



Eötvös Loránd University
Faculty of Informatics
Department of Programming Languages and
Compilers

Analysis Software for FACS-measurements

Supervisor: Tamás Kozsik
Associate Professor

Ambrus Kaposi
qualification:
Programmer Informatician BSc

Budapest, 2012

Contents

1	Introduction	4
2	The Task	6
2.1	Specification	6
2.1.1	Reading and selecting measurement data	6
2.1.1.1	Input: FCS file	6
2.1.1.2	Output: gated file	7
2.1.1.3	Steps from FCS to gated	8
2.1.2	Fitting functions	9
2.1.2.1	Input: gated file	10
2.1.2.2	Output: kinetics file	10
2.1.2.3	Steps from gated to kinetics	11
2.1.3	Comparing measurements	13
2.2	Analysis of Requirements	14
2.2.1	Communication with other programs	14
2.2.2	Runtime Environment	15
2.2.3	Requirements for goodness of the program	15
2.2.3.1	Speed	15
2.2.3.2	Accessibility	15
2.2.3.3	Maintanance and failure handling	16
2.2.3.4	Security	16
3	User Documentation	17
3.1	Program Structure	17
3.2	FacsKin step-by-step User's Guide	19
3.2.1	Acquiring measurement data	19
3.2.2	Starting FacsKin	20
3.2.2.1	Launch with Java Web Start	21
3.2.2.2	Launch from the ZIP bundle	21
3.2.3	Opening and gating FCS files	23
3.2.4	Uploading gated data	26

3.2.5	Receiving .kinetics file as email attachment	27
3.2.6	Opening different .kinetics files and selecting a common function	28
3.2.6.1	Detailed description of the functions	29
3.2.7	Creating groups and comparing parameters of the selected function in different groups	35
3.2.7.1	Exporting comparison data	38
3.2.7.2	Pairing measurements	39
3.3	Maintenance of FacsKin	41
3.4	Caflux User's Guide	42
3.4.1	Installation	42
3.4.2	Maintenance	44
3.4.3	Uninstallation	45
4	Developer Documentation	48
4.1	Structure	48
4.2	FacsKin	50
4.2.1	System Plan	50
4.2.1.1	User Interface	51
4.2.1.2	Table models	56
4.2.1.3	Plots	58
4.2.1.4	Data Containers	60
4.2.1.5	Input/Output	61
4.2.1.6	Math classes	64
4.2.1.7	Error handling	66
4.2.2	Implementation	68
4.2.2.1	Version History	68
4.2.2.2	Further Development	69
4.2.2.3	Interesting Algorithms	70
4.2.2.4	Compilation and Distribution	72
4.2.3	Testing	74
4.2.3.1	Tests using the GUI	74
4.2.3.2	Unit tests	80
4.2.3.3	Test results	81
4.3	Caflux	82
4.3.1	System Plan	82
4.3.2	Implementation	82
4.3.2.1	Further Development	86
4.3.3	Testing	86
4.3.3.1	Unit Tests for fct-64.R	86

4.3.3.2	Common tests for FacsKin + Caflux	87
4.3.3.3	Performance of Caflux	88
Bibliography		88
Journal articles and books		90
Websites		91

Chapter 1

Introduction

Flow Cytometry [17] is a method for investigating the properties of microscopic particles in a stream of liquid by exciting them with light and measuring the emission and reflection properties individually for each particle. This technique can be used for the sequential analysis of millions of particles during a 10-minute timeframe. It is widely used in medicine both for diagnostics and research. A typical usage is assessing the prevalence of different cell subsets in a sample containing biological cells by staining the cells with subset-specific fluorescent dyes. The result of such a measurement is a couple of parameters for each cell:

- *Time*: the exact time point when the cell was measured.
- *FSC, SSC*: Forward Scatter and Side Scatter, these values are measured by two light detectors and roughly correspond to the size and granulation of the cell.
- *Fluorescent parameters*: the other parameters are calculated by the Flow Cytometer by integrating the emission-spectrum of the cell at different wavelength-ranges (each wavelength-range corresponding to the emission spectrum of a particular fluorescent dye the name of which is used as the name of the parameter)¹.

An example of a measurement result is shown in Table 1.1.

The distribution of parameters can be visualized using histograms or scatter plots (2 dimensional density plots), examples are shown in Figure 1.1 and 1.2.

Kinetic Flow Cytometry measurements are defined by having a parameter the distribution of which varies over time. Special statistical methods were developed for the description and statistical comparison for such measurements ([4], [6], [5]). My aim was to develop a software that implements the method as described in [5].

¹The overlap of spectra of different fluorescent dyes are compensated by the Flow Cytometer using specific algorithms [12].

FSC	SSC	FITC	PE	PE-Cy7	APC	APC-Cy7	Time
54528.8	64740.1	6496.4	1727.5	681.4	429.5	717.5	0.001
81142.5	44227.6	5153.4	488.8	2795.1	8.6	522.7	0.002
86443.5	45844.4	19496.7	305.3	977.3	68.4	93.2	0.003
63353.3	125472.2	8034.5	781.2	282.8	1453.7	627.3	0.003
35268.1	122954.8	5138.4	608.0	436.5	890.5	537.8	0.005

Table 1.1: An example of a Flow Cytometry measurement result. One row corresponds to one cell, and each column corresponds to a parameter measured by the Flow Cytometer. Time is shown as seconds, the other parameters are measured on a relative scale that is proportional to the amount of fluorescent dye.

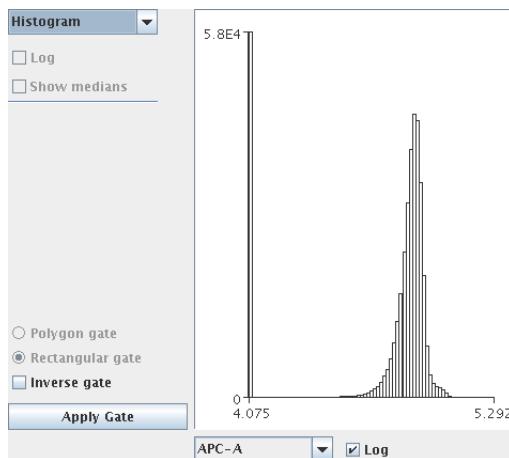


Figure 1.1: An example of the histogram of parameter APC (given as APC-A in the bottom) in a Flow Cytometry measurement. In this measurement white blood cells were dyed with a CD4-antibody binded to fluorescent dye APC.

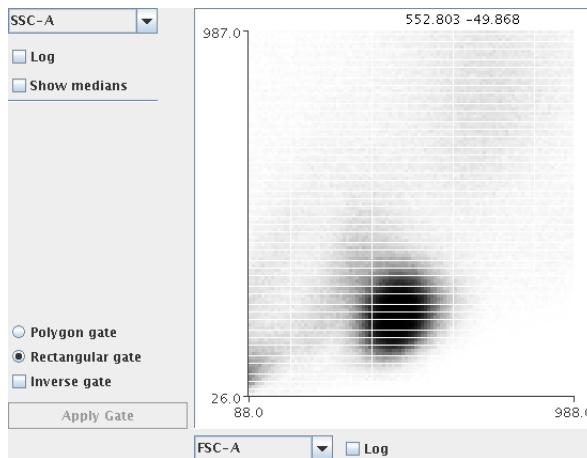


Figure 1.2: An example of the scatter plot of parameters FSC and SSC in a Flow Cytometry measurement. The darker the plot the more cells in that range are.

The software was developed in collaboration with the FACS Laboratory at Department of Laboratory Medicine, Semmelweis University [22]. Potential users of this software are researchers using a Flow Cytometer device (or FACS, Fluorescent Activated Cell Separator, which is a Flow Cytometer with additional features).

Chapter 2

The Task

The task is to create a program for the analysis of Kinetic Flow Cytometry measurements using the algorithm described in [5] as method “Fitting to quantiles”.

The inputs of the program are FCS files produced by a Flow Cytometer and user choices based on intermediate analysis results. The outputs are various figures, tables describing and comparing different characteristics of kinetic processes taking place in the measurements.

2.1 Specification

The program should implement the following three phases:

- Reading and selecting measurement data (Section 2.1.1)
- Fitting functions (Section 2.1.2)
- Comparing measurements (Section 2.1.3)

2.1.1 Reading and selecting measurement data

2.1.1.1 Input: FCS file

The input of this phase is one or more FCS files, either FCS 2.0 [8] or FCS 3.0 [13] format (extension `fcs`).

FCS is a binary file-format which contains a HEADER segment, a TEXT segment, a DATA segment and optionally other segments.

The header contains the ASCII-encoded string "FCS2.0" or "FCS3.0" (6 bytes), 4 bytes of space characters (ASCII 32), then the ASCII-encoded offsets to the first and last bytes of the next segment, then the offsets to the first and last bytes of the next segment and so on.

The TEXT segment contains keyword-value pairs in ASCII format. The first character of the TEXT segment is the delimiter character that separates keywords from values. Then follow the first keyword, a delimiter, the value that belongs to the first keyword, a delimiter, the second keyword and so on. Some of the keywords are compulsory such as **\$PAR** (number of parameters), **\$MODE** (list or histogram mode, list mode should be supported), **\$DATATYPE** (how the data is stored: float, double or integer), **\$BYTEORD** (byte order). The names of each parameter is also given in the text segment (**\$PjN** keywords, where j ranges from 1 to **\$PAR**). The **\$PjE** keyword specifies the logarithmic amplification in case the j^{th} parameter is stored logarithmized (it is a pair separated by comma, we refer only to the first element of the pair here as PjE). The **\$PjR** keyword specifies the theoretically highest value of the j^{th} parameter. Some Flow Cytometers output the compensation matrix and the raw parameter data instead of only compensated parameter data by specifying **APPLY COMPENSATION** (having value **TRUE**) and **SPILL** keywords (the value of which is an $n \times n$ matrix by rows, elements being separated by comma).

The DATA segment contains the parameters of the first *event* in the format specified in the TEXT segment, then the parameters of the second event etc. This is essentially a table with events as rows and parameters as columns.

We refer to this table as the following set: $data \in D^{npar}$, where D is either float, double or integer according to **\$DATATYPE** and $npar$ is the value stored in **\$PAR**. If for some j , the first element of **\$PjE** is not 0, $data$ should be replaced by

$$\left\{ \square_{j \in [1..npar]} f(d, j) \mid d \in data \right\} \text{ where } f(d, j) = \begin{cases} 10^{d_j * PjE / (PjR - 1)} & \text{if } PjE \neq 0 \\ d_j & \text{if } PjE = 0 \end{cases}$$

If **APPLY COMPENSATION** keyword has value **TRUE**, $data$ should be replaced by

$$\left\{ \square_{j \in [1..npar]} (SPILL^{-1} * d) \mid d \in data \right\}$$

(d is a column vector ($npar \times 1$ -size matrix) and $*$ means matrix multiplication, $SPILL^{-1}$ is the inverse of the $SPILL$ matrix.)

The program should support FCS 2.0 and FCS 3.0 files outputted by the FACS-Diva Version 4.1 [15] program.

2.1.1.2 Output: gated file

The output of this phase is a gated file (extension **gated**).

A gated file is a text file containing the following lines (newline characters and backslash characters are escaped by backslash):

1. the number of lines after this line in the file

2. the number of lines in the metadata section (6)
3. measurement date and time
4. measurement name
5. filename of the original FCS
6. email address of the researcher
7. textual description of gates
8. name of the kinetic parameter
9. number of events
10. time and kinetic parameter values of the first event separated by space (using dot as decimal symbol)
11. time and kinetic parameter values of the second event separated by space (using dot as decimal symbol)
- 12.- etc.

The program should handle the possible later growth of the metadata section.

2.1.1.3 Steps from FCS to gated

The program should allow the following:

1. Read FCS data files. Terminology: an FCS file contains a table of *events* each event having the same *parameters*. We refer to the FCS data as $data \in D^{npar}$, where $npar$ is the number of parameters and D is the type of numbers that are stored in the FCS (integer, double or float).
2. Concatenation of event data contained in FCS files using a gap of arbitrary length (given in seconds) in between by increasing the time parameter of each event in the second file by the time parameter of the last event in the first file plus the gap:

$$data_{new} = data_1 \cup \left\{ \square_{j \in [1..npar]} f(d, j) \mid d \in data_2 \right\}$$

where $f(d, j) = \begin{cases} d_j & \text{if } j \text{ is not the time parameter} \\ \max\{e_j \mid e \in data_1\} + gap + d_j & \text{if } j \text{ is the time parameter} \end{cases}$

3. View the distribution of all parameters using either scatter plot (2-dimensional density plot, see Figure 1.2) of two arbitrarily selected parameters or histogram of one parameter (see Figure 1.1). The user should be able to choose between linear and logarithmic scale for arbitrary parameter.
4. Allow the selection of events (terminology: *gating*) graphically by using the following methods:
 - Polygon gate using scatter plot of parameters x, y :
replace $data$ with $\{d \mid d \in data \wedge d \text{ inside}_{x,y} poly\}$ where $poly \in (D^2)^+$
 - Rectangular gate using scatter plot of parameters x, y :
replace $data$ with $\{d \mid d \in data \wedge d \text{ inside}_{x,y} rect\}$ where $rect \in (D^2)^4$
 - Range gate using histogram plot of parameter x :
replace $data$ with $\{d \mid d \in data \wedge rect_1 \leq d_x \leq rect_2\}$ where $rect \in D^2$
 - Inverse gate using either of the previous three:
replace $data$ with $data \setminus (\text{result of one of the previous three})$

$d \text{ inside}_{x,y} poly$ means that the x and y parameters of d are inside the polygon specified by $poly$. The program should allow arbitrary number of sequential steps of gates.

5. Allow the selection of a parameter to be analyzed either as a single parameter or as the ratio of two parameters. We call this parameter *kinetic parameter*. The last step of producing the output of this phase can be formalized as follows (t is the number of the time parameter and k (or $knum, kdenom$ in case of ratio) is the number of the kinetic parameter, $t, k \in [1..npar]$).

$$gated = \begin{cases} \{(d_t, d_k) \mid d \in data\} & \text{in case of a single kinetic parameter} \\ \{(d_t, d_{knum}/d_{kdenom}) \mid d \in data\} & \text{in case of ratio} \end{cases}$$

6. Output a gated file as the result of the previous step.

2.1.2 Fitting functions

After some transformation of the gated data the program should fit 5 different functions to 201 quantiles of the data in this phase. The functions have 1, 4, 4, 8, 8 *parameters*, respectively (note that this is a different usage of the word *parameter* as in *kinetic parameter*). Hence, the result of this phase is $201 * (1 + 4 + 4 + 8 + 8)$ parameter values besides some metadata.

2.1.2.1 Input: gated file

The input of this phase is a gated file specified in Section 2.1.1.2.

2.1.2.2 Output: kinetics file

The output of this phase is a kinetics file (extension `kinetics`).

A kinetics file is a text file containing the following lines (newline characters and backslash characters are escaped by backslash).

Header segment:

1. the number of lines after this line in the file
2. the number of lines in the metadata section (nm , at least 6, this line does not count)
3. measurement name
4. filename of the original FCS
5. email address of the researcher
6. textual description of gates
7. name of the kinetic parameter
8. number of events
9. whether the baseline part was fixed during the fit (not compulsory)
... possibly additional metadata

Functions segment:

- ($nm+3$). number of lines in the Functions segment of the file (nf , this line does not count)
- ($nm+4$). number of lines in the metadata section of the first function (at least 3)
- ($nm+5$). name of the first function
- ($nm+6$). CV (Cross Validation) value of the first function
- ($nm+7$). SAD (Sum of Absolute Deviations) value of the first function
- ($nm+8$). number of quantiles (201)
- ($nm+9$). parameters for the first quantile of the first function separated by space

(nm+10). parameters for the second quantile of the first function separated by space

(nm+...). ...

(nm+210). parameters for the 201th quantile of the first function separated by space

... function data for the second, third etc. functions in the same format

Medians segment of the file:

(nm+nf+4). number of lines in the Medians segment (*nd*, this line does not count)

(nm+nf+5). time value and median value at the first time interval separated by space

(nm+nf+6). time value and median value at the second time interval separated by space

(nm+nf+...). ...

(nm+nf+4+nd). time value and median value at the last time interval separated by space

Dot should be used as decimal separator symbol. The definition of time intervals and medians can be found below.

The program should handle the possible later growth of the metadata section (both for the whole file and for each function) and also the possible modification of the number of quantiles or functions.

2.1.2.3 Steps from gated to kinetics

The program should implement the following:

1. Read the gated file which (besides some metadata) contains (time, kinetic parameter value) pairs. We will refer to kinetic parameter value as *kinetic value* to avoid confusion with *parameters* of functions.
2. Divide the timeframe into 100 intervals of equal length and calculate the quantiles $1/402 + i/201$, $i \in \{0, 1, 2, \dots, 200\}$ in each time interval thus creating a matrix $qdata \in D^{201 \times ntime}$ containing the quantile values and a vector $times \in D^{n time}$ containing the beginning of each time interval. If there are intervals with no events inside, *ntime* can be less than 100.
3. We define the following functions:

(a)

$$\text{constant}(t; y) = y$$

Parameter constraint: $y \geq 0$

(b)

$$\text{logist}_+(t; y0, y2, x1, m1) = \frac{y0 + (y2 - y0)}{1 + e^{\frac{4*m1*(-t+x1)}{y2-y0}}}$$

Parameter constraints: $y0, y2, x1, m1 \geq 0$ and $y0 < y2$

(c)

$$\text{logist}_-(t; y0, y2, x1, m1) = \frac{y0 + (y2 - y0)}{1 + e^{\frac{4*m1*(-t+x1)}{y2-y0}}}$$

Parameter constraints: $y0, y2, x1 \geq 0$, $m1 \leq 0$ and $y2 < y0$

(d)

$$\begin{aligned} d\text{logist}_+(t; y0, y1, y2, x1, xd0, xd2, m0, m2) &= \\ &= \begin{cases} y0 + \frac{y1-y0}{1+(\frac{x1-t}{xd0})^{\frac{4*xd0*m0}{y1-y0}}} & \text{if } t < x1 \\ y2 + \frac{y1-y2}{1+(\frac{t-x1}{xd2})^{\frac{4*xd2*m2}{y2-y1}}} & \text{if } t \geq x1 \end{cases} \end{aligned}$$

Parameter constraints: $y0, y1, y2, m0, x1, xd0, xd2 \geq 0$, $m2 \leq 0$,
 $xd0 \leq x1$, $y1 > y0$, $y1 > y2$

(e)

$$\begin{aligned} d\text{logist}_-(t; y0, y1, y2, x1, xd0, xd2, m0, m2) &= \\ &= \begin{cases} y0 + \frac{y1-y0}{1+(\frac{x1-t}{xd0})^{\frac{4*xd0*m0}{y1-y0}}} & \text{if } t < x1 \\ y2 + \frac{y1-y2}{1+(\frac{t-x1}{xd2})^{\frac{4*xd2*m2}{y2-y1}}} & \text{if } t \geq x1 \end{cases} \end{aligned}$$

Parameter constraints: $y0, y1, y2, m2, x1, xd0, xd2 \geq 0$, $m0 \leq 0$,
 $xd0 \leq x1$, $y1 < y0$, $y1 < y2$

For each of the five functions implement the following:

- (a) Fit the function to each quantile minimizing the Sum of Absolute Deviations (SAD). Store the parameters of the function. Calculate an overall SAD value by adding the SAD values of all 201 quantiles.
- (b) Fit the function to quantiles 0.25, 0.5 and 0.75 minimizing SAD using 10-fold cross validation [10]. Calculate an overall SAD value by adding the 10 SAD values coming from 10-fold cross validation for each of the 3

quantiles, thus the overall CV value should come from $10 * 3 = 30$ SAD values. We refer to this overall SAD value as *CV value* (Cross Validation).

4. Output a kinetics file as a result of the previous step.

2.1.3 Comparing measurements

The input of this phase are kinetics files (format specified in Section 2.1.2.2). The program should allow the analysis of the data contained in the kinetics files, specifically:

1. Display the summary of all opened kinetics files: display the CV, SAD values and parameters (using “median [quartiles]” format) for each function. Graphically distinguish the best function (out of 5) fitted to the measurement according to the CV value and allow the selection of a current function. Every analytical step described below uses the current function.
2. For each kinetics file, plot the median values and the current function fitted to the median values (0.5-quantile, 101th function out of 201).
3. Allow standardization of y values so that the median function starts at value 1 ($function(-\infty) = 1$). This means stretching all functions and median values given in the median section of the kinetics file in direction y by multiplying them with the same number that stretches the median function so that it starts at value 1.
4. Allow putting the kinetics files into arbitrary number of groups by combining the corresponding parameter values in the different kinetics files belonging to the same group.
5. Allow the display of the parameter values in a group by plotting a histogram or a boxplot.
6. Allow the user to specify that she is only interested in the parameter values of the function fitted to the median values (0.5-quantile, 101th function out of 201). We refer to this choice as “use only median functions”. Otherwise, use all the 201 values for each parameter.
7. Allow the comparison of one parameter between the groups:
 - if “use only median functions” is selected, perform a Kruskal-Wallis rank-sum test [3] (which is equivalent to a Mann-Whitney U-test or two-sample Wilcoxon test for two groups). Provide the chi-squared and p-values.

- otherwise, perform a Probability Binning (PB) test [11]. Provide the t and p-values.
8. Allow pairing kinetics files:
- if “use only median functions” is selected, the corresponding parameter of the selected pair should be subtracted from the parameter of the original kinetics file (for each parameter). The result of this subtraction should be the new parameter value.
 - otherwise, a PB comparison should be performed between the corresponding parameter of the selected pair (201 values) and the parameter of the original kinetics file (also 201 values). The t value of the PB comparison should be the new parameter value (which is a single value now).
9. Allow the comparison of paired data by Kruskal Wallis rank-sum test [3].
10. Allow exporting every result, table, image produced by the previous analysis steps in text, Excel-readable and PNG format.

2.2 Analysis of Requirements

2.2.1 Communication with other programs

Flow cytometry uses the FCS standard ([8], [13]) for storage and communication of measurement results. The program should be able to read FCS files produced by FACSDiva Version 4.1 program [15] (which outputs FCS files which are not fully standards-compliant). It should be easily extendable with newer versions of FCS or FCS files produced by other Flow Cytometry programmes. There is no state of art standard for storing gated FCS files or analysis results, however FCS 3.1 [16] aims to be a step towards this. The program should implement it’s own simple text-based format for storing gated files (gating results) and kinetics files (analysis results). For the easier usage and understanding of these own formats, we specify them formally (2.1.1.2, 2.1.2.2) and the implementation should also provide R [27] functions reading and writing these formats. The tables in these text-based formats are also easily readable by Excel for further analyis. The program should allow exporting the numerical analysis results in two formats:

- Excel-readable text format (cells separated by tab characters).
- R code which can be executed in an R environment [27] thus copying all results into R variables.

The graphical analysis results (plots) should be exportable in the widespread PNG (Portable Network Graphics) format [26] which provides lossless image compression.

2.2.2 Runtime Environment

Out of the three phases of the specification the first and third requires user interaction (Sections 2.1.1 and 2.1.3). The program(s) implementing these phases should run on several platforms without modification, including Ubuntu 2010.04 [30], Windows XP SP3, Windows Vista, Windows 7 [31] with Java SE 6 Update 31 [20] installed. The program implementing the second phase should run on Ubuntu 2010.04 [30].

This separation is possible by having a well-defined input and output format of the second phase. The communication can be done over a computer network, in this case the computer running the first and third phase should have internet access.

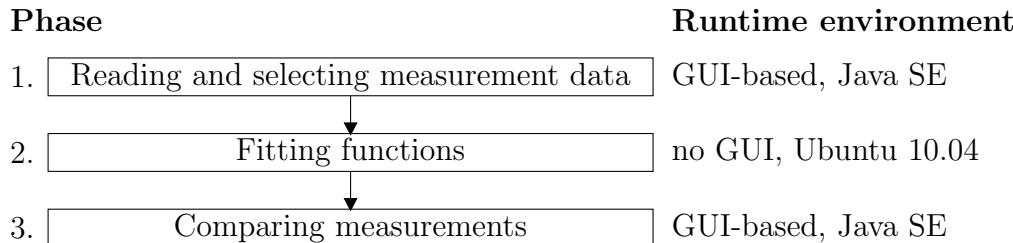


Figure 2.1: Runtime environment requirements of different phases.

2.2.3 Requirements for goodness of the program

2.2.3.1 Speed

In the first and third phase, the program should react immediately to user interaction on a recent machine such an Intel Core Duo 1.67 GHz with 2 GB of RAM (during the execution of longer tasks the program should provide a progress bar), the second phase should in average last no longer than 10 hours on an Intel Core i7 2.66 GHz machine for one gated file input.

