Two Compartment Filtration and 3DSPGR Model**  has now been added to FERRET and complete model definition should look like this

<model>

<name>

<short>**HF1-2CFM+3DSPGR**</short>

<long>**High Flow Single Inlet - Two Compartment Filtration and 3DSPGR Model**</long>

</name>

<module>**LiverModels**</module>

<function>**HighFlowSingleInletGadoxetate3DSPGR**</function>

<image>**HighFlowSingleInletTwoCompartmentGadoxetateModel.png**</image>

<inlet\_type>**single**</inlet\_type>

<parameters>

<parameter>

<name>

<short>**Ve**</short>

<long>**Extracellular Volume Fraction**</long>

</name>

<units>**%**</units>

<default>**23.0**</default>

<!--When parameter spinbox arrows are clicked,

the value of the parameter is incremented/decremented by

the value of step-->

<step>**1.0**</step>

<!--precision is the number of decimal places displayed

in the parameter spinbox-->

<precision>**2**</precision>

<!--Lower and upper display value in the spinbox -->

<display\_value>

<min>**0.01**</min>

<max>**99.99**</max>

</display\_value>

<!--Constraints applied during model fitting-->

<constraints>

<lower></lower>

<upper></upper>

</constraints>

</parameter>

<parameter>

<name>

<short>**Kbh**</short>

<long>**Biliary Efflux Rate**</long>

</name>

<units>**mL/min/mL**</units>

<default>**0.0918**</default>

<step>**0.01**</step>

<precision>**4**</precision>

<display\_value>

<min>**0.01**</min>

<max>**100.0**</max>

</display\_value>

<constraints>

<lower></lower>

<upper></upper>

</constraints>

</parameter>

<parameter>

<name>

<short>**Khe**</short>

<long>**Hepatocyte Uptake Rate**</long>

</name>

<units>**mL/min/mL**</units>

<default>**2.358**</default>

<!--When parameter spinbox arrows are clicked,

the value of the parameter is incremented/decremented by

the value of step-->

<step>**0.1**</step>

<!--precision is the number of decimal places displayed

in the parameter spinbox-->

<precision>**3**</precision>

<display\_value>

<min>**0.0001**</min>

<max>**100.0**</max>

</display\_value>

<constraints>

<lower></lower>

<upper></upper>

</constraints>

</parameter>

</parameters>

</model>

# Creating an executable of FERRET.

Having added a new model(s) to FERRET, you may wish to create an executable for easy distribution to users who do not have Python installed on their computer. We recommend the open source application **auto-py-to-exe.exe** to do this. It has an easy to use GUI and can be downloaded from <https://pypi.org/project/auto-py-to-exe/>