2.2.3.2 Accessibility

The program should provide an intuitive graphical user interface that is similar to well-known Flow Cytometry analysis software such as FlowJo [19] or FACSDiva [15]. Only one instance of the program should be allowed to run at a certain time and it should be able to open several files using a tab-based interface for example. The program should have a thorough and accessible User's Guide available from a Help menu. Updating the program to a newer version should be automatic without any

user intervention. Any settings of the program should be easily modifiable using the graphical user interface.

2.2.3.3 Maintanance and failure handling

The installation and update of the program implementing the first and third phase should work without any user intervention provided the user has the installation prerequisites on her computer. The user should be able to install the software without having root access to her computer.

A power loss during phase two should not result in data loss, and the computations should restart automatically without any user intervention. The implementation of phase two should be designed for moderate usage by a couple of laboratories and should handle a load of 20 analyses pro day.

2.2.3.4 Security

The type of scientific data handled by the software is not secret and is usually made publicly available so it is not necessary to secure the communication between the second and first/third phases. Data size limits should be used to prevent overload attacks.

Chapter 3

User Documentation

The task specified in the previous chapter is solved by two programs, *FacsKin* and *Caflux*. FacsKin solves the first and third phases of the specification while Caflux solves the second phase.

3.1 Program Structure

FacsKin is a computer program that enables the mathematical description and statistical comparison of Flow Cytometry acquired kinetic measurements (such as calcium flux measurements). The output of a Flow Cytometry measurement is an FCS file containing FSC, SSC, fluorescent and time parameter values for each cell in the specimen. Kinetic measurements are special because the distribution of at least one parameter (the *kinetic parameter*) changes as time passes. FacsKin is able to describe the kinetic change that occurs during the measurement by fitting different **functions** to the kinetic parameter values. By selecting a common function that describes every measurement well, FacsKin is able to compare different groups of measurements based on parameters of the selected function. The available functions and thus the kinetic changes that FacsKin is able to describe are the following (Figure 3.1):

- **constant**: the value of the kinetic parameter is constant during the measurement timeframe
- **logist+**: the kinetic parameter starts at a given value, increases during the measurement timeframe and reaches a given value
- **logist-**: same as logist+, but the kinetic parameter value is decreasing during the measurement timeframe
- **dlogist+**: the kinetic parameter starts at a given value, increases, reaches

a maximum value and then decreases and reaches a given value during the measurement timeframe

- **dlogist-:** same as dlogist+, but the kinetic parameter value first decreases, reaches a minimum and then increases

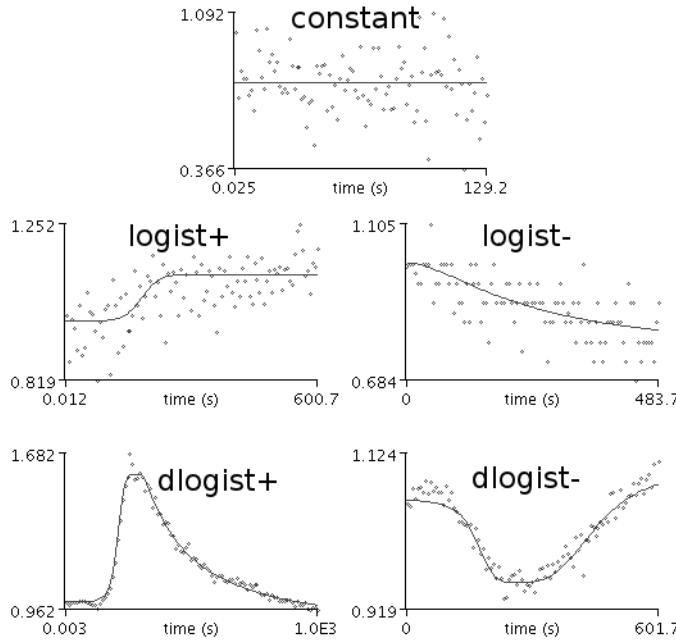


Figure 3.1: Supported functions

The name FacsKin comes from the acronym FACS (Fluorescent Activated Cell Separator) and Kinetics.

The analysis with FacsKin requires the following steps (Figure 3.2):

3.2.1 Acquiring measurement data

3.2.2 Starting FacsKin

3.2.3 Opening and gating FCS files

3.2.4 Uploading gated data

3.2.5 Receiving .kinetics file as email attachment

3.2.6 Opening different .kinetics files and selecting a common function (a .kinetics file contains the results of fitting all 5 previously described functions to the uploaded gated data)

3.2.7 Creating groups and comparing parameters of the selected function in different groups

The detailed description of the these steps follows. The usage of the server-side part of the software (called *Caflux*) is described in Section 3.4.

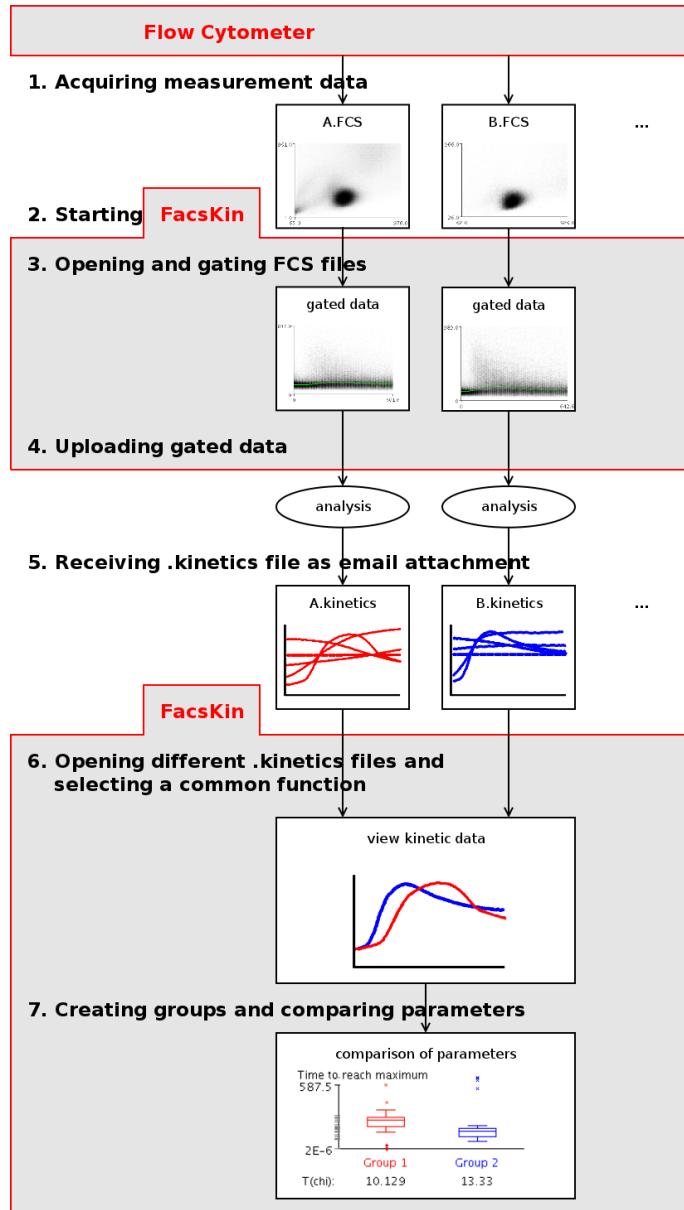


Figure 3.2: Steps of data analysis with FacsKin

3.2 FacsKin step-by-step User's Guide

The up to date version of this User's Guide is also available through the *Help / User's Guide* menu item in FacsKin. This section describes the usage of FacsKin Version 0.5.13.

3.2.1 Acquiring measurement data

FacsKin is only able to analyze those measurements that were acquired by a Flow Cytometer in compliance with these criteria:

1. All requirements specified for the particular dyes (such as staining time, pro-

tecting samples from light, vortexing the sample before measurement etc.) should be carefully followed.

2. Before starting recording data check the stream and flow rate so that the event count per second be stable and the control dot-plots look as expected. The minimum flow rate that could be analyzed with FacsKin is 200 events/s but optimally the flow rate should be around 1000 events/s.
3. IMPORTANT: before stimulating the specimen (eg. PHA activation of lymphocytes) record a baseline of at least 30 seconds length. Optimally the length of the baseline is 10% of the length of the whole measurement. Also make sure that the kinetic parameter of interest reaches a constant value before the end of the recording (for calcium flux measurements in activated lymphocytes this usually takes at least 15 minutes). This second constant time range should be optimally 10% of the whole measurement as well.
4. In case you need to pause sample acquisition and data recording for the stimulation of your sample after recording a baseline (eg. because you are using a Flow Cytometer in which the sample is not accessible during the measurement) the time taken for the stimulation of the specimen should be recorded either by the Flow Cytometer (creating an FCS with a gap between the baseline and the stimulated measurement) or the researcher (by measuring the time with a clock and recording the baseline and the stimulated measurements into separate FCS files - you will be asked for the length of this time when opening the two FCS files with FacsKin). This time should be as short as possible.
5. The sample should contain enough cells so that the desired timeframe could be recorded.
6. During the measurement the flow rate should be constant (size of the flow rate as described earlier) and the control dot-plots should be monitored at all time.
7. If possible use FCS 3.0 format when exporting data.

3.2.2 Starting FacsKin

Prerequisite for running FacsKin is Java SE 6 Update 31 [20] or later (available on Ubuntu 10.04 [30], Windows XP, Windows 7 [31], Mac OS X Tiger [14] etc.) and a computer with at least 2 GB of RAM.

FacsKin can be launched through Java Web Start from the <http://www.facskin.com> website or downloaded as a ZIP bundle and executed from the computer directly.

3.2.2.1 Launch with Java Web Start

1. To start FacsKin, click the link on page <http://www.facsSkin.com/node/5>.
2. Wait until FacsKin is downloaded automatically to your computer (Figure 3.3).

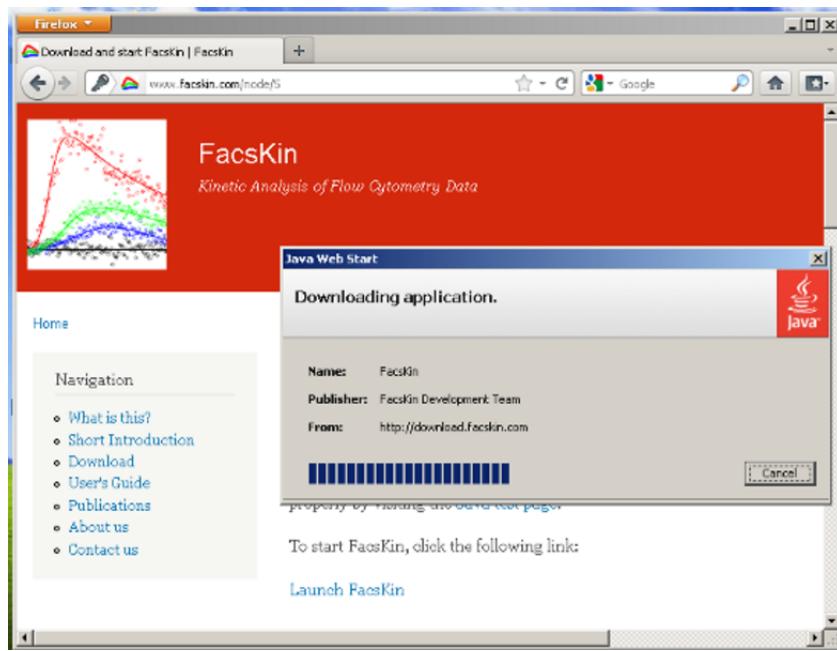


Figure 3.3: Downloading FacsKin

3. Allow FacsKin to run with full privileges on your computer. Check “Always trust content from this publisher” and click “Run” (Figure 3.4).
4. (Optional) ask your firewall to unblock FacsKin (which is running in the Java platform) so that it is allowed to upload your measurement data to the server when you click the Upload button later on (Figure 3.5)
5. FacsKin is running, you can open a .fcs or a .kinetics file (Figure 3.6).

3.2.2.2 Launch from the ZIP bundle

If you have trouble launching FacsKin with Java Web Start as described above, you can download and start FacsKin directly using the following method:

1. Download FacsKin from <http://download.facsSkin.com/FacsKin.zip> and open the downloaded zip file (Figure 3.7).
2. Copy the FacsKin folder from the zip file to your Desktop (or to any folder on your computer).

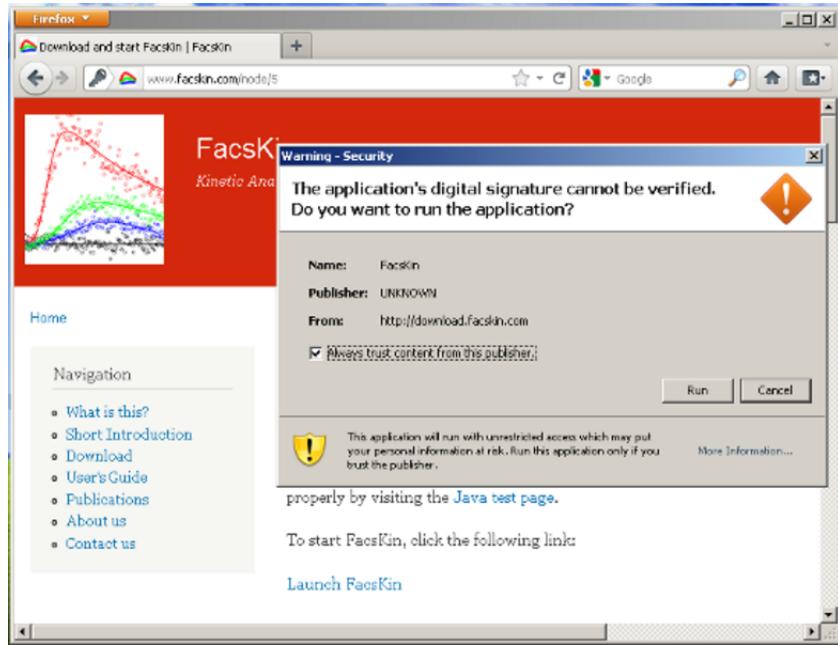


Figure 3.4: Allowing FacsKin to run with full privileges

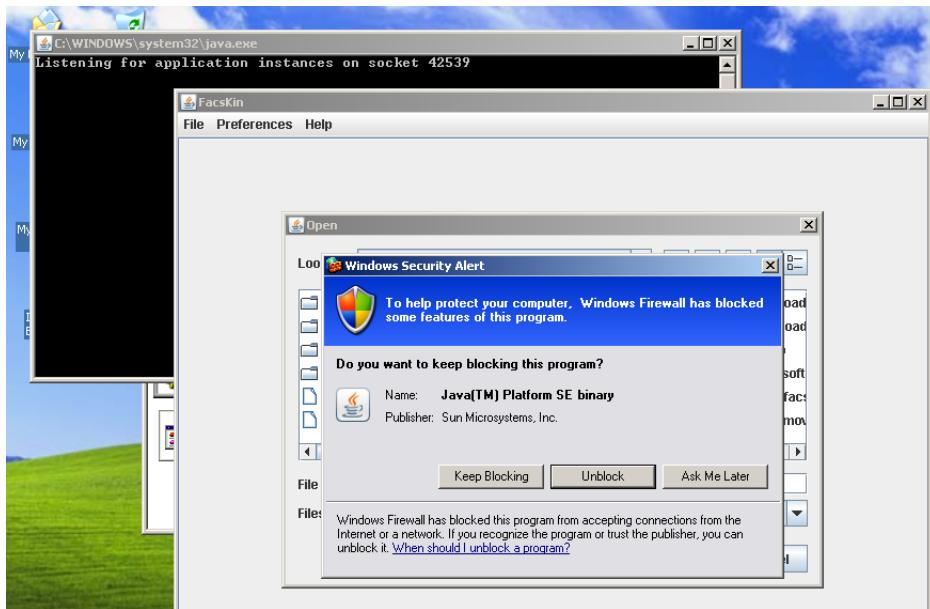


Figure 3.5: Ask the firewall to unblock FacsKin

3. To start the program click FacsKin (MS-DOS Batch File). (Figure 3.8)
4. Ask your firewall to Unblock FacsKin (which is running in the Java platform) so that it is allowed to upload your measurement data to the server when you click the Upload button later on (Figure 3.5).

Note for Linux users: to launch FacsKin with the `FacsKin.sh` provided, you should `chmod +x` it and adapt the path of FacsKin in the file and then move it to a folder which is in your `$PATH` (such as `~/bin`).

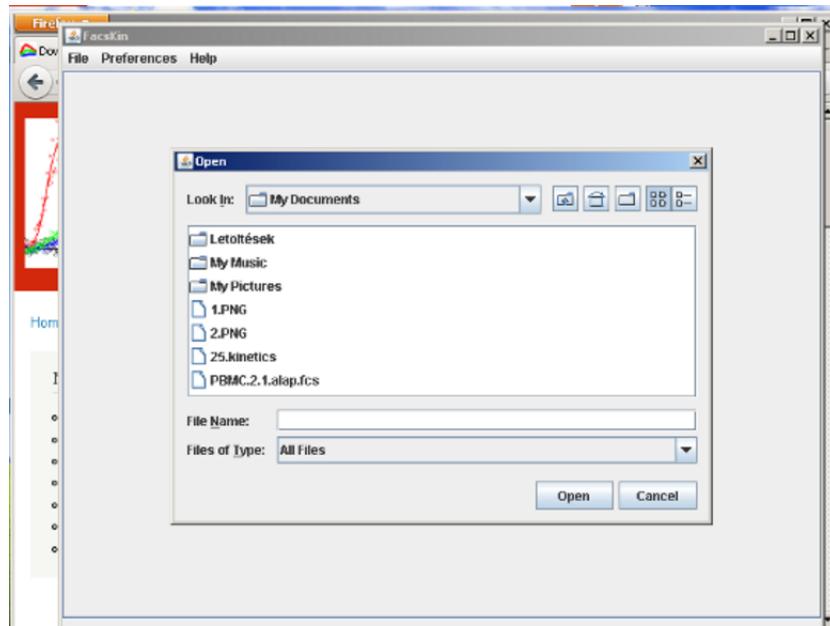


Figure 3.6: Opening a file with FacsKin

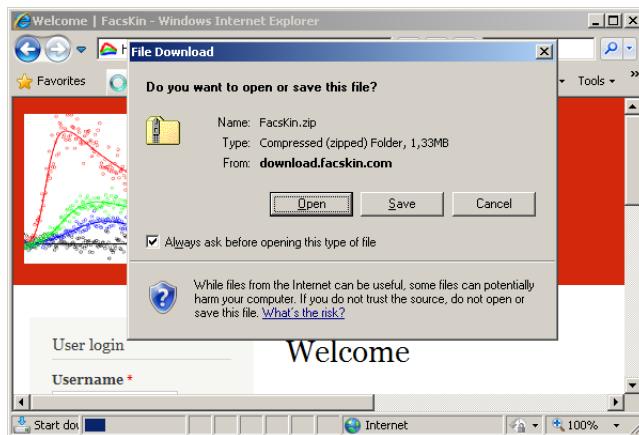


Figure 3.7: Download FacsKin

3.2.3 Opening and gating FCS files

When starting FacsKin, the Open file panel appears automatically but you can reach this panel from the *File/Open* menu item later. From the Open panel, select the FCS file you want to load and click *Open* (you can select multiple files at once, they will be opened in separate tabs).

Concatenating FCS files: If you have a separate FCS file for the baseline measurement and another one for the measurement after stimulation you should open the first measurement and then append the second one by the *File/Append* FCS menu item. A pop-up window will ask you about the size of gap (in seconds) that should be inserted between the two measurements. This value should be as close to the real value as possible because it can influence the analysis. The default

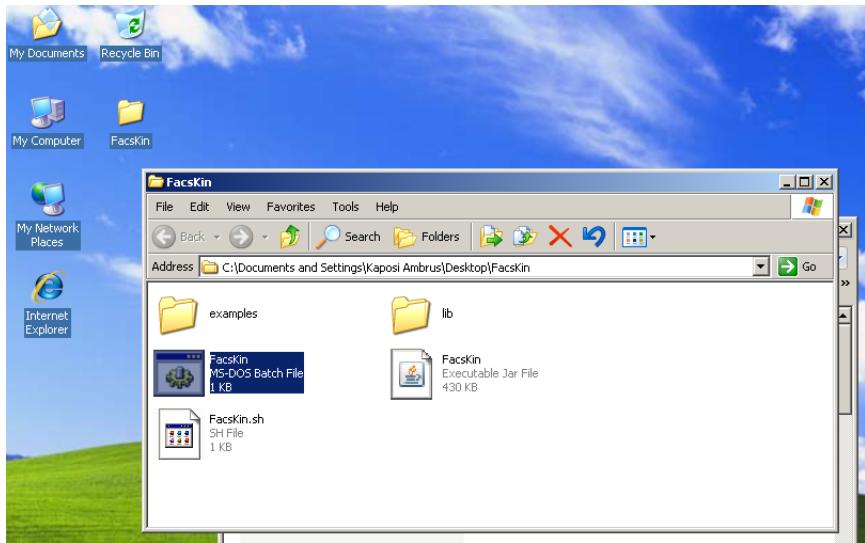


Figure 3.8: Start FacsKin

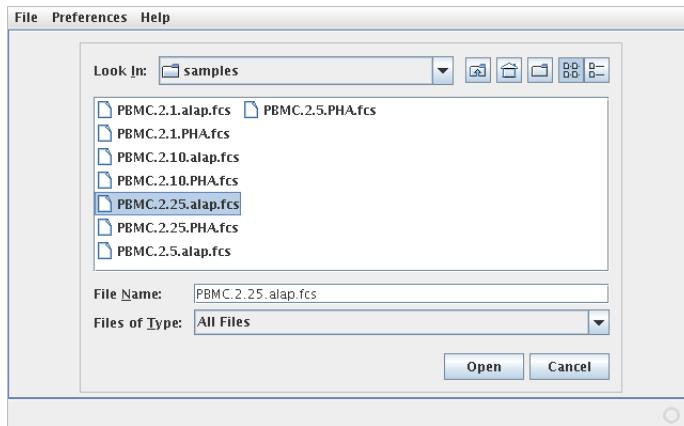


Figure 3.9: Opening an FCS file

value is 30 seconds.¹

After opening an FCS file a tab will appear labeled with the name of the FCS file. In the middle you will see the **scatter plot** of the selected channels (by default FSC (forward scatter) on the horizontal and SSC (side scatter) on the vertical axis). You can select the channel on the corresponding axis by clicking on the drop down menu near the corresponding axis. You can display the selected channel on a **logarithmic scale** by clicking the *Log* checkbox. To view the **histogram** of the channel selected on the horizontal axis, select *Histogram* from the drop down menu for the vertical axis. On the right side you can see some information about the measurement (filename, FCS version, date of measurement and count of events).

Gating:

- **Scatter plot gate:** to pick out events based on two parameters, select the

¹You can check whether the measurement was appended correctly by selecting the Time parameter on the horizontal axis. You should notice a gap of the same size you selected.

desired parameters on the horizontal and vertical axes, and select the vertices of the polygon by clicking on the scatter plot with the left mouse button. The semi-finished polygon is drawn with red color and if you finished the selection it is drawn with green color. To restart the selection of the polygon click on the scatter plot with the right mouse button. You can select a rectangle instead of a polygon by selecting *Rectangular gate* on the left.² (Figure 3.10)

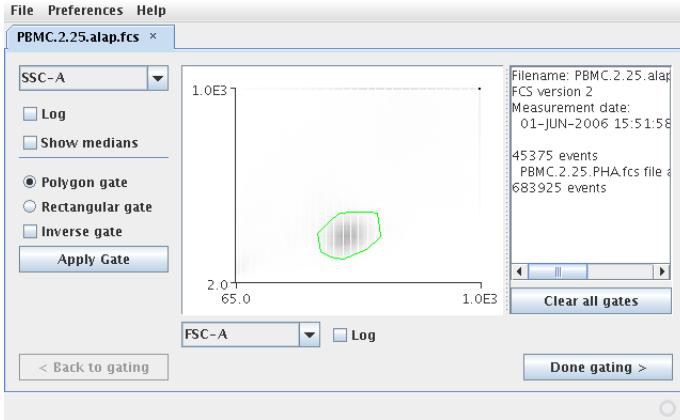


Figure 3.10: Creating a scatter plot gate

- **Histogram gate:** to pick out events based on one parameter, select *Histogram* on the vertical axis and select the area on the histogram plot by clicking the left mouse button and dragging until selection end. The red rectangle shows the currently selected range. (3.11)

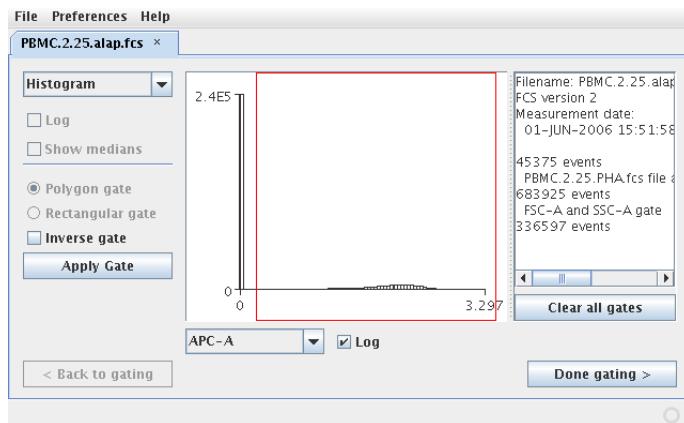


Figure 3.11: Creating a histogram gate

By clicking the *Apply Gate* button you can apply the selected gate. If you want to pick out the events outside the selected area, select the *Inverse gate* checkbox

²It is especially helpful when gating based on the time parameter. This is sometimes important when there was some disturbance during a time point of the measurement. In this case the time range where the disturbance occurred should be gated out with inverse gate, see below.

before clicking *Apply Gate*. You can follow the change in the number of events on the right side of the window.

Currently you cannot **cancel gates** individually but you can restart gating by clicking the *Clear all gates* button.

To view the change in the value of the **measured kinetic parameter** over time, select the time channel on the horizontal axis, select the channel for the kinetic parameter on the vertical axis and click the *Show medians* checkbox (Figure 3.12). The green curve shows how the median values change over time. From this plot you can get an impression about the type of kinetic response the measurement shows. If the kinetic response shown by this plot can't be described by the available functions (eg. the value of the measured kinetic parameter first increases, then decreases, then increases again - out of the 5 functions available none can describe the last increasing part) you should consider creating a gate based on the time parameter selecting only a region where the change of the measured kinetic value could be described by one of the functions.

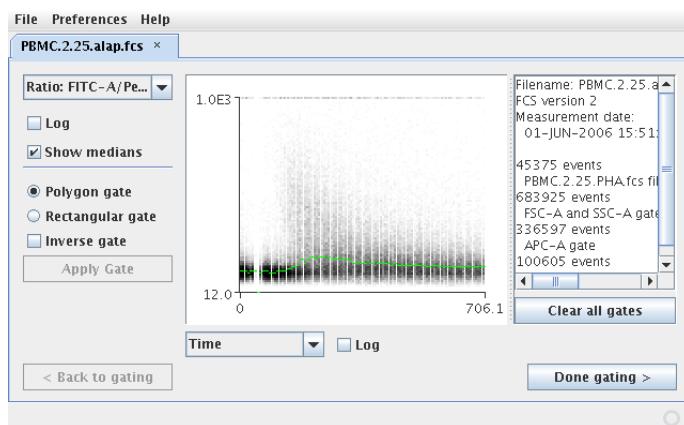


Figure 3.12: Viewing the change of the kinetic parameter over time

When you gated the desired cell population finish gating by clicking the *Done gating* button.

3.2.4 Uploading gated data

The analysis of the gated data is performed on the server hosted by Semmelweis University, Budapest, Hungary. That's why you have to upload the data to the server. You can do this by the following steps (Figure 3.12):

1. Select the measured **kinetic parameter**. If you have a channel for that parameter you should choose the appropriate channel in the left drop down menu under the *Choose parameter ratio to be analyzed against time* label and leave the value 1.0 in the right drop down menu. If you want to analyze the **ratio** of

two parameters (eg. Fluo-3/Fura Red corresponding to FITC/PerCP), choose the appropriate parameters in the nominator (left) and denominator (right) checkbox.

2. Choose a **name** for the analysis. You will receive the analysis results (a .kinetics file) with the name you have chosen here.
3. Give your email address. The .kinetics file containing the analysis results will be sent to this email address.
4. Click the *Upload gated data* button and wait till the data uploads. On success, you should be presented with a *Data uploaded successfully. You will receive...* infobox.

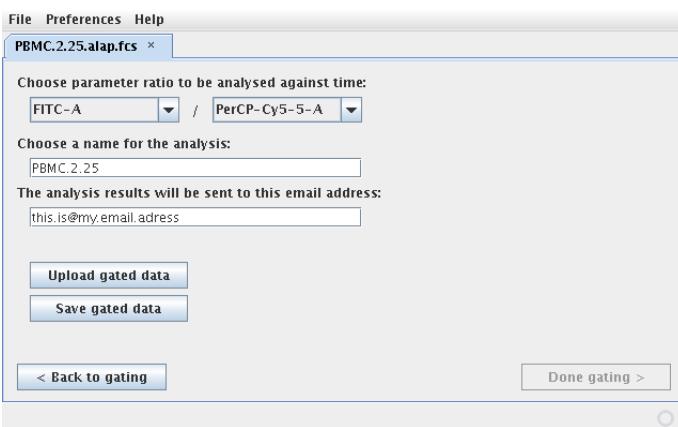


Figure 3.13: Uploading gated data

Troubleshoot: If you receive a question from your firewall whether it should allow FacsKin to have network access, please give FacsKin access to be able to upload data. If you receive a "Network Error" message from Facskin after clicking the *Upload gated data* button, please check your firewall configuration and allow FacsKin to have network access.

If you want to save the gated data on your computer you can do this by clicking the *Save gated data* button. This way you will get the exact same file as the one you uploaded to the server. It is a gzip-compressed file containing the gated data in text format. Currently this file cannot be opened by FacsKin.³

3.2.5 Receiving .kinetics file as email attachment

Some hours after uploading, the server sends you a .kinetics file containing the analysis result of your uploaded measurement (gated data). A .kinetics file contains

³The upload to the server is not secured and we cannot guarantee that the uploaded data won't be accessed by a third party.

the results of fitting all 5 previously described functions to one gated measurement. The name of the .kinetics file is the same you specified in FacsKin before uploading the gated data.

3.2.6 Opening different .kinetics files and selecting a common function

You can open .kinetics files in FacsKin just as you did it with FCS files. A tab named Kinetics will be opened where you see the following (Figure 3.14):

- (green) on top you see a plot showing the **median points** of the measurement and the currently selected **function** fitted to these median points. The median points are the medians of the selected kinetic fluorescent parameter values in the uploaded gated cell population (exactly the same points as the green lines shown in Figure 3.12).
- on the bottom you see a table where each row corresponds to an opened .kinetics file.
- (red) to **choose between functions**, click on the column header of the gray-backgrounded columns. To select the constant function, click *constant* etc. Note how the plot changes while selecting different functions.
- (blue) note how the last columns of the table change when selecting different functions: each function has a different set of parameters.
- while moving the mouse over the plot you can **track the coordinates** with the cursor. When selecting *Use Only Median Functions* only the parameters of the function fitted to the median points appear in the table. Try to match the parameters in the table with the corresponding coordinates seen on the plot (eg. maximum value parameter in the table should have the same value as the second coordinate of the highest value of the function as seen on the plot).
- (purple) by default the kinetic parameter is **standardized** so at time point 0 the value of the median point is 1 (the vertical axis shows relative parameter values). If you deselect *Preferences/Standardization*, the function will start at the real measured value (note how the values on the vertical axis change in the plot - the vertical axis shows *absolute* parameter values). Standardization can help when you wan't to compare measurements that were measured under different conditions / different Flow Cytometer machines etc.

- to open more .kinetics files click *File/Open*. You can select more .kinetics files at once.
- to save the currently opened .kinetics files in one zip file so that you can easily open them next time at once, click *File/Save as*. Next time just open the zip file with FacsKin and each .kinetics file contained in that zip file will be loaded.
- (light blue) to remove a .kinetics file right click the row of the .kinetics file in the first column (*Filename*) and select *Remove row*
- (light blue) to get information about a .kinetics file such as measurement time, measurement name right click the row of the .kinetics file in the first column (*Filename*) and select *Get info*

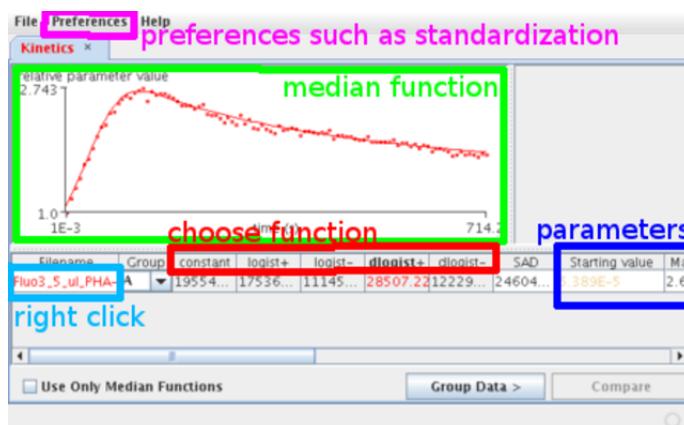


Figure 3.14: FacsKin after opening one .kinetics file

3.2.6.1 Detailed description of the functions

Each function has an own set of parameters. These parameters describe the function entirely: eg. if you know all the 8 individual parameters of a dlogist+ function you will be able draw the function. The available functions and the parameters describing them are the following:

- each function has an **AUC** parameter:
 - **AUC**: the area under curve from time point 0 to by default 600 s (this can be changed in *Preferences/Set Maximum Time for AUC* menu item - set it to length of the measurement). Eg. if the vertical axis is calibrated for intracellular Calcium concentration (mmol/l), the AUC shows the amount of time all the Calcium atoms spend in one liter of cell volume during the measurement timeframe.

- **constant**: the function is a horizontal line having the same value all the time.

Parameter:

- **constant value**

- **logist+**: an S-shape function that starts at a given value (starting value) increases and reaches a higher given value (ending value). (Figure 3.15)

Parameters:

- **starting value**: the limit of the function at $-\infty$ (minus infinity). It is not necessarily the value at time point 0. If the function begins with a steep, the starting value is lower than the value at time point 0. This is one reason why it is necessary to record a baseline at the beginning of the measurement
- **ending value**: the limit of the function at $+\infty$ (positive infinity). Not necessarily the value of the function at the end of the measurement
- **time to reach 50% value**: the time point when the function reaches the 50% value. The 50% value is the average of the starting value and the ending value (unit: s)
- **slope at 50% value**: the slope of the function at the 50% value. This is always positive (unit: int/s where int is the unit of the vertical axis. Meaning: how much does the intensity change during 1 second)

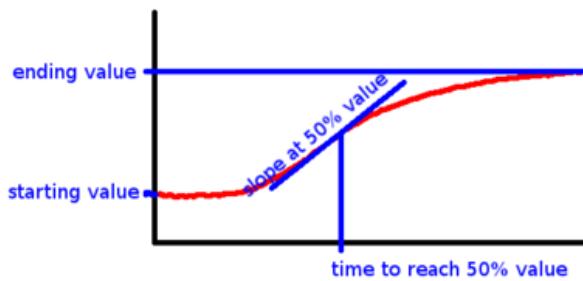


Figure 3.15: Parameters of logist+ function

- **logist-**: an S-shape function that starts at a given value (starting value) decreases and reaches a lower given value (ending value). (Figure 3.16)

Parameters:

- **starting value**: the limit of the function at $-\infty$ (minus infinity). It is not necessarily the value at time point 0. If the function begins with a steep, the starting value is higher than the value at time point 0. This is one reason why it is necessary to record a baseline at the beginning of the measurement

- **ending value**: the limit of the function at $+\infty$ (positive infinity). Not necessarily the value of the function at the end of the measurement
- **time to reach 50% value**: the time point when the function reaches the 50% value. The 50% value is the average of the starting value and the ending value (unit: s)
- **slope at 50% value**: the slope of the function at the 50% value. This is always negative (unit: int/s where int is the unit of the vertical axis. Meaning: how much does the intensity change during 1 second)

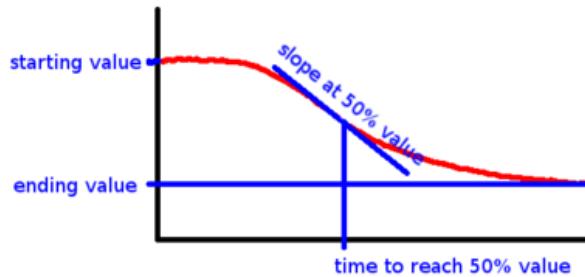


Figure 3.16: Parameters of logist- function

- **dlogist+**: a function that starts at a given value, has an increasing phase, reaches a maximum, has a decreasing phase and reaches a given ending value. (Figure 3.17)

Parameters:

- **starting value**: the limit of the function at $-\infty$ (minus infinity). It is not necessarily the value at time point 0. If the function begins with a steep, the starting value is lower than the value at time point 0. This is one reason why it is necessary to record a baseline at the beginning of the measurement
- **maximum value**: the maximum of the function. It is possible that the maximum point is not in the measurement timeframe (usually meaning that the selected function doesn't fit the measurement well)
- **ending value**: the limit of the function at $+\infty$ (positive infinity). Not necessarily the value of the function at the end of the measurement
- **time to reach maximum value**: the time point when the function reaches the maximum value (unit: s)
- **time from the first 50% value to maximum**: the distance between the time point where the function reaches the first 50% value and where the function reaches the maximum. The first 50% value is the average of the starting value and the maximum (unit: s)

- **slope at first 50% value:** the slope of the function at the first 50% value. It is always positive (unit: int/s where int is the unit of the vertical axis. Meaning: how much does the intensity change during 1 second)
- **time from maximum to the second 50% value:** the distance between the time point where the function reaches the maximum and where the function reaches the second 50% value. The second 50% value is the average of the maximum and the ending value (unit: s)
- **slope at second 50% value:** the slope of the function at the second 50% value. It is always negative (unit: int/s where int is the unit of the vertical axis. Meaning: how much does the intensity change during 1 second)

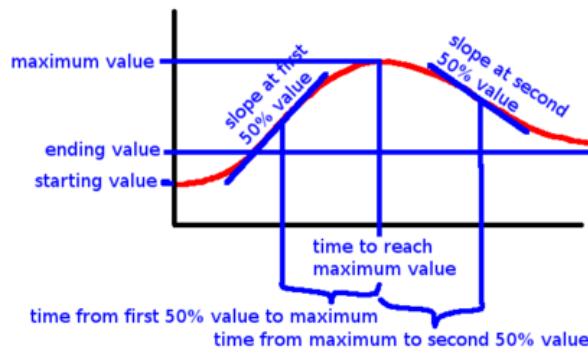


Figure 3.17: Parameters of dlogist+ function

- **dlogist-:** a function that starts at a given value, has a decreasing phase, reaches a minimum, has an increasing phase and reaches a given ending value. (Figure 3.18)

Parameters:

- **starting value:** the limit of the function at $-\infty$ (minus infinity). It is not necessarily the value at time point 0. If the function begins with a steep, the starting value is higher than the value at time point 0. This is one reason why it is necessary to record a baseline at the beginning of the measurement
- **minimum value:** the minimum of the function. It is possible that the minimum point is not in the measurement timeframe (usually meaning that the selected function doesn't fit the measurement well)
- **ending value:** the limit of the function at $+\infty$ (positive infinity). Not necessarily the value of the function at the end of the measurement
- **time to reach minimum value:** the time point when the function reaches the minimum value (unit: s)

- **time from the first 50% value to minimum:** the distance between the time point where the function reaches the first 50% value and where the function reaches the minimum. The first 50% value is the average of the starting value and the minimum (unit: s)
- **slope at first 50% value:** the slope of the function at the first 50% value. It is always negative (unit: int/s where int is the unit of the vertical axis. Meaning: how much does the intensity change during 1 second)
- **time from maximum to the second 50% value:** the distance between the time point where the function reaches the minimum and where the function reaches the second 50% value. The second 50% value is the average of the minimum and the ending value (unit: s)
- **slope at second 50% value:** the slope of the function at the second 50% value. It is always positive (unit: int/s where int is the unit of the vertical axis. Meaning: how much does the intensity change during 1 second)

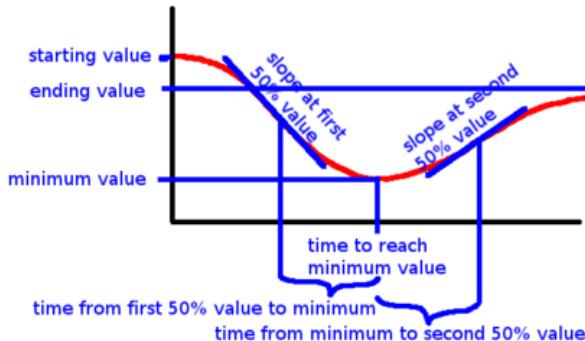


Figure 3.18: Parameters of dlogist- function

How to choose the best function (Figure 3.19): to compare measurements you have to select a common function that describes every .kinetics file that you opened. Most of the time it is obvious from the type of measurement which function to choose but we give some criteria that should be met to avoid making mistakes:

- select the function that **describes the kinetic response** of the measurement best: the plot shows you how well the fitted function follows the median values. Eg. if the median values are increasing with time during the measurement you should consider selecting the logist+ function.
- select the function with the **lowest CV value**. CV means cross validation values. The lower the CV value the closer the function to the median values. The CV values are shown for each function in the column corresponding to the function. The lowest CV-value for each .kinetics file is written with *red color*.

A small difference (5% difference should be considered small) in CV values of two functions means that the two functions are almost equally good in this aspect and you should make a choice between them based on the other criteria.

- if there are still more functions to consider you should select the **simpler function**: the function that is more to the left in the table and with fewer parameters. If you select a too complex function for a simple measurement kinetic some of your parameters will be without meaning. Eg. if you select the dlogist+ function for a measurement with logist+ kinetic the parameters that describe the decreasing phase of the dlogist+ function (time from maximum to second 50% value, slope at second 50% value, ending value) will have no biological meaning and will have random values with big error. The parameters outside the measurement timeframe are printed with *bright color* and suggest that the selected function is too complex for the measurement. Another reason for having bright colored parameters can be that you haven't recorded a baseline before starting the stimulation and thus the starting value and related parameters will have a value outside the range of the median values. This is why you should always record a baseline.
- if you opened multiple .kinetics files you should select the **best common function** that describes all measurements. If there are some measurements that don't follow the selected function (eg. don't have a decreasing phase while most of the measurements have an increasing and a decreasing phase) be prepared that some of the parameters (in the case of our example the parameters describing the decreasing phase) won't have a biological meaning. Another solution to this problem would be to gate every FCS and upload them for analysis again so that the each uploaded dataset only contains only the events from the same phase (only from the increasing phase in our example).

Each function is fitted 201 times to different quantiles of the kinetic values so as to enable the description of the whole data range of the measurement (Figure 3.20). That's why every parameter in the table is given as a **parameter distribution** (*median [quartiles]* values). You can think of these parameter distributions in a similar way as of the distributions of directly measured parameters like FSC, SSC and fluorescent parameters: each event in the measurement has an FSC parameter and you are allowed to view the FSC-distribution of a gated set of events. For kinetic parameters like starting value, maximum etc. we don't know the individual values for each event but we are able to view and compare (see below) the distributions.

If you select *Use Only Median Functions* only the parameters of the median function are shown in table and only these are used for further comparison (so eg.

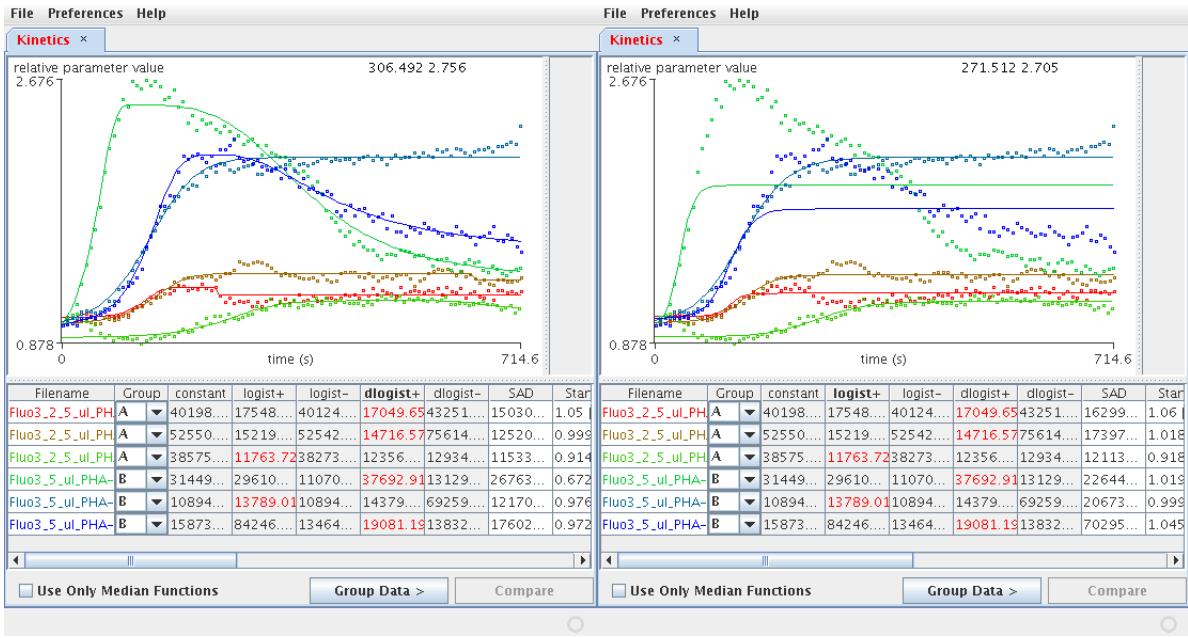


Figure 3.19: Function selection when the measurements have different kinetics. Dlogist+ is selected on the left side and logist+ on the right side. In most of the measurements it makes no difference which function is being used but in the 4th (green color) and 6th (blue) files there is real difference in the CV values and in the plot as well. In this case it is possibly wise to use the dlogist+ function even if the parameters describing the decreasing phase will have no real meaning for the other measurements (1st, 2nd, 3rd and 5th).

from one measurement you will only get one maximum parameter value instead of a whole distribution for the maximum parameter).

3.2.7 Creating groups and comparing parameters of the selected function in different groups

FacsKin allows you to compare measurements based on:

- whole parameter distribution for each parameter for each measurement (*Use Only Median Functions* deselected) or
- just the parameters of the median functions - one value for each parameter for each measurement (*Use Only Median Functions* selected)

The latter method does not take into account the whole range of the measured values, only the middle (median) of the measurement but it allows us to use traditional nonparametric statistical methods for comparison.

Creating groups: in the table, by clicking on the drop down menu in a row of a .kinetics file you can assign a group to that .kinetics file. To create a new group select (*new*) from the drop down menu and give a name to the new group. Each

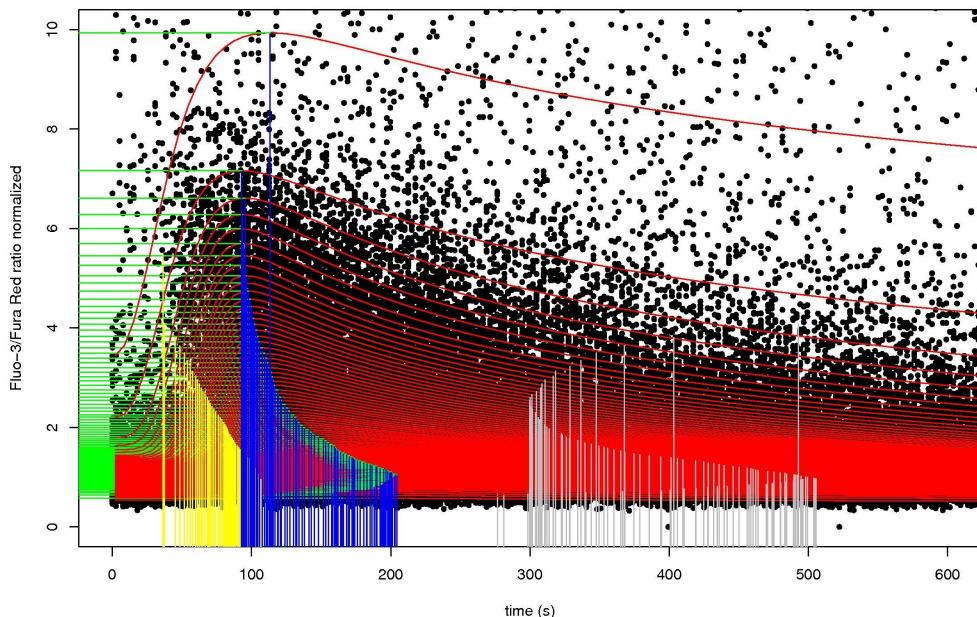


Figure 3.20: The distribution of a parameter derives from fitting a function 201 times to the same measurement. Plot of a calcium flux measurement (black dots), the fitted dlogist+ functions (red) and their parameters: maximum value (green), time to reach maximum value (blue), time point of the first 50% value (yellow), time point of the second 50% value (gray).

newly opened .kinetics file will be given the group of the .kinetics file in the last row. To view the parameter values aggregated by groups click the *Group Data* button. The resulting table shows one row per group and the parameter distributions are summed from the .kinetics files that are members of that group. You can create arbitrary number of groups. The number of the parameter values in each group is given in column *size*. If *Use Only Median Functions* is selected the size of each group is the number of .kinetics files in that group. Otherwise the size is 201 times the number of .kinetics files in the group since the distribution of each parameter is given as 201 values. A color is assigned to each group and all the functions and median points in the plot that correspond to a group are drawn with the same color.

Comparison: to compare the distribution of a given parameter between different groups select the column of the parameter by clicking the column and click the *Compare* button.

- when *Use Only Median Functions* is deselected, a new tab labeled Comparison (by groups) is opened (Figure 3.23). This tab contains the box plot or histogram of the parameter distribution in each group. You can choose between **box plot** and **histogram** plot by selecting the corresponding checkbox

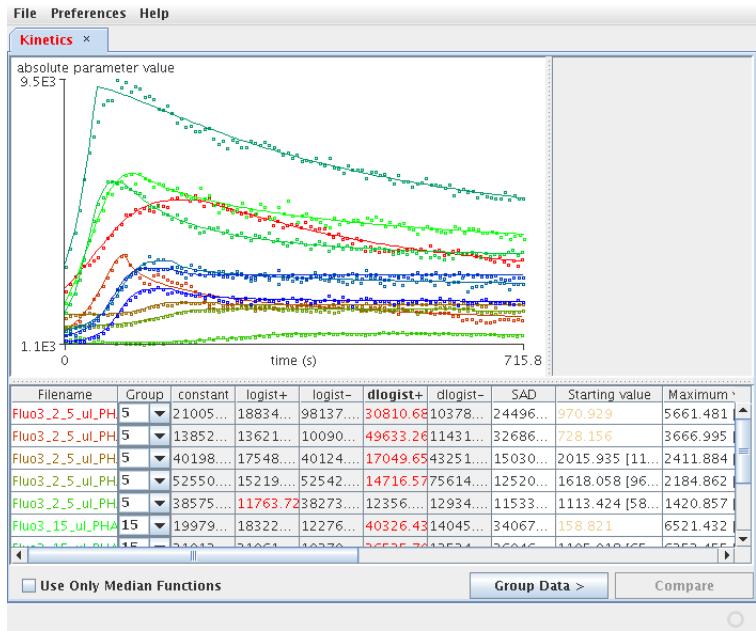


Figure 3.21: Opening several .kinetics files and creating two groups: 5 and 15. The selected function is dlogist+.

under the plot. The compared parameter is printed on the left top of the plot. Each group is colored with the same color as in the *Kinetics* tab. On the right there is a table containing values of the statistical comparison. Each row corresponds to one group. The T(chi) values of a **probability binning comparison** (see [11]), the corresponding p-values and the **overlap of the middle 50% of the distributions** are shown in the table for each row. These values are calculated against the control group specified in the right bottom corner in the drop down menu *Control group*. By default the control group is the combination of all groups but you can select one of the groups as well as control. You can change the bin count for the probability binning comparison by setting *Bin count* and clicking *Set* but usually the default bin count is sufficient. The bigger the T(chi) value the more the distribution differs from the control distribution. The value that holds a biologically relevant difference should be determined individually for each type of measurement. The same is true for the p-value and the overlap. The lower the overlap the bigger the difference between the examined and control distribution.

- when *Use Only Median Functions* is selected, a new tab labeled *Median comparison (by groups)* is opened (Figure 3.24). This tab contains the results of a **Kruskall-Wallis rank-sum test** comparing the parameter values in the selected groups. In the case of two groups this test is equivalent to the Wilcoxon rank-sum test (Mann-Whitney U-test). The chi square value, the p-value and the degrees of freedom is given as well as some descriptive statistical parameters.

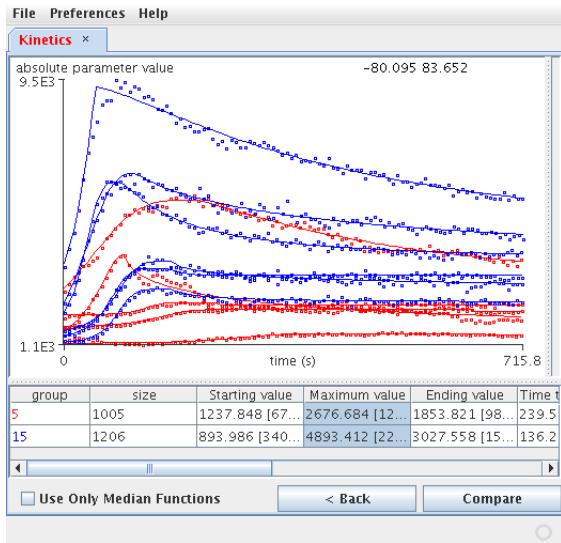


Figure 3.22: Grouping data and selecting the Maximum value parameter for comparison.

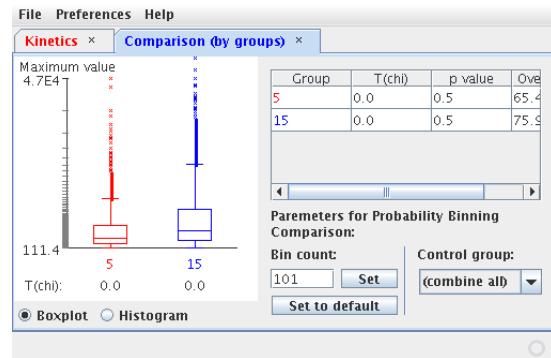


Figure 3.23: Comparing the maximum parameter in two groups.

ters for each group. In this case only the median range of each measurement is compared so the comparison doesn't take into account the whole range of the measurement data.

- when clicking *Compare* without grouping data the box plot and comparison results for each individual .kinetics file will be shown.

3.2.7.1 Exporting comparison data

- by right clicking the plots in the *Kinetics* and *Comparison* tabs you are able to export the image as a PNG file
- by right clicking the table in the *Kinetics* or *Comparison* panel you can copy the contents of the table to the clipboard (*Copy table contents*). You can paste the table directly into a spreadsheet program such as Excel.
- by right clicking the column of a parameter in the *Kinetics* tab you can copy

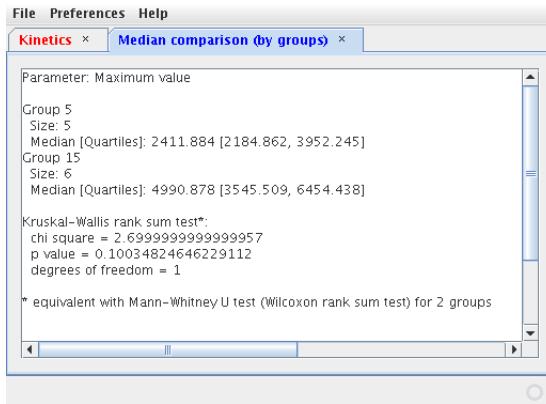


Figure 3.24: Comparing the maximum parameter in two groups using only values from the median functions.

the distributions of the selected parameter in each group/.kinetics file to the clipboard (*Copy parameter data*). You can paste these values directly into a spreadsheet program such as Excel.

- by right clicking the name of a group/.kinetics file (first column in the table) in the *Kinetics* tab you can copy the distributions for every parameter to the clipboard (*Copy parameter data*). You can paste these values directly into a spreadsheet program such as Excel.
- by right clicking the table in the *Kinetics* tab you can export all data contained in the .kinetics files opened to the clipboard and you can paste it as R code into the R statistical programming environment (<http://www.r-project.org>) for further statistical analysis. You can do this in the *Median comparison (by groups)* panel as well by selecting the text under *#R code for comparison* and clicking Ctrl-C.

3.2.7.2 Pairing measurements

If you have logically paired measurements, such as a control and a test measurement for each sample and you would like to test whether the difference between the control and the test measurement differs among different groups of measurements you can use the Pair Data feature:

- For each test measurement select the corresponding control measurement from the drop-down menu in column Pair (Figure 3.25).
- Select the appropriate groups in the *Group* column.
- Click the *Pair Data* button. Depending on whether you selected *Use Only Median Functions*, the result is the following:

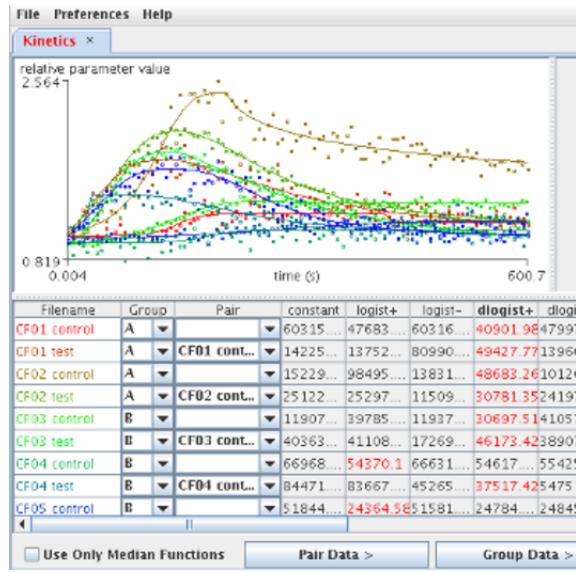


Figure 3.25: Selecting pairs for each test measurement.

- If *Use Only Median Functions* was not selected, a probability binning comparison is performed between each test measurement and its corresponding control and the T(chi) values are given in the table (Figure 3.26).
- If *Use Only Median Functions* was selected, the parameter of the median function of the control measurement is subtracted from that of the test measurement for each corresponding parameter and for each measurement.

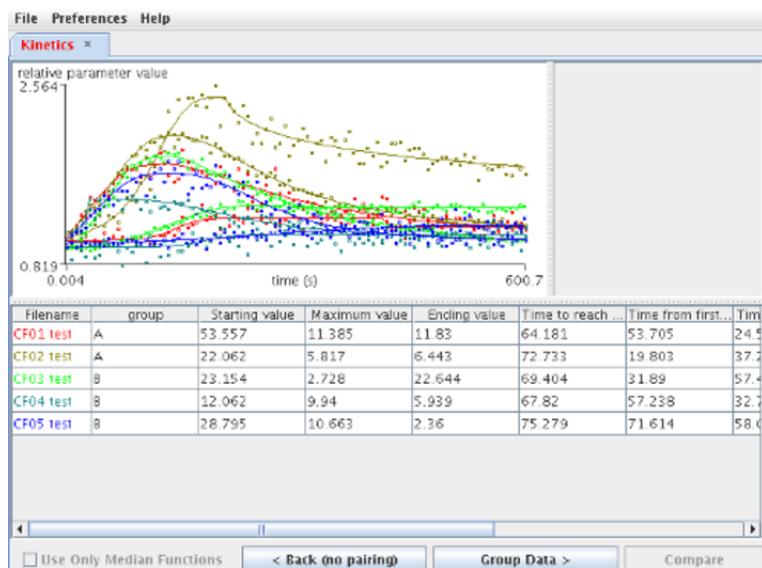


Figure 3.26: Measurement data after pairing. In this case (not selecting *Use Only Median Functions*) the probability binning comparison results between the control and the test measurements are shown.

- Click the *Group Data* button. This groups the values calculated in the previous step as specified in the *Group* column (Figure 3.27)

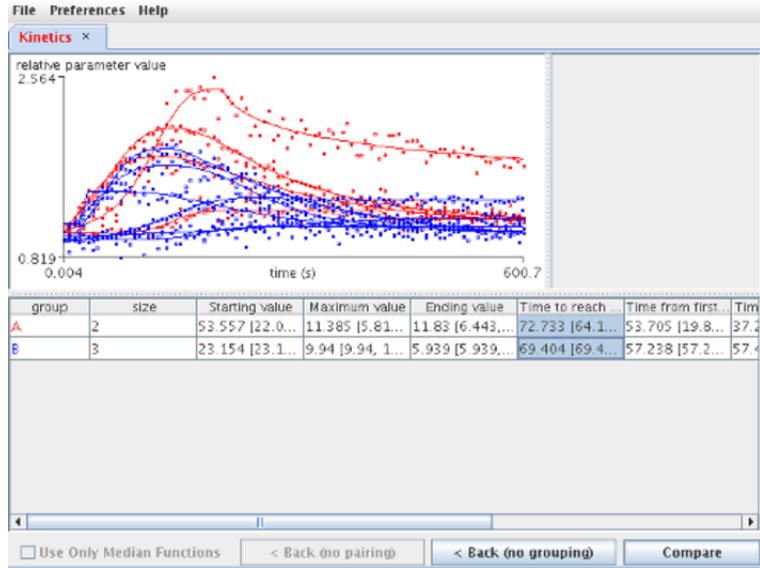


Figure 3.27: Grouping paired comparison data. The parameter Time to reach maximum is selected for comparison.

- You can compare a parameter among groups with Kruskal-Wallis test by selecting the column corresponding to the parameter and clicking the *Compare* button.

3.3 Maintenance of FacsKin

The installation of FacsKin is automatic if started with Java Web Start. Upgrades to newer versions will be applied automatically. Before starting FacsKin, Java Web Start checks whether there is a newer version and if there is, it downloads the new version of the software and starts it. To uninstall FacsKin, launch the Java Configuration tool (Windows: *Start menu / Settings / Control Panel / Java*, “jcontrol” tool on Linux) and in the *General* tab, under the *Temporary Internet Files* title click the *View* button, select *Applications* from the top-left drop-down menu, select FacsKin from the list and press the delete button (or click the red X button on the top). An alternative way of uninstalling FacsKin on Linux is deleting the `.java` folder in your home directory. Note that this also uninstalls all other Java Web Start applications and clears all Java settings on your system.

If FacsKin was installed using the zip bundle method (Section 3.2.2, Sub-section “Launch from the ZIP bundle”), it does not detect new versions automatically. To check whether a new version is available, look at the top-left corner of the website <http://www.facsSkin.com> and compare the version given

there with the one mentioned in the About Box (*Help / About*). The newest version of FacsKin is always available as a ZIP bundle on the following URL: <http://download.facskin.com/FacsKin.zip>. To uninstall FacsKin installed by this method simply delete the containing folder.

The user data of FacsKin on Linux is stored in `~/.FacsKin`, on Windows it is stored in the `Application Data` folder in the user's home directory. The size and position of the FacsKin window is stored here. When uninstalling FacsKin, this data will remain on the computer, but you can safely delete it any time.

3.4 Caflux User's Guide

The server side software which implements phase two (Section 2.1.2) of the specification is called *Caflux*. (The name comes from Calcium Flux, which is a typical kinetic flow cytometry measurement.) This software contains an R script that reads a gated file and generates a kinetics file and wrapper bash scripts which handle the communication and storage of data. The installation and maintenance is not automatized, in order to make the administrator of Caflux more conscious about the structure of the software and help him correct arising problems.

3.4.1 Installation

Installation prerequisites:

- **Computer:** it is recommended to install Caflux on a computer with a processor equivalent to or faster than an Intel Core i7 2.66 GHz (4 cores), having 2 GB of RAM and 1 TB of hard disk space. Caflux runs on machines with much lower specification but to provide reasonable analysis times (in average less than 10 hours for one gated data file), it is recommended to run it on a powerful machine. Ideally this should be a dedicated computer for analysis.
- **Operating System:** a recent Linux distribution is required to run Caflux such as Ubuntu 10.04 [30]. It is recommended to install a Long Time Support (LTS) distribution.
- **FTP storage** with at least 1 GB of space. FacsKin uploads measurement data to this FTP storage and Caflux looks at this storage from time to time for new gated data files uploaded for analysis. For security you should set a data size limit of 300 MB for uploaded files.
- **Email account** with SMTP interface for sending emails. The measurement results (kinetics files) will be sent back to the users using this email account.

The installation requires the following steps:

1. Install the following packages that are required by Caflux (Ubuntu package names): `libio-socket-ssl-perl`, `r-base`, `curlftpfs`.
2. Create a user named `caflux` without administrator privileges (command `useradd`), the home folder should be `/home/caflux`. Make sure that the user belongs to `fuse` group. This is required because the communication with the FTP server is done using `curlftpfs` which uses FUSE (Filesystem in USErspace).
3. Unzip the contents of `caflux.zip` to `/home/caflux`, thus creating a folder `/home/caflux/caflux`.
4. Execute the following commands to compile `dlogistx.c` which is required by the R scripts:

```
cd /home/caflux/caflux
R CMD SHLIB dlogistx.c
```

5. Execute the following commands to download and unpack the `sendEmail` program [29] which Caflux uses for sending emails:

```
cd /home/caflux/caflux
wget \
http://caspian.dotconf.net/menu/Software/SendEmail/\
sendEmail-v1.55.tar.gz
tar xvzf sendEmail-v1.55.tar.gz
mv sendEmail-v1.55/sendEmail .
rm -rf sendEmail-v1.55*
```

6. Edit the `passwords` file with the data of the FTP storage and an SMTP server details of the mail account. Make sure that parameters having spaces are quoted, such as:

```
MAILFROMFIELD="\"FacsKin Team <calciumflux@gmail.com>\""
```

7. Set necessary permissions to the scripts:

```
cd /home/caflux/caflux
chmod go-rwx *
chmod u+x caflux_* *.sh sendEmail passwords
chmod u+r *
chmod u-w * # we turn off write mode to avoid accidental modification
```

- Create folders containing analysis results with the `create_folders.sh` script:

```
cd /home/caflux/caflux
./create_folders.sh
```

- Copy the contents of `rc.local` to the end of `/etc/rc.local` so that the analysis automatically starts when the computer is turned on.
- Restart the computer. The script should be running from now on.

3.4.2 Maintenance

The list of files in `/home/caflux/caflux` and their purpose is listed in Tables 3.2 and 3.1. The data flow between components is shown in Figure 3.28. Gated files have unique names which contain the date of upload and the email address of the uploader.

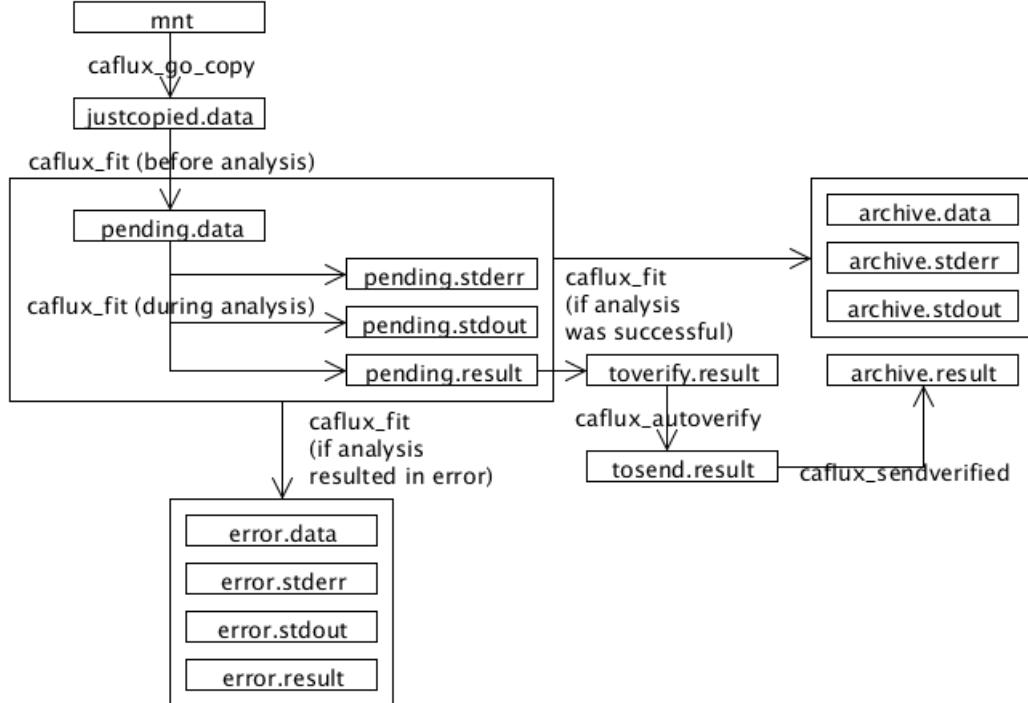


Figure 3.28: Data flow between components in Caflux.

Caflux is designed for moderate usage. This means that the following maintenance steps should be performed at least monthly:

- Perform an analysis by uploading arbitrary measurement data and wait for the result. If no result arrives, find the cause for the error.
- Check if there are any analyses that resulted in error (`error.*` folders). If this is the case, find out the cause of the error. Errors can be caused by bad measurements (such as too few events), communication problems (file corruption during upload) or bugs in Caflux.
- Check the disk space on the computer where Caflux is running. If there is a lack of disk space, delete or archive gated data in the `archive.data` folder. It is recommended to have 500 GB of free disk space continuously on the computer where the analyses run.
- Check if there are any partially uploaded measurements in the FTP storage (these should have a file extension `.gz`, not `.gz.completed`) that have been there for a long time, ie. they are not being uploaded currently. If this is the case, delete them.
- Check the application log `caflux-stdout` for errors especially for records of problems with sending emails (`SENDVERIFIED`).

Additional tasks of the administrator can include:

- If a FacsKin user is not able to upload the gated data, she can save it to her computer with FacsKin, and send the gated file to the administrator. By renaming the file to `yyyyMMddHHmmss.email_address.gz` format and simply copying it to folder `justcopied.data`, the analysis of the file will be performed just as if it had been uploaded.
- If a FacsKin user complains about not receiving the email containing a specific kinetics file, the administrator can look up that file in the `archive.result` folder and send it manually to the user.

3.4.3 Uninstallation

To uninstall Caflux:

1. Optionally archive measurement and analysis data from `/home/caflux/caflux`.

2. Delete user `caflux` using the `userdel` command and delete the home folder (`/home/caflux`).
3. Delete the line from `/etc/rc.local` copied during the installation (line starting with `su caflux -c "nohup /home/caflux/caflux/start_analysis.sh"`).

Folder names	Contents
<code>archive.*</code>	Archived measurements, results and outputs.
<code>error.*</code>	Measurements (and outputs) the analysis of which resulted in an error.
<code>examples</code>	Test files used by <code>test-64.R</code> .
<code>justcopied.data</code>	Gated files waiting for analysis.
<code>pending.*</code>	Measurements, outputs and results during analysis.
<code>toverify.result</code>	Kinetics files waiting for verification by <code>caflux_autoverify</code> .
<code>tosend.result</code>	Kinetics files waiting for sending by <code>caflux_sendverified</code> .
<code>toattach.result</code>	Temporary folder used by <code>caflux_sendverified</code> .
<code>mnt</code>	The FTP storage is mounted here by <code>caflux_go_copy</code> .

Table 3.1: Components of Caflux (directories in folder `caflux`).

Filenames	Purpose
<code>caflux-64.R,</code> <code>fct-64.R, dlogistx.so</code>	The core of Caflux, the programs executing the calculations.
<code>create_folders.sh,</code> <code>rc.local, dlogistx.c</code>	Files only needed for installation (purpose described there).
<code>passwords</code>	Contains authentication data for the FTP and SMTP servers.
<code>caflux_go_copy</code>	Shell script that copies the files from the FTP server to folder <code>justcopied.data</code> .
<code>caflux_fit</code>	Shell script that looks for newly copied gated data in folder <code>justcopied.data</code> and executes the analysis by calling <code>caflux-64.R</code> . During the analysis, the <code>pending.*</code> folders contain the gated file (<code>pending.data</code>), the standard output and standard error (<code>pending.stdout, pending.stderr</code>) and the produced kinetics file (<code>pending.result</code>). If the analysis ended successfully, the kinetics file is moved to folder <code>toreview.result</code> and all the other files belonging to the analysis (in folders <code>pending.*</code>) are moved to the corresponding <code>archive.*</code> folders. If the analysis resulted in an error, all files belonging to the analysis are moved to the corresponding <code>errors.*</code> folder. This script runs in 8 instances by default (determined by <code>start_analysis.sh</code>) in order to speed up analysis.
<code>caflux_autoverify</code>	Moves files found in <code>toreview.result</code> to <code>tosend.result</code> . This script could be modified in the future to run verification / further analysis on kinetics files. Verification could also be done by hand by deleting this file.
<code>caflux_sendverified</code>	This script sends the kinetics files found in <code>tosend.result</code> to the user initiating the analysis (the email address of the user is contained in the metadata section of the kinetics (and corresponding gated) file) and moves them to <code>archive.result</code> .
<code>sendEmail</code>	This script is used by <code>caflux_sendverified</code> to send emails.
<code>start_analysis.sh,</code> <code>stop_analysis.sh</code>	Scripts for starting and stopping analysis. The number of <code>caflux_fit</code> processes can be adjusted in <code>start_analysis.sh</code> (it is recommended to set the number of processes to twice the number of processor cores).
<code>caflux-stdout</code>	Application log, all scripts' standard output is redirected here by the command copied to <code>/etc/rc.local</code> .

Table 3.2: Components of Caflux (files in folder caflux).

Chapter 4

Developer Documentation

4.1 Structure

The software is built up of two main components:

- FacsKin, which is a GUI-based Java program the user interacts with. It implements Sections 2.1.1 and 2.1.3 of the specification. (There is a website for the easy distribution and launch of FacsKin at <http://www.facskin.com>.)
- Caflux, which is a text-based service-like application which implements Section 2.1.2 of the specification. Caflux is put together from R [27] and bash scripts.

The separation of these two components has the following benefits:

- Phase two can be quite resource-intensive, and a dedicated hardware can speed up the calculations.
- Allows the improvement of the algorithm for the second phase and re-analysis of previous measurements centrally, if needed.
- Allows easier implementation (as R provides easy-to-use tools for statistical analysis, data visualization etc) and more possibilities of experimentation.
- Allows the registration and assessment of need for such calculations, the popularity of the program can be measured.

The data flow between the components is shown in Figure 4.1.

The communication protocols between components are the following:

- The user can upload gated files for Caflux to the FTP connection. The exact format of a gated file is specified in Section 2.1.1.2. The gated file is compressed using GZIP and renamed to `yyyyMMddHHmmss.email_address.gz` where the

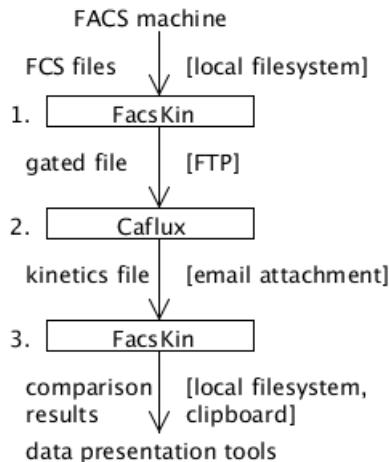


Figure 4.1: FacsKin, Caflux and communication between them. The communication protocol is written in squared brackets.

timezone is that of Budapest and the email address is specified by the user before clicking the Upload button in FacsKin. All characters not belonging to [a-zA-Z0-9.-@] should be replaced by a _ character in the email (the filename is only for easy identification of filenames for the administrator of Caflux, the exact email address of the user is a metadata field in the gated file). After the upload successfully finished an additional .completed extension should be added to the filename to indicate to Caflux, that this file can be downloaded from the FTP storage.

- The output of FacsKin is a kinetics file (extension .kinetics) the format of which is specified in Section 2.1.2.2. This file is sent by Caflux to the email address specified in the metadata section of the corresponding gated file as an attachment.
- Comparison results can be exported from FacsKin by means specified in Section 3.2.7.1

Failure handling:

- If the FTP upload does not succeed, the user is presented with an error message and is able to save the gated file to her computer and send it manually to the administrator of Caflux.
- If the FTP upload is interrupted, the filename will not contain the additional extension .completed, and hence it will not be downloaded and analyzed by Caflux.

- If the gated file is corrupted during FTP upload, it will only be noticed during analysis and probably results in an error (and the results and error messages end up in folders `error.*` on the computer running Caflux).
- If sending the email does not succeed, Caflux retries after 120 seconds. If the email address was specified incorrectly in the gated file, the email gets lost.

The rest of Developer's Documentation is separated to a section dealing with FacsKin (4.2) and Caflux (4.3).

4.2 FacsKin

FacsKin is a GUI-based program implementing sections 2.1.1 and 2.1.3 of the specification.

4.2.1 System Plan

FacsKin is implemented in Java and contains several packages.

- User interface packages:
 - `facskin`: contains the main application (`class FacsKinApp`) and the class implementing the main window (`class FacsKinView`) along with some helper classes and interfaces.
 - `facskin.fcs`: user interface for opening, gating and uploading FCS files.
 - `facskin.kinetics`: user interface for opening, grouping, pairing kinetics files.
 - `facskin.compare`: user interface for statistical comparison of data (`class ComparePanel`, `class MedianComparePanel`).
 - `facskin.graph`: classes implementing plotting mathematical data, functions, histograms, box plots. These classes are used throughout the other interface packages.
- Container package:
 - `facskin.formats`: contains classes for storage and modification of FCS data, gated data, kinetics and comparison data.
- Application logic packages:
 - `facskin.io`: classes implementing input/output for FCS, gated and kinetics files.

- `facskin.math`: classes implementing mathematical and statistical methods used by the application.

4.2.1.1 User Interface

The static class diagram of FacsKin showing only classes that are important from the point of view of user interface design are shown in Figure 4.2. `FacsKinView` has a menu, a tab pane and a progress bar, the tab pane can contain tabs of the following types: `FcsPanel`, `KineticsPanel`, `ComparePanel`, `MedianComparePanel`. Each header of a panel has a title and a close button. The kinetics panel can only be opened once (it can hold unlimited number of kinetics files, `FacsKinView`'s `kineticsPanel` field stores a reference to the kinetics panel or `null` if no kinetics tab is opened), the other panels can have unlimited number of instances.

The design of an FCS panel (together with the structure of `FacsKinView`) is shown in Figure 4.3. The design of the kinetics panel is shown in Figure 4.4 and a comparison panel is shown in Figure 4.5.

The contents of the menu bar are the following:

- File
 - Open (opening an FCS file brings up a new FCS panel, opening a kinetics file adds one row to the kinetics panel, opening a zip file containing n kinetics files adds n rows to the kinetics panel)
 - Append FCS (should be only active when there is an FCS file already opened)
 - Save As (should be only active when the kinetics panel is selected and there is at least 1 kinetics file)
 - Exit
- Preferences
 - Standardization (checkbox)
 - Draw function only in measurement timeframe (checkbox)
 - Use `,`/`.` as decimal separator (toggle)
 - Set maximum time for AUC
 - Set number of breaks in histogram plot
- Help
 - User's Guide

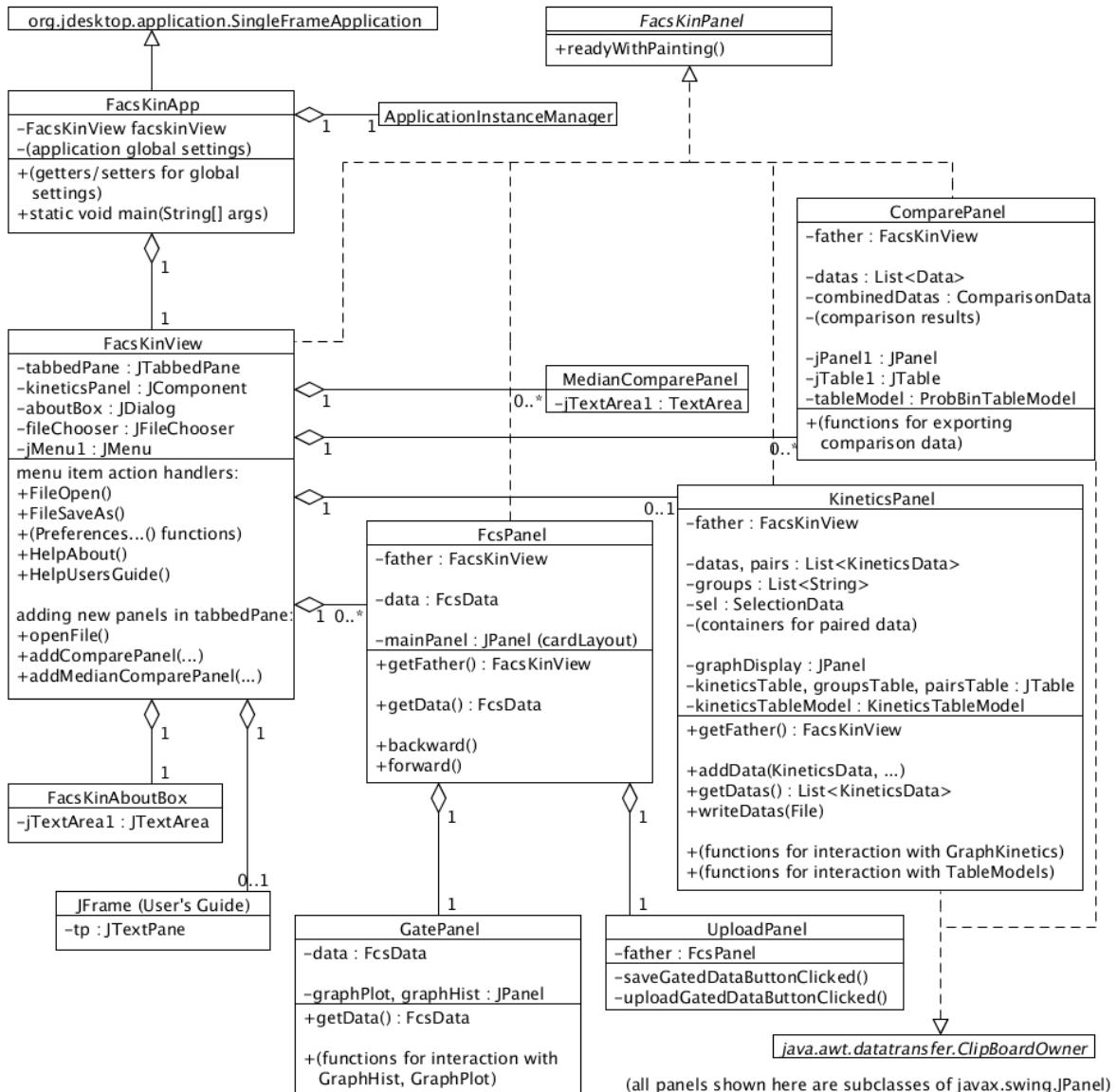


Figure 4.2: Class diagram of FacsKin showing classes important from the perspective of user interface. FacsKinView contains a tabbed pane which can hold the other panels as shown on the diagram. Fields and methods of the panels are grouped in the following way: 1. pointers to parent panels, 2. container types for data that the panel is holding, 3. pointers to child panels. Dashed line corresponds to implementing an interface.

- About: shows an about box (**class FacsKinAboutBox**)

The options reached by the Preferences menu are stored globally by **class FacsKinApp**. The about box contains general information about the program, contact information and a version history. The User's Guide is a HTML version (using **JTextPane**) of Section 3.2.

FacsKin is a single frame application, **FacsKinApp** extends

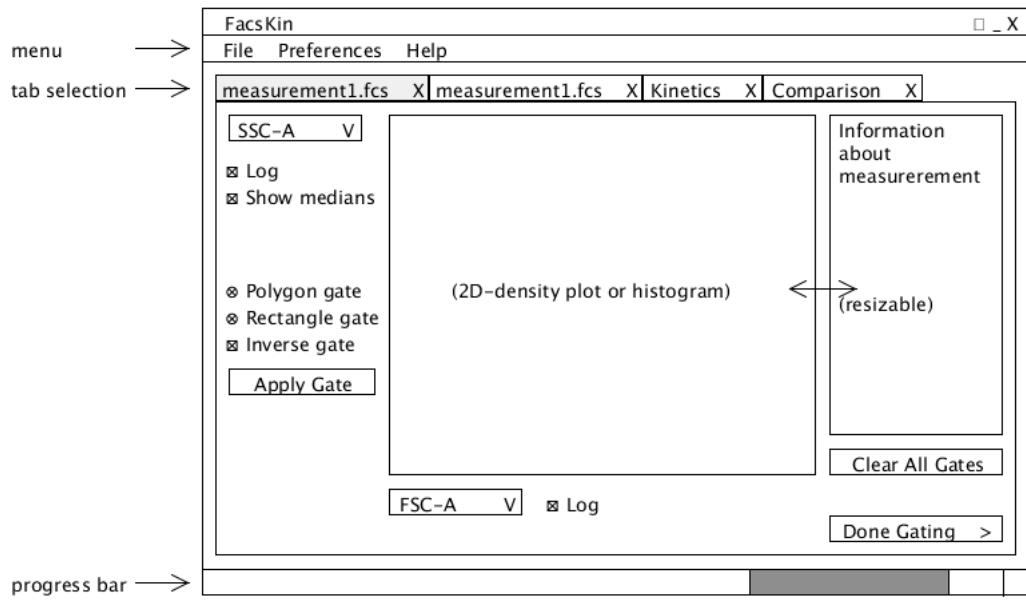


Figure 4.3: UI design of FacsKin: gating an FCS file.

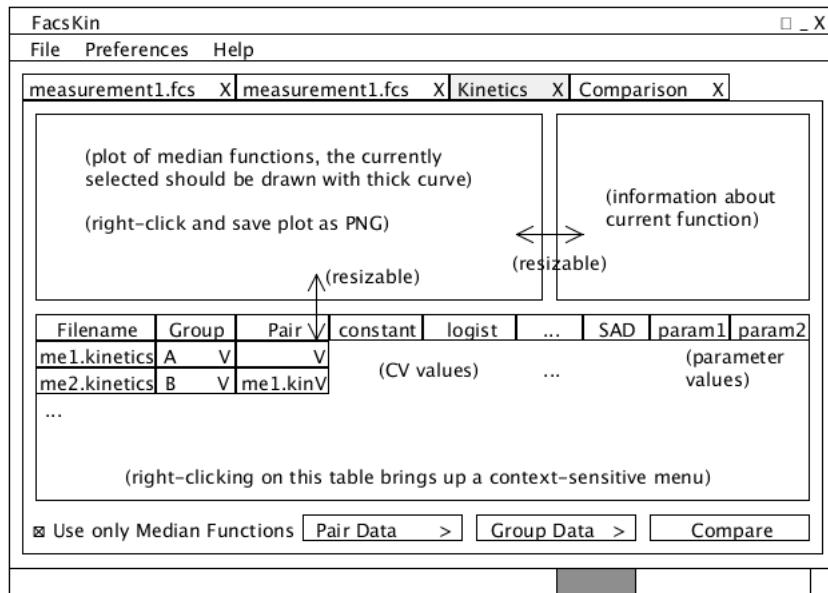


Figure 4.4: UI design of FacsKin: viewing kinetics files.

`org.jdesktop.application.SingleFrameApplication` and discovers new instances using class `ApplicationInstanceManager` and implementing the `ApplicationInstanceListener` interface. This method was described in [28]. The essence of the method is creating a dedicated socket for the application when first starting the application, and when a new instance of the application is launched, it first tries to connect to this socket, and when it is successful (and a shared

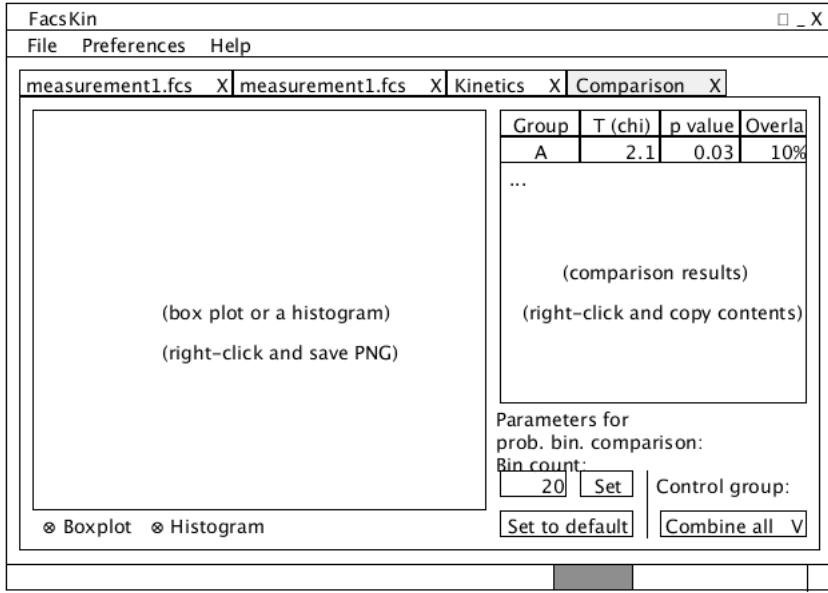


Figure 4.5: UI design of FacsKin: comparison panel.

key is matched), it notifies the first instance and passes the arguments to the new instance. FacsKin can be launched using the command line and accepts the command line parameter `-open`. If this parameter is given, later parameters will be regarded as file paths to open (this is in compliance with the Java Web Start specification [21]).

`KineticsPanel` and `ComparePanel` are subclasses of `java.awt.datatransfer.ClipBoardOwner`, so they can copy comparison data on the clipboard. Right-clicking the table in `KineticsPanel` brings up a context-sensitive floating menu having the following options:

- Get info: brings up a message box and prints all metadata contained in the kinetics file in the current row. Only active when clicking the first column.
- Remove row: only active when clicking the first column.
- Copy parameter data: copies all parameter data in the corresponding row (if clicked on first column) or the corresponding column (if clicked on one of the parameter columns). The data will be copied in tab-separated, Excel-readable format.
- Copy table contents: copies the full table contents to clipboard in tab-separated, Excel-readable format.
- Copy all data as R code: creates R code which initializes variables containing the same data as in the table and copies this R code to clipboard.

Clicking on the header of the table has the following effect:

- Clicking one of the function names selects that function as current function (and updates the parameters and the plot).
- Clicking one of the parameter names selects that parameter (the column background becomes blue) and enables the Comparison button.

Clicking one of the table cells has the following effect:

- Filename column: selects the kinetics file in that row (row background becomes blue, the corresponding curve in the plot becomes thick)
- Group column: one can select a group for the kinetics file from one of the current groups or create a new group in a drop-down menu. Clicking “(new)” brings up a string input dialog for the name of the new group.
- Pair column: one can select another kinetics file as a pair from a drop-down menu.
- One of the parameter columns: selects that parameter (the column background becomes blue) and enables the Comparison button.

The contents and states of tables are specified by table models, see Section 4.2.1.2.

Right clicking the plots in `KineticsPanel` and `ComparePanel` brings up a floating menu with a menu item “Save as PNG”.

`ComparePanel` contains a table the contents of which can be copied to clipboard in Excel-readable format by right-clicking the table and selecting “copy table contents”.

`MedianComparePanel` contains a text panel the content of which can be exported by Ctrl-C. It contains the comparison results (see 2.1.3) and R code for performing the same computations (which can be used as a verification).

```
class TaskWithProgress      in      package      facskin      extends
org.jdesktop.application.Task which is used to implement time-consuming
calculations while keeping the User Interface responsive. All time-consuming
tasks in FacsKin are subclasses of TaskWithProgress and call the super
constructor with the currently running application (FacsKinApp) as a parameter.
TaskWithProgress has a method setProgressValue(float) which sets the
progress of a task between 0 and 1. FacsKinView contains a task monitor
targetting the application which shows and updates the progress bar by imple-
menting a java.beans.PropertyChangeListener object. The following tasks are
implemented by this method in FacsKin:
```

- class FacsKinView: class OpenTask
- class FacsKinView: class FileAppendFCSTask
- class GatePanel: class ClearAllGatesButtonClickedTask
- class GatePanel: class ApplyGateButtonClickedTask
- class UploadPanel: class SaveGatedDataButtonClickedTask
- class UploadPanel: class UploadGatedDataButtonClickedTask

The state diagram of the user interface is shown in Figure 4.6.

4.2.1.2 Table models

`KineticsPanel` can contain three different kind of tables and the configuration settings `getShowDataByGroups()` and `getShowDataByPairs()` in `class FacsKin` specify the table currently shown. Table 4.1 shows which of the three tables is shown in each case. See also the `KineticsPanel` part in Figure 4.6.

getShowDataByGroups()	getShowDataByPairs()	table shown	comparison
false	false	kinetics table	allowed*
false	true	pairs table	not allowed
true	false	groups table	allowed
true	true	groups table	allowed

Table 4.1: The possible state of tables in the kinetics panel as given by the global settings `getShowDataByGroups()` and `getShowDataByPairs()`. * If “Use only median functions” not checked.

These tables are implemented using the model-view design pattern. The three table models each corresponding to one table:

- `KineticsTableModel` shows all kinetics files in different rows and allows selecting the group and the pair for each kinetics file. It is interactive and the data that it shows is stored in fields of `class KineticsPanel` (i.e. `datas`, `groups`, `pairs`).
- `PairsTableModel` underlies the table shown after pairing kinetics files. One row corresponds to two kinetics files, an original one and its pair. The data that it shows is stored in fields of `KineticsPanel`, namely `pairsValues`, `pairsNames`, `pairsGroups`.
- `GroupTableModel` is used when grouping kinetics files (either directly after using `KineticsTableModel` or after pairing (`PairsTableModel`)). The data it contains is generated when clicking “Group Data” button in local variables of

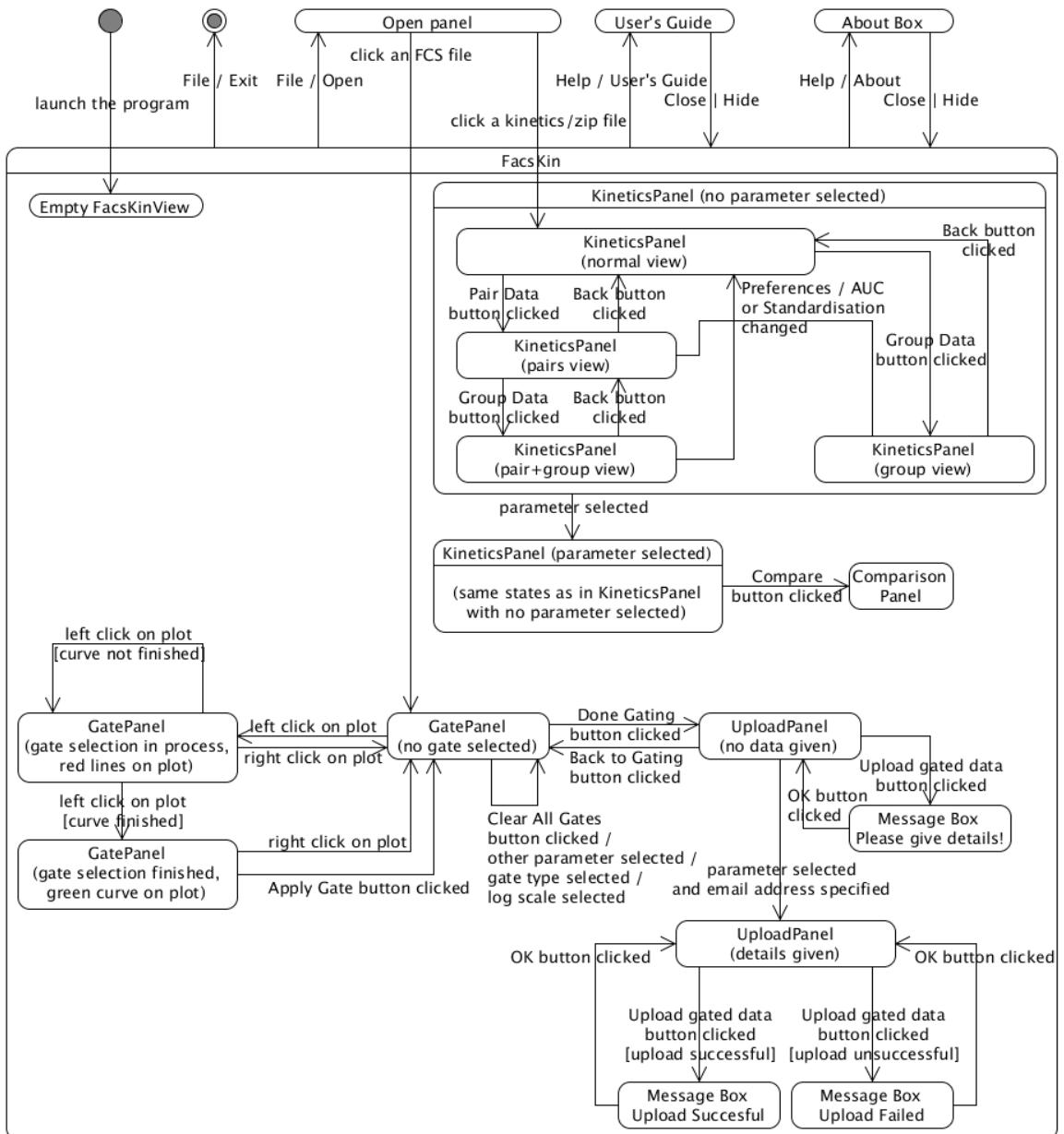


Figure 4.6: State diagram of FacsKin User Interface.

function `groupDataButtonClicked()` in class `KineticsPanel`, and is passed in the constructor of `GroupTableModel`.

These table models extend `javax.swing.table.AbstractTableModel` and override it's `get[Row|Column]Count()`, `[get|set]ValueAt()`, `getColumn[Name|Class]()`, `isCellEditable()` methods, but also implement the following interfaces to the data contained in them:

- `getContentsAsString() : String`: returns the contents of the table in tab-separated text table format as it is shown on the screen.

- `getDistributionsFor1ParameterAsString(int param) : String`: returns the distribution of the given parameter in a tab-separated text table format.
- `getDistributionsFor1DataAsString(int param) : String`: returns the distributions of the given data for all parameters in a tab-separated text table format.
- `getRCode() : String`: returns R code [27] the execution of which initializes variables containing the same data as in the table.

In `ComparePanel` `ProbBinTableModel` is used for the presentation of probability binning comparison results. All data that it presents is given as constructor parameter.

For every table the view is implemented using `javax.swing.JTable`.

4.2.1.3 Plots

Figure 4.7 shows the relation between the panels containing plots (`GatePanel`, `KineticsPanel`, `ComparePanel`) and the classes in package `facskin.graph`. The type of plots contained in that package are as follows:

- `GraphHist`: produce a histogram from one distribution (specific to `FcsData`).
- `GraphScatterPlot`: produce a two-dimensional density plot from a 2D-distribution (two parameters in an `FcsData`).
- `GraphKinetics`: produce a plot of a function and corresponding median values from one of the functions in a `KineticsData` object. The median values are plotted as squares with 3-pixel long side length.
- `GraphBoxPlot`: produce box-and-whisker plots [2] for a list of distributions (each box-plot in different colour).
- `GraphHistogramPlot`: produce histograms for a list of distributions (each histogram in different colour with the same limits). The number of breaks in the histogram plot can be controlled by the `FacSkinApp.setNumberofBreaksinHistogramPlot()` global setting.

`GatePanel` contains interactive plots, this is why these `Graphs` need mouse event handling and need their parent `GatePanel` in a private field. They use the functions `getData()`, `getSelectedParam()` etc. provided by `GatePanel` to plot data. `KineticsPanel`'s plot (`GraphKinetics`) is not interactive but changes while adding/removing/selecting different kinetics files or groups/pairs of kinetics files in the table. To update the plot, `KineticsPanel` calls the `reloadTransformer`

function in `GraphKinetics` which repaints the plot by querying the kinetics data provided by the functions `getSelectedFunction()`, `getSelectedKineticsData()`, `getGroupNumber()` etc.

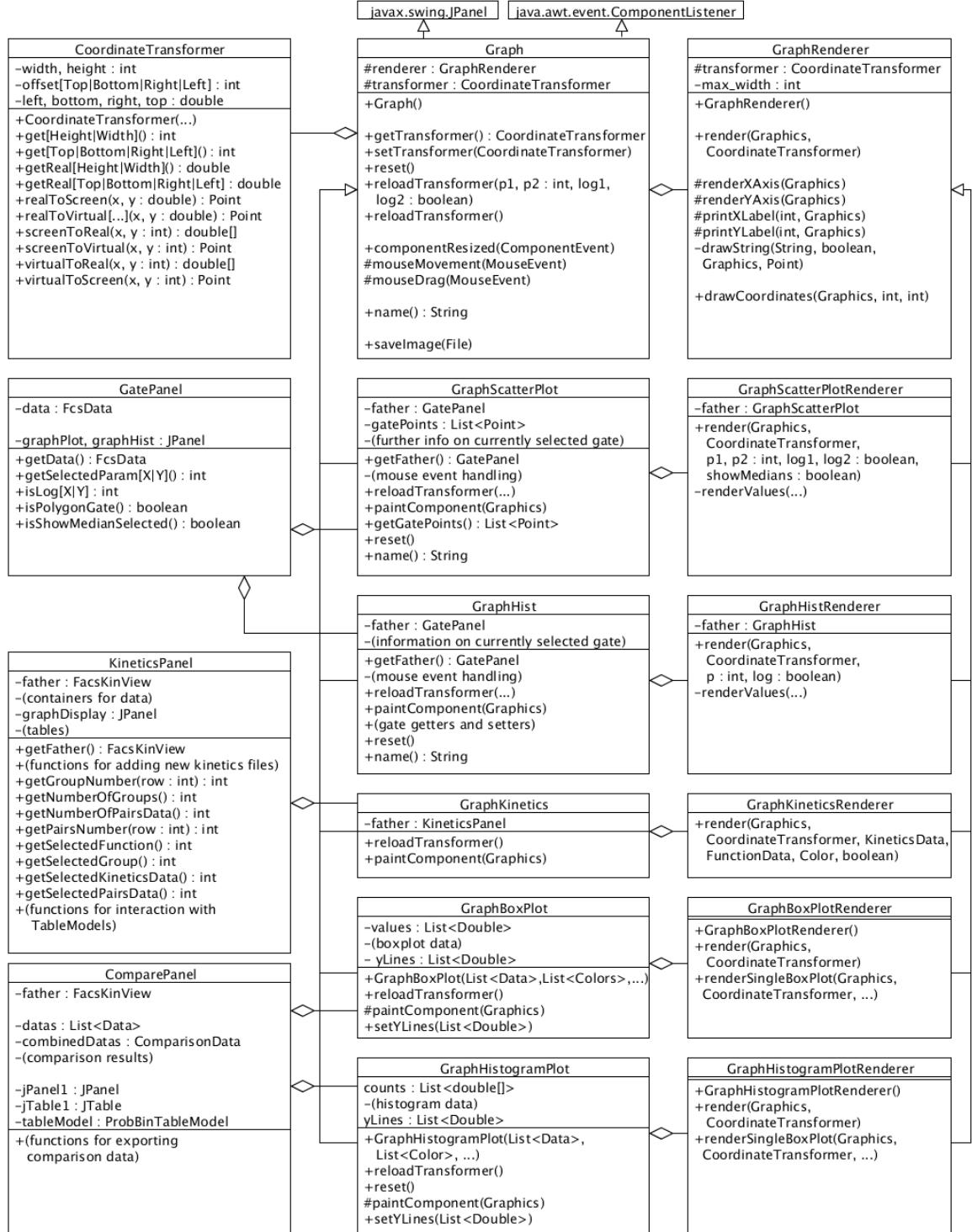


Figure 4.7: Class diagram of plots and how different panels use the classes in package facskin.graph.

The coordinate system of the screen in Java is different from that of the plots that we want to create. Distances on the screen are measured in pixels, not in arbitrary units and the y coordinates of the pixels increase from top to bottom. To allow the `Graph*` classes plotting seamlessly using real mathematical values and also enable drawing strings or bullets on the plot using pixel values (so that a bullet would be always recognizable regardless of the pixels pro real values ratio) we implement class `CoordinateTransformer` for the conversion between the following units (see the relation of the coordinate systems in Figure 4.8):

- real values (conventional mathematical coordinate system)
- virtual values (pixels, y coordinates increase from bottom to top)
- screen values (pixels, y coordinates increase from top to bottom)

The size of frame around the plot is fixed in pixels (`offset[Right|Left|Top|Bottom]`) and the origo of the real coordinate system can be placed anywhere by setting `real[Right|Left|Top|Bottom]` values in the constructor.

4.2.1.4 Data Containers

The container classes are grouped together in package `facsSkin.formats`.

Classes storing an **FCS file** or **gated data** are shown in Figure 4.9. `FcsData` contains some metadata (an array of strings) and a table containing events as rows and columns as parameters (`data`). To speed up generating plots while gating, caches for histograms and scatter plots are also kept there (these are stored in the helper classes `HistogramList` and `PlotList`). There are functions for appending another `FcsData` (`append(FcsData)`) and for creating gates (`[inverse] [Hist|Plot]Gate(...)`). We keep a copy of the original FCS data in `firstData` so that all gates can be undone by overwriting `data` with `firstData`. This is performed by `clearGates(...)`. `FcsData` closely represents the gated data format as described in Section 2.1.1.2.

The runtime representation of a **kinetics file** (specification in Section 2.1.2.2) is class `KineticsData`. A kinetics file contains 5 functions (`FunctionData`) and each function contains a number of parameter distributions (`ParameterData`). That is, the parameter table stored in a kinetics file is stored by columns in memory (which is helpful for extracting comparison data). When standardizing (see 2.1.3) parameter data, the original data is not be modified but a multiplier is set from 1 to something else and the getters provide values multiplied by this standardizer. The standardizer values are stored at the level of `ParameterData` (field `standardizer`) while their exact value can be set at level `FunctionData` (method `setStandardizer`) and turning on and off standardization can be set at level `KineticsData`. `ParameterData`

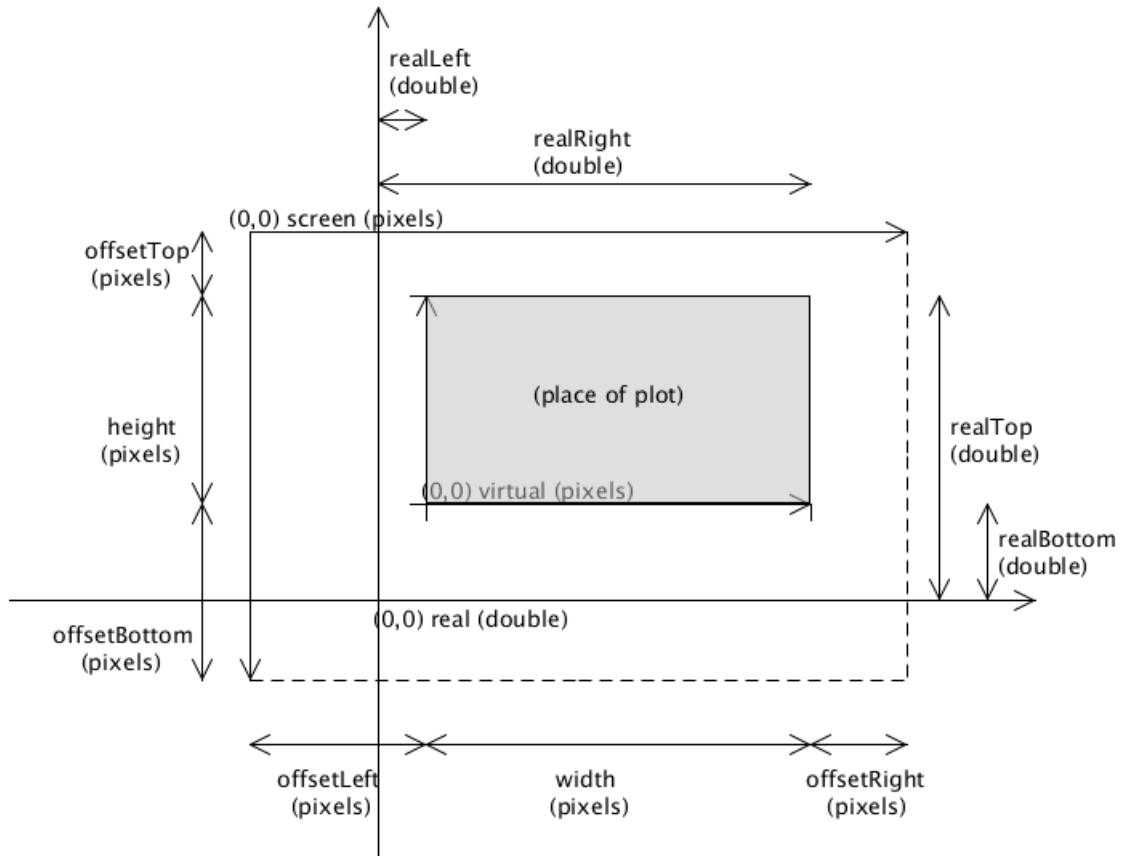


Figure 4.8: The three coordinate systems (screen, virtual and real) and the relationships of different fields in `CoordinateTransformer`. The origo point of each coordinate system is marked with (0,0).

is a dynamic container in sense of standardization but it does not allow the modification of parameter data since that is fixed in a kinetics file. However, during comparison distributions of parameters can be merged, paired (substracted) etc. so we have to provide a class that allows such modifications (but disallows) standardization. This is `ComparisonData` which, together with `ParameterData` is an implementation of `interface Data` that provides getters to access data contained in such an object. Plotting these values is helped by caching quantile values (methods `getQuantile(...)`). The relations between these classes and their corresponding fields are shown in Figure 4.10.

4.2.1.5 Input/Output

Input/Output classes are placed in package `facsSkin.io`.

Reading data from FCS files (2.1.1) is handled by `facsSkin.io.FcsOpener`. The constructor needs a `java.io.InputStream`

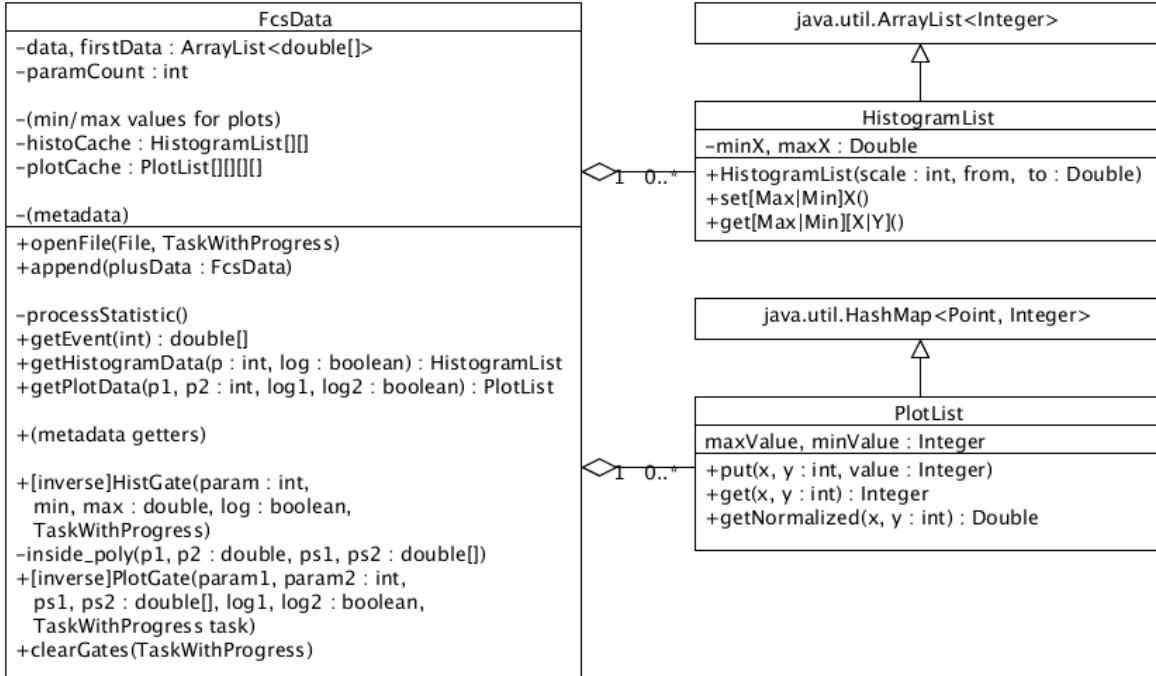


Figure 4.9: Class diagram of container classes for FCS data.

and the class provides the `getData() : ArrayList<double[]>` and `getKeywordValue(keyword : String) : String` functions for getting data and metadata as well as functions for getting parameter names etc.

`facskin.io.GatedDataWriter` writes data for one selected parameter in an `FcsData` to a `java.io.OutputStream` in the format specified in 2.1.1.2. It is used by `UploadPanel` during uploading data to the FTP storage and also when saving gated data as a single file.

`facskin.formats.KineticsData` contains a reader for the kinetics format (2.1.2.2). We also provide a writer for kinetics format (`KineticsDataWriter`). We allow saving multiple kinetics files into one zip so as to ease opening multiple files. Reading and writing such files is implemented in `KineticsZip`. `KineticsZip` has two constructors, one that accepts a file to read from (getters provide the data just read) and another that accepts the runtime representation of a list of kinetics datas (method `writeDatas(File)` saves datas to file). Some preferences are also stored in these zip files such as the group for each kinetics file (filename `__groups`, reader `GroupReader`), selected pairs (filename `__pairs`, reader `GroupReader` is used here as well because we use the same format), selected function (file `__selectedFunction`, reader `SelectedFunctionReader`) and maximum time for AUC (file `__maxTimeForAUC`, reader `MaxTimeForAUCReader`). The format that `GroupReader` accepts is the following: a simple text file containing even number of lines, the $(2 * i + 1)^{th}$ line being the filename of a kinetics file and the $(2 * i + 2)^{th}$

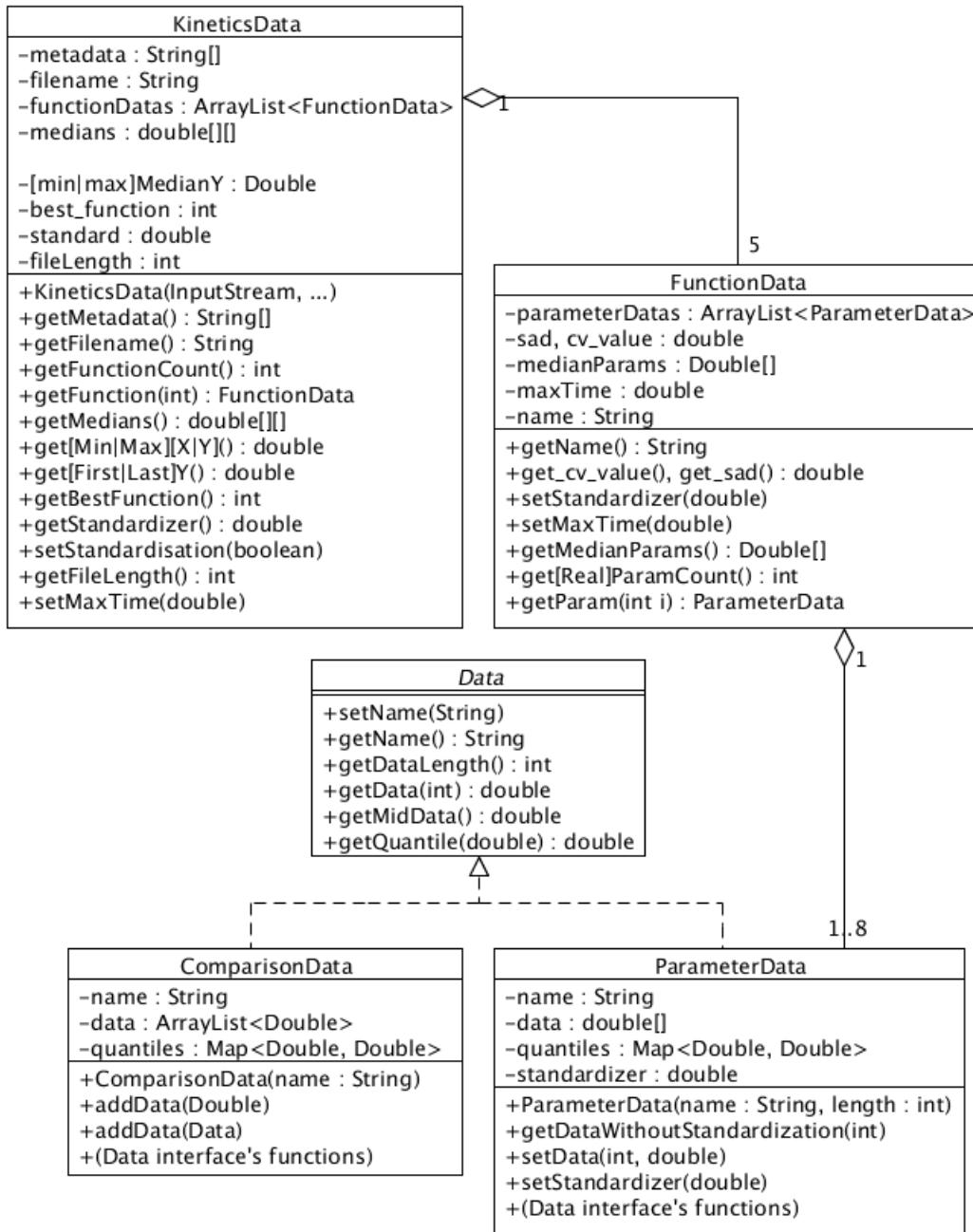


Figure 4.10: Class diagram for containers of kinetics data.

line is it's group (in case of pairing this latter is the filename of it's pair or empty if no pair was set).

Figure 4.11 show the usage of IO classes.

Other kind of outputs FacsKin provides are cut and paste methods by several User Interface classes as described in 4.2.1.1.

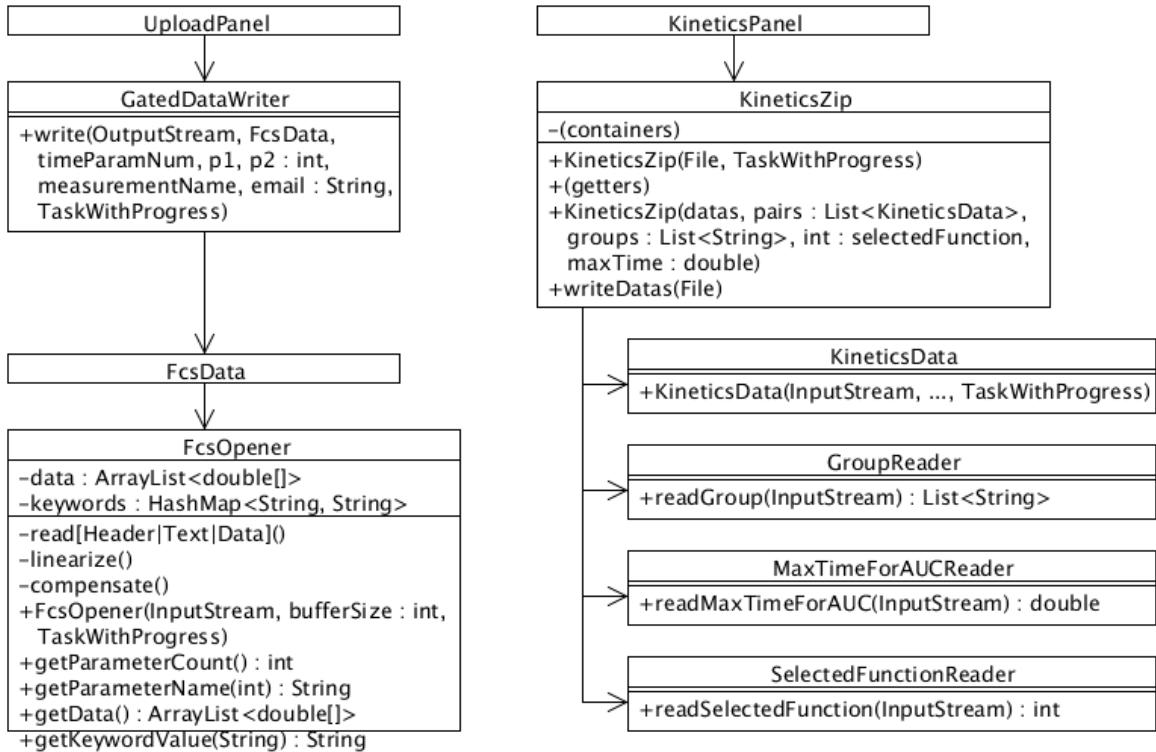


Figure 4.11: Class diagram for input/output classes.

4.2.1.6 Math classes

Package `facskin.math` contains classes providing mathematical and statistical definitions and algorithms needed for FacsKin. These classes mostly contain static functions.

- **class Functions:** provides static definitions of the 5 functions listed in 2.1.2.3 and some helper methods:
 - `constant()`, `logistp()`, `logistn()`, `dlogistp()`, `dlogistn()`: the functions as-is
 - `fct()`: reach the functions by name
 - `getFunctionName()`: reach the functions by id (which ranges from 0 to 4)
 - `paramCount()`: query the number of parameters of a function
 - `getParameterName(fct : [int|String], param : int)`: name of parameter of a function
 - `paramsToStandardize()`: query the parameters which need standardization

- `getAUC(fct : String, pars : double[], tmax : double)`: calculate parameter AUC
- `isParameterOutside()`: a method which calculates whether a parameter is outside a given range (this is used as a feedback to the user to assess overfitting of functions, see Section 3.2.6.1, part “How to choose the best function”).

This class is used in the table models in package `facskin.kinetics`, in `KineticsPanel`, `FunctionData` and `GraphKineticsRenderer`.

- `class KruskalWallis` implements Kruskal-Wallis rank-sum test [3] by providing a static function `test(List<List<Double>>) : double[]`. The return value is an array giving the statistic and the p-value (in this order). This class is used by `MedianComparePanel`.
- `class MannWhitney` implements Mann-Whitney U test (or Wilcoxon test) which is a more specific version of Kruskal-Wallis rank-sum test. The interface is similar to that of Kruskal-Wallis: `test(p1, p2 : List<Double>) : double[]`. This class is only used for test purposes.
- `class ProbabilityBinning` implements PB comparison [11]. It provides static functions (`defaultBinCount()`) for calculating a suggested bin count for a list of `Data`s or `KineticsData`s. The suggested default bin count is the $1/10^{th}$ of the number of values in the smallest group. PB works as follows: a control group (list of `double` values) and a bin count are specified (parameters of constructor `ProbabilityBinning(control : Data, binCount : int)`), and we divide the range of values in the control group into the specified number of bins, each bin having the same amount of values. (Because the number of values is not necessarily divisible by the number of bins, there may be a slight variation of number of values pro bin, hence we store the number of control values for each bin in field `control_counts : double[]`.) We can compare a list of values with the control group by counting the number of values for each bin and calculating a chi-squared statistic by the following formula: $\chi^2 = \sum_{i \text{ is a bin}} \frac{(control_Relcounts_i - test_Relcounts_i)^2}{control_Relcounts_i + test_Relcounts_i}$. According to [11], the $T = \max(0, \frac{\chi^2 - B/E}{\sqrt{B/E}})$ statistic has standard normal distribution (B is the bin count, $E = \min(\text{size of control}, \text{size of test})$). Function `compare(Data) : double[]` returns the T value and the associated p-value. We also provide a function `getLimits : double[]` returning the limits of the bins (this is used by `MedianComparePanel` when plotting). This class is used

by `ComparePanel` and by `KineticsPanel`, the latter when pairing kinetics files if “Use only median functions” is not selected.

- `class Statistics` implements some very basic statistical functions:

- `median(List<Double>) : double`
- `quantile(List<double>, q : double) : double`
- `overlap(highs, lows : double[]) : double`: calculates the relative size of overlap among intervals given by their low (`lows`) and high limits (`highs`): $\min_i \frac{\min(\text{highs}_i) - \min(\text{lows}_i)}{\text{highs}_i - \text{lows}_i}$.

This class is used by `[Median]ComparePanel` and `ParameterData`.

- `class MyDecimalFormat` implements methods for formatting decimal numbers so that they match the global configuration `FacsKinApp.getDecimalSeparator()` and ensure a nice short output (`format()`, `formatShort()`) or do not lose any decimals which is helpful when extracting numeric data to other programs (`formatLong()`, `formatLongPoint`). If needed for the nice presentation, numbers are outputted in exponential format.

4.2.1.7 Error handling

Errors are detected either:

- before they arise by careful handling of state (checking valid ranges of a variable, disabling buttons, automatically moving to another state).
- after they occur by catching exceptions (usually popping up an error message) and changing to a valid state / reverting to a valid state.

In the Graphical User Interface, some errors are avoided by **disabling user interface elements** that do not make sense:

- “File/Append FCS” menu item is only enabled when an FCS file is already opened for gating. It is disabled when the active tab is not an `FcsPanel` (`[dis|en]ableAppendFCS()` function in `FacsKinView`).
- “File/Save as” menu item is only active when the kinetics tab is active and there is at least one kinetics file in the table (`[dis|en]ableSaveAs()` function in `FacsKinView`).
- `FcsPanel`:

- “Apply Gate” button is only enabled when a gate is selected ([`dis|en`ableGateButton() function in `GatePanel`]).
- “Polygon/Rectangular gate” selection, “Log” and “Show medians” check-boxes on the y axis are only enabled when using scatter plot.
- “Done gating” button is only enabled when `GatePanel` is active, “Back to gating” button is only enabled when `UploadPanel` is shown and an upload is not currently in progress ([`dis|en`ableBackward() functions in `FcsPanel`]).

- `KineticsPanel`:

- “Compare” button is only active when a parameter is selected (and not in paired mode) and when “Use Only Median Functions” is selected, only in grouped mode.
- “Pair Data” button is only active when measurements are not grouped
The text of buttons are changed according to the states as shown in Figure 4.6.

Disabling a button is not always the best idea, because it does not give any user feedback why that particular UI component cannot be used. We use a **message box** (`showMessageDialog()` function in `javax.swing.JOptionPane`) in 31 cases in `FacsKin` giving a detailed error description and hints on how to correct the error:

- `FacsKinView`:

- When the preference settings are not numeric or outside a range (≥ 1 for maximum time for AUC, ≥ 0 and ≤ 100 for number of breaks in histogram plot).
- File read errors (specific error messages for FCS files containing no time parameter).

- `UploadPanel`:

- When clicking the “Upload gated data” button without having at least 200 events or without having selected a parameter, entered a name for the analysis and an email address. The same holds for “Save gated data” button.
- Different messages and hints for different kind of network errors.
- Error messages when saving gated data not does not succeed.

- `KineticsPanel`:

- When trying to compare using Kruskal-Wallis test without having selected at least two groups.
 - When trying to save the plot but there is a file write error.
- `ComparePanel`:
 - When trying to set a bin count outside the range (bin count should be between 2 and the third of the size of the smallest group).
 - When trying to save the plot but there is a file write error.

In `KineticsPanel`, when table data has to be recalculated because the maximum time for AUC has been changed or standardization has been toggled, the grouped/paired data can be corrupted so in such cases the **state is changed** back to non-paired and non-grouped mode automatically as shown in Figure 4.6 to avoid data corruption.

4.2.2 Implementation

4.2.2.1 Version History

Facskin was implemented using NetBeans 6.9.1 [24] on NixOS [25] and Ubuntu [30] operating systems between 15 April 2009 and 29 April 2012. The version history can be viewed in the About box. The current version described in this document is Version 0.5.13.

Facskin has been first implemented for internal usage of the FACS Laboratory at Semmelweis University, Budapest, but later has been used by other laboratories throughout the world. The analysis algorithm has changed several times during implementation (together with the kinetics format to reflect changes in the algorithm) and several user interface improvements have been added as well. Changes in the user interface between different versions are shown in Figures 4.12, 4.13, 4.14. The overall structure of the program has remained the same in compliance with the specification (Section 2.1).

Main changes in the program structure included:

- Creating data containers for kinetics data, comparison data.
- Adding support for more than two groups (version 0.4.13), thus creating one more table and corresponding table model in package `facskin.kinetics` and replacing Mann-Whitney U test with Kruskal-Wallis rank sum test.
- Adding support for pairing kinetics datas (version 0.5.9), thus creating one more table and corresponding table model in package `facskin.kinetics`.

- Implementing probability binning comparison (version 0.4.17).
- Adding `org.apache.commons.math` as a library dependency to provide functions of approximated chi-squared and normal distributions.
- Standardization has been once turned off (version 0.3.1), then turned on (0.4) and then made configurable (version 0.4.4).
- The decision was made that `facskin.formats.FcsData` should contain the in-memory copy of the fcs file read, so that gates can be cancelled in an easy way. (The other solution could have been to re-read the original FCS file when clicking “Clear all gates”.)
- The way the program can be launched changed several times by adding the possibility to open multiple files, a command line interface and a Java Web Start interface.

My overall impression of the NetBeans programming environment has been very positive, the function completion, instant function documentation feature and the “Find usages” command have been especially helpful.

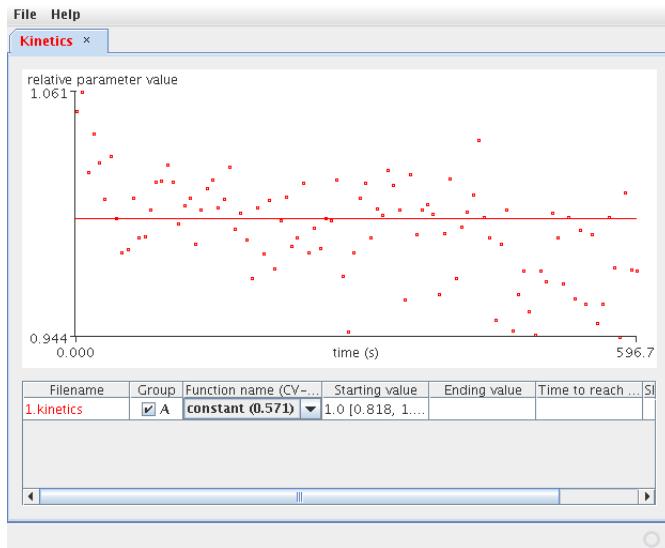


Figure 4.12: Screenshot of FacsKin version 0.1-3. Note that there is only a checkbox for the group (hence, only two groups are possible) and a drop-down list for functions.

4.2.2.2 Further Development

FacsKin could be improved in several ways:

- The gating phase (Section 2.1.1) could allow cancelling gates individually by storing the gates in symbolic expressions. FCS format 3.1 [16] has support for

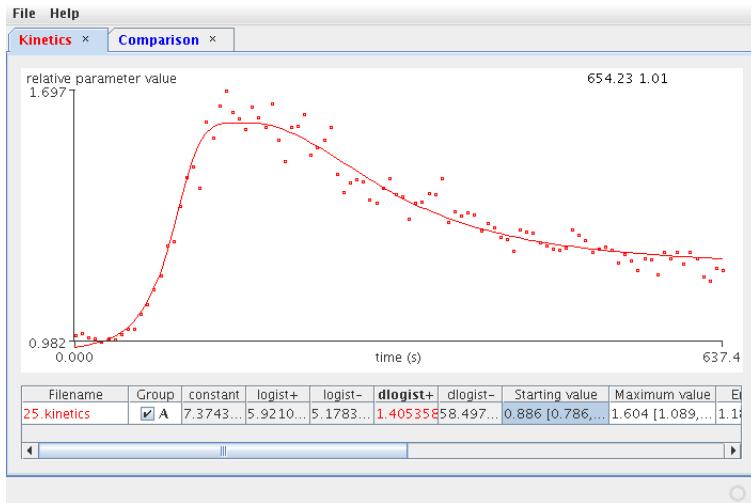


Figure 4.13: Screenshot of FacsKin version 0.3.1. Note that each function has a separate column.

storing the gates, hence the storage of such files would be possible without creating a new format. Currently FacsKin cannot save FCS files.

- The analysis phase (creation of kinetics file, Section 2.1.2) could be integrated into FacsKin which would speed up and ease the usage of the program.
- FacsKin could be integrated into another Flow Cytometry analysis program which already has useful features such as creating multiple gates, trees of gates, one-parametric analysis, grouping of FCS files etc.

4.2.2.3 Interesting Algorithms

Interesting algorithms used by FacsKin include:

- Scatter plot polygon gate in `facskin.formats.FcsData` uses the so-called crossing-count algorithm to detect whether a point is inside a polygon. The (x, y) point is inside the polygon represented by the $xs_i, ys_i, i \in \mathbb{Z}_n$ values iff the following value is odd:

$$\sum_{i=0}^n \chi \left((ys_i < y \leq ys_{i+1} \vee ys_{i+1} < y \leq ys_i) \wedge \frac{y - ys_i}{x - xs_i} < \frac{ys_{i+1} - ys_i}{xs_{i+1} - xs_i} \right) \text{ where } \chi(true) = 1 \text{ and } \chi(false) = 0$$

`point1` and `point2` correspond to x and y and `points1` and `points2` correspond to xs and ys in the function `inside_poly`. To count whether the

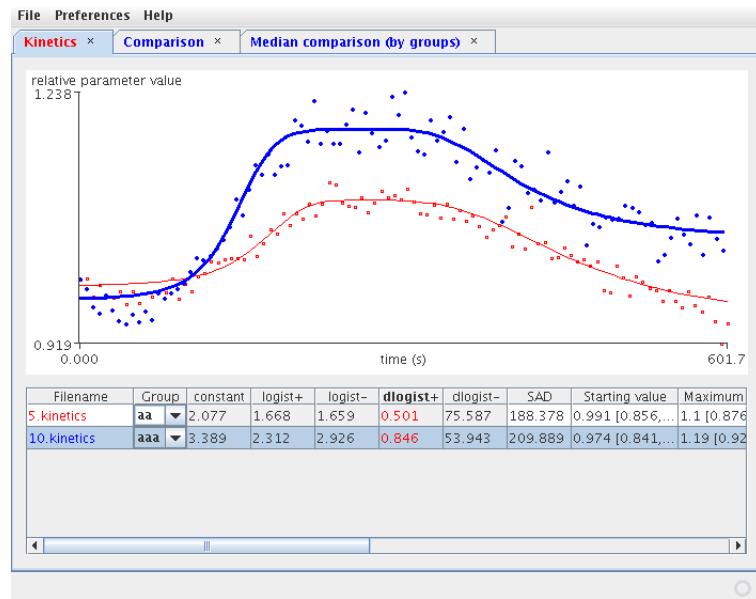


Figure 4.14: Screenshot of FacsKin version 0.4.13. Note that there is a drop-down list for groups but still no pairing and no comparison buttons in the bottom.

number is odd, we simply negate a boolean variable every time when the above expression inside χ is true.

- Kruskal-Wallis rank sum test in `facskin.math.KruskalWallis`. This was implemented following the R function `kruskal.test` (see the comments in the source code). We used the chi-squared distribution provided by library `org.apache.commons.math`.
- Mann-Whitney U test in `facskin.math.MannWhitney` has been implemented as follows:

The values in the two groups are $x_i, y_j, i \in \{1, \dots, m\}, j \in \{1, \dots, n\}$.

$$U = \max(U', m * n - U') \text{ where}$$

$$U' = \sum_{i=0}^m \sum_{j=0}^n f(x_i, y_j)$$

$$f(a, b) = \begin{cases} 0.5 & a = b \\ 1 & a > b \\ 0 & a < b \end{cases}$$

$$p = \min(p', 1) \text{ where}$$

$$p' = \begin{cases} pw(m * n - U, \min(m, n), \max(m, n)) * 2 & \max(m, n) < 10 \\ pstnorm(\frac{u - m * n / 2}{\sqrt{m * n * (m + n + 1) / 12}}) * 2 & \text{otherwise} \end{cases} \text{ where}$$

$$pw(q, m, n) = \sum_{i=0}^q \frac{cw(i, m, n)}{\binom{m+n}{n}} \text{ where}$$

$$cw(k, m, n) = \begin{cases} 0 & k < 0 \vee k > m * n \\ 0 & m = 0 \wedge n = 0 \wedge k > 0 \\ 1 & m = 0 \wedge n = 0 \wedge k = 0 \\ cw(k - n, m - 1, n) + cw(k, m, n - 1) & \text{otherwise} \end{cases}$$

$$pstnorm(z) = \frac{1}{\sqrt{2 * \pi}} * \int_z^{\infty} e^{-\frac{x^2}{2}} dx$$

The procedure returns U and p . Function $cw()$ and the calculation of combinations is implemented using memoization (this is why the fields are static in class `MannWhitney`) and integration is implemented using a simple numerical integration method using trapezoidal rule.

- The implementation of probability binning comparison in `facskin.math.ProbabilityBinning` is described in Section 4.2.1.6.

4.2.2.4 Compilation and Distribution

The source code and compilation utilities for FacsKin in folder `FacsKin` are the following:

- folder `src`: Java source code of FacsKin organized in packages as described in Section 4.2.1.
- folder `test`: test packages.

- folder **nbproject**: NetBeans project configuration files.
- folder **lib**: libraries used by FacsKin. These are the following:
 - `AbsoluteLayout.jar`, `appframework-1.0.3.jar`,
`swing-worker-1.1.jar`: packages needed for the Swing GUI framework.
 - `commons-math-2.2.jar`: package needed for statistical functions (Section 4.2.2.3).
 - `commons-net-ftp-2.0.jar`: package providing FTP upload support (used by class `UploadPanel` in package `facskin.fcs`).
- folder **dist**: the distribution folder, `FacsKin.jar` and a copy of the libraries are placed here after compilation.
- folder **build**: folder containing objects files during build etc.
- folder **examples**: folder containing example and test kinetics and fcs files (see Section 4.2.3), temporary files are created here during running the test suite.
- **build.xml**: build configuration file (with instructions on how to perform build).
- **FacsKin.bat**, **FacsKin.sh**: application launchers for Windows and Linux.
- **keystore**: storage for private keys for signing FacsKin (this is required when distributing applications with Java Web Start). This file was generated with the following command:
`keytool -genkey -alias facskin -keystore keystore -validity 36500`. It is referred to in `nbproject/project.properties`.
- **FacsKin.php**: FacsKin Java Web Start launcher (JNLP). We create a specific version of this file for new versions of FacsKin because we distribute FacsKin in a `FacsKin-VERSION.jar` file where `VERSION` is the current version number.
- **package.sh**: a shell script helping the distribution of FacsKin. It executes the following steps:
 1. Grep for the version number and release date in `src/facskin/resources/FacsKinApp.properties` (editing the field values in the About box places these data here).
 2. Create a zip bundle for FacsKin (see Section 3.2.2).
 3. Mount remote filesystem.
 4. Copy the zip bundle to the server.

5. Rename `FacsKin.jar` to `FacsKin-VERSION.jar`, apply the version number in the JNLP file (`FacsKin.php`) and copy the JNLP bundle to the server.
6. Extract the documentation (corresponding to Section 3.2 in this document) from the source code (`src/facskin/resources/usersguide.html`), update links in it and copy it to the server.
7. Delete temporary files and unmount remote filesystem.

To compile and distribute FacsKin, you need to do the following:

1. Specify the login information for the FTP storage for gated files (`src/facskin/fcs/UploadPanel.java`, fields `server`, `user`, `password`) and for the server used for the distribution (`package.sh`, fields `FTPUSER`, `FTPPASSWD`, `FTPPDOMAIN`).
2. Execute the following commands from the command line in folder `FacsKin`:

```
ant jar      # compilation
./package.sh # distribution
```

4.2.3 Testing

4.2.3.1 Tests using the GUI

The following black-box tests should be performed with the User Interface each time before distributing a new release of FacsKin. They test almost every part of the GUI and basic properties of the main algorithms.

- Program launch and file handling:
 - Launch FacsKin, “File/Open” `examples/1.kinetics`. “File/Open” `examples/5.kinetics`. Both should be shown in the same kinetics tab. Exit FacsKin.
 - Launch FacsKin, “File/Open”, select `examples/1.kinetics` and `examples/5.kinetics` in the File Open panel and click “Open”. Both kinetics files should be opened. Exit FacsKin.
 - Launch FacsKin. “File/Open” `examples/1.kinetics`, “File/Open” `examples/baseline.fcs`, in the kinetics tab, click column “Starting value” and click button “Comparison” so that we have 3 tabs. Check

wether the “File/Save as” menu item is only active when the kinetics tab is active and the “File/Append FCS” menu item is only active when the FCS tab is active. Right click the first column in the table of the kinetics tab and select “Remove row”. When there are no kinetics files in the kinetics tab, the “File/Save as” menu item should be inactive. It should stay inactive when we switch to another tab and back. Exit FacsKin.

- Linux: test the command line interface (in folder FacsKin):
`cd examples; FacsKin.sh 1.kinetics 5.kinetics baseline.fcs.`
 Two kinetics files should be shown in the kinetics tab and there should be a tab named “baseline.fcs”. Exit FacsKin.
- Test multiple launches:
- Windows: test filetype association and multiple instances. After launching FacsKin in Java Web Start, double-click a `1.kinetics` in folder `examples`. It should appear in the kinetics tab of the already launched instance.
- Test zip file opening: open `examples/cf.zip`. Modify some groups, pairs, delete a row, modify “Preferences/Set Maximum time for AUC”, modify the currently selected function, click “File/Save As” and save as a different filename. Exit FacsKin, launch FacsKin and open the previously saved zip file and check whether the kinetics files, groups, pairs, maximum time for AUC and selected function are the ones you specified.

- Gating:

- Launch FacsKin. “File/Open” `examples/baseline.fcs`, “File/Append FCS” (30 seconds) `examples/reaction.fcs` and check whether there is a 30 second gap by selecting “Time” on x axis and any other parameter on the y axis. Exit FacsKin.
- Launch FacsKin. “File/Open” `examples/test.fcs`, the event count should be 13013 and the measurement date “01-JUN-2006 15:06:40”. FSC-SSC polygon gate the rectangular area at the bottom. The remaining event count should be 7444. Clear all gates (13013 event should appear). Inverse polygon gate the same range. 5569 events should remain. Clear all gates. Rectangular gate and rectangular inverse gate the same range, and the same number of events should remain as previously. Clear all gates.
- Repeat the same process with logarithmized axes (3 possibilities: x: log, y: lin; x: lin, y: log; x: log, y: log). The numbers should remain the same.

- Clear all gates. Select “Time” on axis x and “Histogram” on axis y. Gate so that the first three and last three columns remain outside. 6925 events should remain. Clear all gates. Apply the same gate with “Inverse gate” checked. 6088 events should remain.
- Clear all gates. Gate so that the last three columns remain outside. 10303 events should remain. Clear all gates. Select logarithmic scale on axis x, and repeat the previous gate. The same amount of events should remain.
- Clear all gates. FSC-SSC polygon gate the rectangular area at the bottom. Select “APC-A” and logarithmic scale on axis x and “Histogram” on axis y. Perform a gate so that only the first big column remains outside (gate range between values 0.2 and 2.9). 2688 events should remain.
- Select a gate so that 0 events remain. The plot should become gray and all the buttons, controls disabled. If you click “Clear all gates”, the controls should become enabled again and the plot should show scatter plot.

- Upload:

- Launch FacsKin. “File/Open” `examples/baseline.fcs`, “File/Append FCS” (30 seconds) `examples/reaction.fcs`. Click “Done gating” and click “Upload gated data” and “Save gated data”. You should not be allowed to upload/save until you have selected a parameter, specified a name and an email address. If you remove one of these, you should not be allowed to do upload/save.
- Launch FacsKin. “File/Open” `examples/baseline.fcs`, gate so that less than 200 events remain. Click “Done gating” and try to upload data. You should not be allowed to upload or save gated data unless you have at least 200 events.

- KineticsPanel:

- Launch FacsKin. “File/Open” `examples/cf.zip`, click the header in each column corresponding to a function and check that the parameter columns change so that each function has the parameters as specified in Section 3.2.6.1. Exit FacsKin.
- Launch FacsKin. “File/Open” `examples/cf.zip`, select function “constant” and change the “Preferences/Set Maximum Time for AUC” to 100. Check that the AUC value in “CF01 Control...” row becomes 119.332. Set maximum time to 200, the AUC value should double, i.e. to 238.664. Exit FacsKin.

- Launch FacsKin. “File/Open” `examples/cf.zip`, remove the first row, notice that in row “CF01 test” the value in column “Pair” disappears. Exit FacsKin.
- Launch FacsKin. “File/Open” `examples/cf.zip`, click “Pair Data” and check whether 5 rows appear with filenames “CF0* test...” where * ranges from 1 to 5. Click “Group Data”, note that 3 rows appear, “A”, “B” and “C” with sizes 2, 2, 1, correspondingly (3 different colours). Check whether selecting a row selects the corresponding curves in the plot. Click “Back (no grouping)”, click “Back (no pairing)” and click “Group Data”. Note that 3 rows appear, “A”, “B” and “C” with sizes 804, 804, 402, correspondingly (3 different colours). Check whether selecting a row selects the corresponding curves in the plot. Exit FacsKin.
- Launch FacsKin. “File/Open” `examples/cf.zip`, check “Use only median functions”, click “Pair Data” and check whether 5 rows appear with filenames “CF0* test...” where * ranges from 1 to 5. Click “Group Data”, note that 3 rows appear, “A”, “B” and “C” with sizes 2, 2, 1, correspondingly (3 different colours). Check whether selecting a row selects the corresponding curves in the plot. Click “Back (no grouping)”, click “Back (no pairing)” and click “Group Data”. Note that 3 rows appear, “A”, “B” and “C” with sizes 4, 4, 2, correspondingly (3 different colours). Check whether selecting a row selects the corresponding curves in the plot. Exit FacsKin.
- Launch FacsKin. “File/Open” `5.kinetics` and `25.kinetics`, toggle “Preferences/Standardization” and check whether the parameters and plots change accordingly. Toggle “Draw function only in measurement timeframe” and check whether the plot changes accordingly. Toggle “Use ,/. as decimal separator” and check whether the presentation of numeric values in the table change accordingly.

- Comparison:

- Launch FacsKin. “File/Open” `examples/5.kinetics` and `10.kinetics`. Click Column “Time to reach maximum” and click “Compare”. In the appearing comparison panel, check that the plot contains two box and whiskers diagrams and the median and quartile values on the diagrams are the same as the ones specified in the table in the kinetics tab. Switch to Histogram and check whether it corresponds to the boxplot. In the lower right corner select “5.kinetics” as control group and check that the “T(chi)” value in the row “5.kinetics” is 0.0, the p-value is 0.5 and the

overlap is 100%. Check whether the “T(chi)” values in the bottom of the box plot are the same as in the table. Select “10.kinetics” in the lower right corner and check that the values change the same way in the corresponding row. Set the bin count to different values and note how the gray lines change correspondingly in the plot.

- Perform the same steps but after deselecting “Preferences/Standardization”. Create a different group for one of the measurements, click “Group Data” and click “Compare”. The result should be the same as before.
- Change “Preferences/Set number of breaks in histogram plot” to 20 and check whether it has effect on plots created after selecting this.
- Launch FacsKin. “File/Open” `examples/cf.zip`, click “Pair Data”, click “Group Data” click column “Starting value” and click “Compare”. Copy and execute the R code in an R environment and check whether the chi square, degrees of freedom and p-values are the same as given by FacsKin.
- Launch FacsKin. “File/Open” `examples/cf.zip`, select “Use only median functions”, click “Group Data” click column “Starting value” and click “Compare”. Copy and execute the R code in an R environment and check whether the chi square, degrees of freedom and p-values are the same as given by FacsKin.

- Data export:

- Launch FacsKin. “File/Open” `examples/cf.zip`, right-click the plot, select “Save as PNG” and verify that the file is the same as the plot. Select a row in the table, save the plot as PNG and verify that the file is the same as the plot (showing a function selected). Repeat the same after pairing data, grouping data and pairing+grouping data.
- Launch FacsKin. “File/Open” `examples/cf.zip`, right-click the table, select “Copy table contents”. Paste the contents of the table into a text editor and verify that they are the same. Check “Use only median functions” and repeat the same.
- Launch FacsKin. “File/Open” `examples/cf.zip`, right-click a column in the table, select “Copy parameter data”. Paste the contents into a spreadsheet program (such as LibreOffice [23]) and calculate the median and quartiles (functions `median(CELLRANGE)`, `percentile(CELLRANGE, 0.25)`, `percentile(CELLRANGE, 0.75)`) for each row and verify that these are the same as that in the table.

- Launch FacsKin. “File/Open” `examples/cf.zip`, check “Use only median functions”, right-click a column in the table, select “Copy parameter data”. Paste the contents into a text editor and verify that they are the same.
- Launch FacsKin. “File/Open” `examples/cf.zip`, right-click the first column in a row of a kinetics file in the table, select “Copy parameter data”. Paste the contents into a spreadsheet program (such as LibreOffice [23]) and calculate the median and quartiles (functions `median(CELLRANGE)`, `percentile(CELLRANGE, 0.25)`, `percentile(CELLRANGE, 0.75)`) for each row and verify that these are the same as that in the table.
- Launch FacsKin. “File/Open” `examples/cf.zip`, check “Use only median functions”, right-click the first column in a row of a kinetics file in the table, select “Copy parameter data”. Paste the contents into a text editor and verify that they are the same.
- Launch FacsKin. “File/Open” `examples/1.kinetics` and `5.kinetics`, right-click the table and select “Copy all data as R code”. Paste the data into a new R environment and execute it. Verify that the following two command reproduce the data in the table:

```

lapply(params[["1.kinetics"]], function(pars) {
  print(paste(median(pars), quantile(pars, 0.25),
              quantile(pars, 0.75)))
})
lapply(params[["5.kinetics"]], function(pars) {
  print(paste(median(pars), quantile(pars, 0.25),
              quantile(pars, 0.75)))
})

```

Check whether the variable `fct` has value "dlogist+" and metadata has the following value:

	<code>name</code>	<code>size</code>
1	<code>1.kinetics</code>	201
2	<code>5.kinetics</code>	201

- Launch FacsKin. “File/Open” `examples/cf.zip`, select a parameter, click “Compare”, right click the table appearing and select “copy table contents”. Paste the contents of the table into a text editor and verify that the contents are the same. Right-click the plot and select “Save as PNG”, verify that the file is the same as the plot. Change to histogram view, save the plot as a PNG and verify that it looks the same.

- The user interface should feel responsive during all the previous steps.

4.2.3.2 Unit tests

The unit tests for FacsKin are contained in the `test` folder:

- `facskin.io.FcsOpener`: tests reading different FCS formats, esp. FCS 2, FCS 3, files with logarithmized parameters, FCS 3 files with spillover matrix. The `flowCore` R package [18] was used to verify the data. When implementing support for new FCS formats, these tests should ensure that support for old files does not break.
- `facskin.io.KineticsDataWriterTest`: tests whether reading a kinetics file, writing it to another file and reading it back results in the same data. The comparison for `KineticsData` is implemented in the helper class `KineticsComparison`.
- `facskin.io.KineticsZipTest`: tests whether reading a zip file (which contains kinetics files and metadata for groups, max. time for AUC etc.), writing it to another file and reading it back results in the same data. The comparison for individual `KineticsDatas` is implemented in the helper class `KineticsComparison`.
- `facskin.formats.FcsDataTest`: tests reading FCS data, and also all kinds of gating, clearing gates, appending another FCS file, creation of histograms and scatter plots. Guessing the number of the time parameter from the metadata keywords in the FCS file is the job of this class, this is why several different types of FCS files are read in function `testGetTimeParamNumber()`.
- `facskin.formats.ComparisonDataTest`: tests both types of addition and query of comparison data.
- `facskin.math.KruskalWallisTest`: we test the implementation of Kruskal-Wallis rank sum test by static reference tests (`testTest()`) and by dynamically creating test data (`randomData()`) and R code for comparison. We execute the R code which compares the results provided by `KruskalWallis` and the R implementation.
- `facskin.math.MannWhitneyTest`: we compare the Kruskal-Wallis rank sum test when using two groups with the results of Mann-Whitney U test (they should be equivalent).

- `facskin.math.ProbabilityBinningTest`: we test the algorithm providing bins and limits for the control group (`testGetLimits_ControlCounts()`) and the comparison algorithm (`testCompare()`) separately.
- `facskin.math.StatisticsTest`: we test the public functions of class `Statistics`, namely `overlap()`, `median()` and `quantiles()` with different kinds of input.

4.2.3.3 Test results

When implementing the tests several flaws in the algorithms were discovered (and corrected). Some examples follow:

- Several UI bugs were discovered when following the testing algorithm as described in Section 4.2.3.1, such as problems with plots when there are no events during gating, enabling/disabling buttons and menu items was not always consequent, both radio buttons in one group could be deselected at the same time etc. (Version 0.5.12.)
- `FcsOpener` has transformed logarithmic values differently than `flowCore` [18]. The interpretation of the spillover matrix, hence implementation of compensation was wrong. These errors were corrected after comparing the data in different FCS files in `FacsKin` and in `flowCore`. These comparisons are now done in the unit tests. (Version 0.5.13.)
- The interface of classes was described more precisely after testing, such as the `getData(int i)` function in interface `Data` turned out not to return the data that was added i^{th} time, because data could have been rearranged inside. The interface only ensures that all data that was added should be returned for some i with `getData(i)` and nothing else should be returned. (Version 0.5.13.)
- Non-necessary functions were removed and algorithms were simplified. For example, the “make values unique” function was removed from class `ProbabilityBinning` and function `overlap(double[] highs, double[] lows)` was made private in class `Statistics`. (Version 0.5.13.)

`FacsKin` provided overall usable performance on an Intel Core Duo 1600 MHz notebook with 2.5 GB of RAM using NixOS [25] and JRE 1.6.0_32 [20], the graphical interface was responsible. Doing the GUI tests (Section 4.2.3.1) took about 20 minutes, running the unit tests (described in Section 4.2.3.2) took 98 seconds, and all tests passed.

4.3 Caflux

Caflux is a text-based program implementing Section 2.1.2 of the specification.

4.3.1 System Plan

Caflux is put together from an R script (`fct-64.R`) some helper C functions (`dlogistx.c`) and wrapper shell scripts providing an always-running failure-safe daemon-like program. The overall structure of Caflux is described in Section 3.4.2. Figure 3.28 shows how the different scripts work together and Tables 3.1 and 3.2 describe the purpose of every file distributed with Caflux.

The main design decisions for Caflux were the following:

- Separate the computer to which FacsKin uploads the measurements and the computer that performs the analysis. This is done for security purposes so that the machine which performs the critical analysis steps is not exposed directly on the internet.
- Use the R environment [27] for analysis which provides tools for fitting functions, transforming data, calculating quantiles etc.
- Use bash shell scripts to handle communication and separate them so that every script has it's own small job. This provides opportunity to parallelize them.

`fct-64.R` script is a collection of functions performing the analysis steps as described in Section 2.1.2.3. The data flow between functions is shown in Figure 4.15. The main function is `caflux.analysis()` which performs every step by calling other functions. Step 3 is done for all 5 functions, and data is collected in a list of function datas, one function data containing the name of the function, distribution of parameters of the function, SAD (Sum of Absolute Deviation) and CV (10-fold Cross Validation) values corresponding to the function. Fitting the functions is done by R's own optimization function, which needs starting values for each parameter. These starting values are provided by the `estimate.*.params()` functions.

4.3.2 Implementation

To install Caflux, as it is distributed in source-based form, you need all libraries that it uses, and you need to compile some parts (others run interpreted) as described there in Section 3.4.1. The following packages and libraries are used:

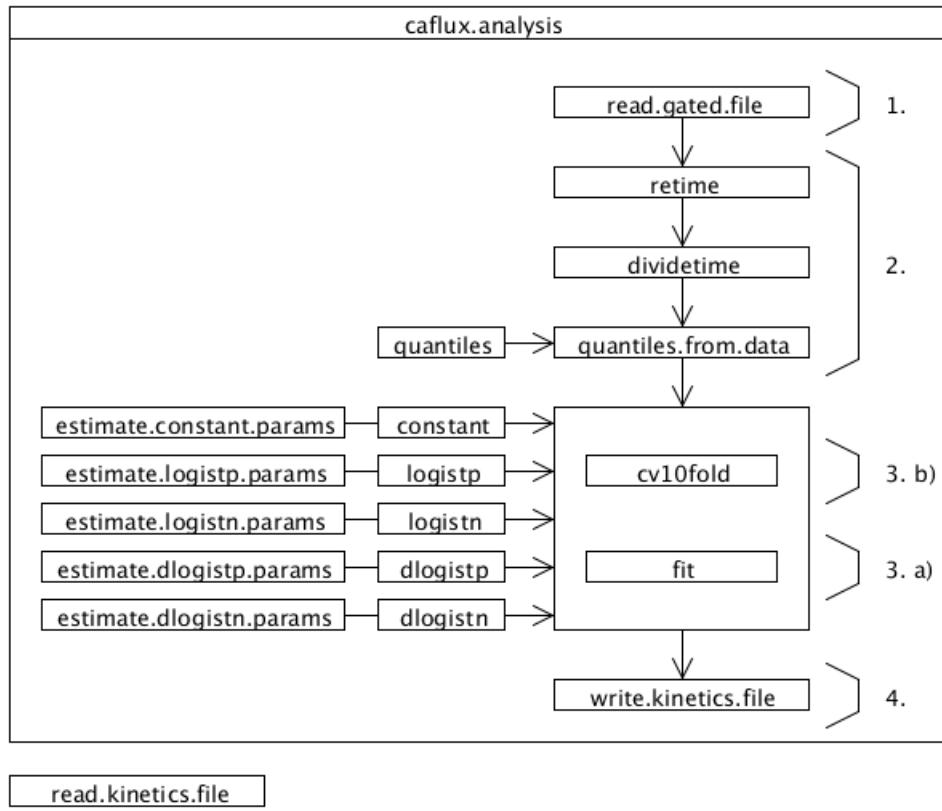


Figure 4.15: Data flow between functions in fct-64.R. Numbers show which function(s) implement which step of phase 2 of specification (Section 2.1.2.3).

- R (package `r-base` in Ubuntu): fitting functions is done using R, we use the following specific statistical functions: `optim()` (with methods “Nelder-Mead” and “SANN”), `lowess()`, `quantile()`, two-dimensional array-indexing functions which can seamlessly extract a row, a column or a specific set of rows/columns, `apply()`, `split()`, `sample()`; functions we use for debugging (creating diagrams): `plot()`, `lines()`; all other R functions used are for input-output or easy to implement using other languages.
- `libio-socket-ssl-perl` is needed by `sendMail` which we use to send kinetics files to the users.
- `curlftpfs` is used to access the FTP storage.

The first experiments with kinetic flow cytometry data were done using R, and when the analysis algorithm matured [4] an easy-to-use GUI was developed in Java around the main algorithm which was still implemented in R. Specifications for communication between the Java and R parts were made (gated and kinetics files) and the R calculations received a dedicated computer.

In `fct-64.R` the following global configurations are set:

- `fct.names`: a vector of strings (names of functions). Each corresponds to a function implementing the mathematical function itself and an estimator function. `fct.names` and the corresponding functions are not completely separate from the underlying algorithm because of efficiency reasons, that is, implementing a new function needs also modifications in `fit()`.
- `n`: the number of quantiles to fit a function to. This is set to 201 as given in the specification (so that the median and quartiles have an exact quantile, namely the 101th, 51th and 151th).
- `lambda`: during fitting, the sum of $(\text{abs}(\text{fittedValue} - \text{originalValue}))^{\text{lambda}}$ is minimized. The default value is 1, which results in minimizing the sum of absolute deviation.
- `baseline.part.if.fixed`: the implementation allows to fit a function with the first part fixed to the median of the first `baseline.part.if.fixed` seconds. To fit this way, the `fix.starting=TRUE` should be specified as a parameter for function `caflux.analysis()`.

The implementation matches the steps of the specification (Section 2.1.2.3):

1. Reading the gated file is done by function `read.gated.file()`.
2. Before dividing the timeframe into intervals (function `dividetime()`) and calculating quantiles (function `quantiles.from.data()`) we “retime” the events. This is necessary because some flow cytometers export values using very low time resolution: it can happen that 10,000 subsequent events have the same time value. If the number of unique time values in the measurement is not divisible by 100, it is possible that some time intervals have much more events than others. To prevent this, we make (almost) all time values unique by the following algorithm: if there are in average minimum 5 events with the same measurement time we recalculate the time of the consecutive events having same time value as if there were evenly distributed between their original time plus delta where delta is the smallest time difference that exists between arbitrary two events in the measurement.¹
3. Each mathematical function has an own implementation, `fct(x, p)`, where `x` is the time value (which can be given as a vector providing multiple time values) and `p` is a vector of the parameters. The parameter constraints are kept using negation of parameters if needed. If the constraint is $a > b$ where

¹A previous attempt was that delta should be the next time point, but in this case if there is a gap between data without any events, events would flow into this gap artificially.

a and b are two parameters and the function is called with parameters $a \leq b$, a and b are swapped inside the function. The real parameters (which are used internally) can be extracted from the function by giving `x=NULL` as the first parameter. If the function is called with `p=NULL`, it returns the number of its parameters. For efficiency reasons, calculating the values of `dlogistp` and `dlogistn` is implemented in C (`dlogistx.c`) and the R functions only act as a wrapper.

(a) We start the SAD-optimization algorithm from different points so that it could find the real minimum with higher chance. After each step, we compare the produced SAD with the overall minimum and if it is less, we store the parameter values (`mindevi` and `par`). We use the following starting points and algorithms:

- Estimate the function parameters with lowess parameter $f = 0.05$ and use Nelder-Mead optimization method [9].
- Estimate the function parameters with lowess parameters $f = 0.1$, $f = 0.15$, $f = 0.35$ and use Nelder-Mead optimization method.
- Estimate the function parameters with lowess parameter $f = 0.05$, $f = 0.1$, $f = 0.15$, $f = 0.2$, $f = 0.35$ and use SANN optimization method [1].
- Estimate the function parameters with lowess parameter $f = 0.2$ and multiply the slope parameters by 0.01, 0.1, 1, 10 and 100, and start the Nelder-Mead optimization method from each point (for functions with two slope parameters this means $5 * 5 = 25$ starting points).
- Finally, we use one Nelder-Mead, one SANN and three more Nelder-Mead optimization steps where the input of one step is the output of the previous step.

(b) 10-fold Cross Validation [10] is a method for testing the goodness of fit of a function to data by taking into account the phenomenon of overfit. A good introduction to the method is [7]. Function `cv10fold` implements it the following way: the input is n `x` and `y` values contained in `data`. We randomly rearrange the sequence $\langle 1, 2, \dots, n \rangle$ (named `s`) and we take the `x` and `y` values indexed by the first $n/10$ elements of `s` as the test set, the rest will be the training set. Then we take the `x` and `y` values indexed by the second $n/10$ elements of `s` as the test set, the rest as training set and so on. In each step we fit the function to the training set, and calculate the SAD value from the difference between the fitted function and the test set. We sum up these SAD values to get the overall 10-fold CV value.

4. Writing the kinetics file is done by function `write.kinetics.file()`.

Some functions have a parameter `plot` providing graphical debug output. Function `read.kinetics.file()` is also for debugging and testing reasons.

4.3.2.1 Further Development

The security of Caflux could be increased by implementing a verification tool for gated files (in C, for example) so that only real gated files are feeded into R. Security flaws in R could be easily exploited using the current implementation. It would further increase security to have all users upload with separate user names and passwords to the FTP storage.

Function `fit()` in `fct-64.R` is currently very slow, it could be optimized but only with strong testing guarantees so that a large pool of previously fitted test functions fit just as well (in terms of SAD) with the faster implementation.

4.3.3 Testing

4.3.3.1 Unit Tests for fct-64.R

Unit tests for the functions in `fct-64.R` are implemented in `test-64.R`. To perform the tests, issue the following commands:

```
time R --no-save <test-64.R
echo $?
gqview examples # look at the diagrams of fits
rm examples/test.*.png
```

If the second command gives 0, the tests ended successfully. If one of the tests failed, error messages are printed during the tests and the second command gives a non-zero value.

The following functions have separate test cases:

- `read.gated.file()`: we test reading reference data (which was created by FacsKin, hence we also test the output of FacsKin).
- `retime()`: we test retiming with simple time values having/not having repeated values.
- `cv10fold()`: we test whether the test and training sets are separated well.
- `fit()`: we fit functions to example datasets contained in `examples/test.*.csv`. We check whether the fit is good enough by specifying limits to the distribution of SADs and we also output the functions fitted in `examples/test.*.*.png` files.

- `read.kinetics.file()`, `write.kinetics.file()`: these are tested by the same function. We check whether reading a kinetics file, writing the same kinetics file out and reading it again results in the same kinetics data as the original one.

Results: `fct-64.R` passed all unit tests. Running the test suite on an Intel Core Duo 1600 MHz notebook with 2.5 GB of RAM using NixOS [25] took 12 minutes and 23 seconds.

4.3.3.2 Common tests for FacsKin + Caflux

Testing the whole software is done as a regular maintenance step by the administrator of Caflux (Section 3.4.2). However, before giving effect to any modification of Caflux the following test should be performed:

1. Launch FacsKin. For * in {1, 5, 10, 25}, “File/Open” `examples/*.baseline.fcs`, “File/Append” `*.PHA.fcs` with a 30 seconds gap, FSC-SSC gate for lymphocytes, histogram-gate for APC+ cells and upload it using the kinetic parameter ratio “FITC-A/PerCP-Cy5-5-A” and measurement name `*_test`.
2. Wait until the measurement results arrive by email.
3. Open the resulting kinetics files and also `examples/*.kinetics` (* in {1, 5, 10, 25}).
4. Pair kinetics files so that the tests are the ones with `*_test.kinetics` filename and the corresponding `*.kinetics` files are their pairs.
5. Perform pairing (without selecting “Use only median functions”). The starting value, maximum value, ending value and AUC parameters should have very low (close to 0) T(chi) values. The other parameters are not stable enough, hence their values T(chi) values can be higher.

This test was performed and the result was that maximum value, ending value and AUC parameters were 0, while starting value was 0.889 and 15.346 in cases of `1_test.kinetics` and `25_test.kinetics`, respectively.

Powering off the computer should not result in data loss during analysis, the interrupted analyses should automatically restart after the computer has been restarted. This was tested and it worked as expected.

4.3.3.3 Performance of Caflux

Between 27th January 2011 and 13th May 2012 Caflux analyzed 1055 measurements successfully on the same computer. 8 different `caflux_fit` processes were running in parallel and sometimes the machine stopped because of power failure. During this time 6 measurements ended up in the `errors.data` folder, out of which 5 were too small (the error was “Too many gaps in the experiment.” which means less than 50 time values) and 1 was not successfully uploaded (it was possibly corrupted during upload). Figure 4.16 shows the distribution of time for evaluating one measurement. The median value is 6.197 hours, that is $24/6.197 * 8 = 30.98$ measurements could have been evaluated a day when working with full load. In 50% of the cases the kinetics file was sent less than 4 hours after upload and in 90% of the cases this took less than 11.4 hours. The maximum response time was 69 hours (this probably corresponds to a power failure). There was no correlation between analysis times and event count which suggests that analysis time depends more on the shape of data than the size (see Figure 4.17; a Spearman rank-correlation test was performed between analysis times and event counts, without significant effect: p-value = 0.08184, $\rho = -0.05508103$. Only events with less than 19 hours of analysis time were considered in the test).

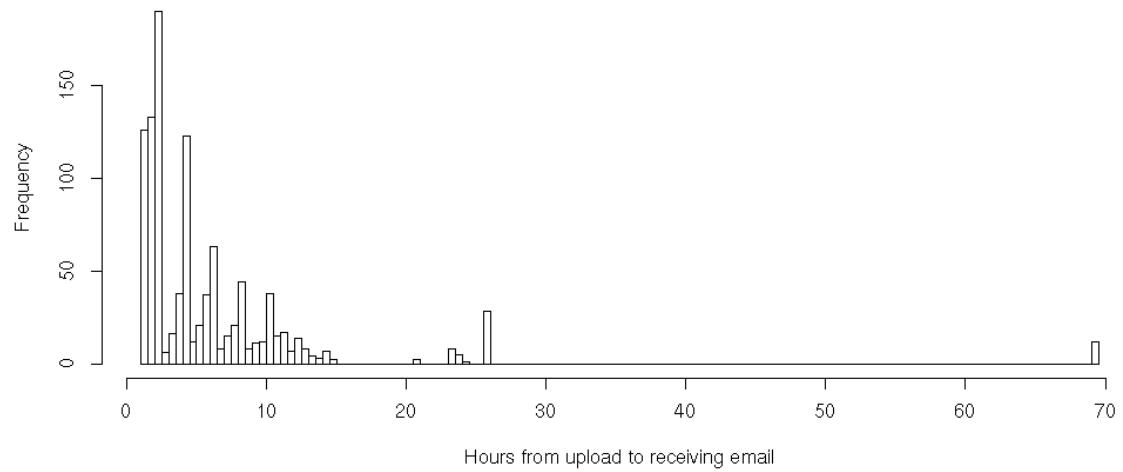


Figure 4.16: Histogram of analysis times using Caflux.

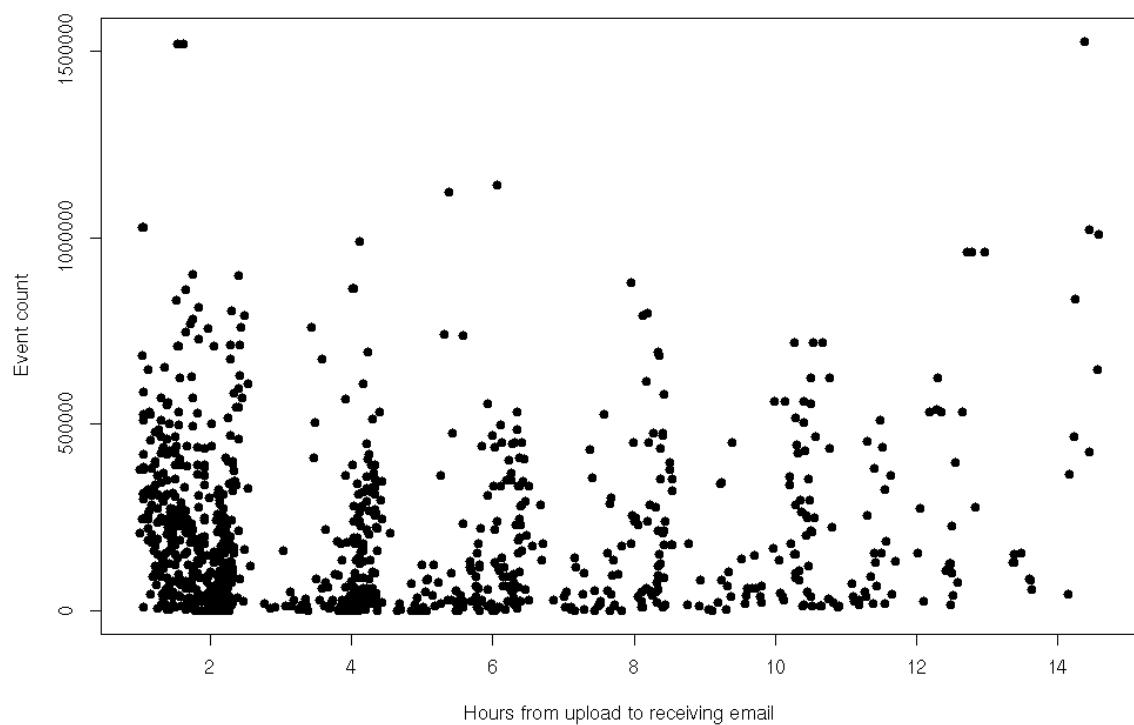


Figure 4.17: Plot of analysis times vs. event count. Only cases when analysis time was less than 19 hours are shown.

Bibliography

Journal articles and books

- [1] Belisle, C. J. P. Convergence theorems for a class of simulated annealing algorithms on Rd. *J. Applied Probability*. No. 29, 1992, pp. 885-895.
- [2] Chambers, J. M., Cleveland, W. S., Kleiner, B. and Tukey, P. A. (1983) Graphical Methods for Data Analysis. Wadsworth & Brooks/Cole
- [3] Hollander M and Wolfe DA. Nonparametric Statistical Methods. New York: John Wiley & Sons, 1973, pp. 115-120.
- [4] Kaposi AS, Veress G, Vásárhelyi B, Macardle P, Bailey S, Tulassay T, Treszl A. Cytometry-acquired calcium-flux data analysis in activated lymphocytes. *Cytometry A*. Volume 73, Issue 3, Mar 2008, pp. 246-53.
- [5] Kaposi A, Toldi G, Mészáros G, Szalay B, Veress G, Vásárhelyi B. Experimental conditions and mathematical analysis of kinetic measurements using flow cytometry - the FacsKin method. In: Flow Cytometry/Book 1. Schmid I (ed.). Intech, 2012. ISBN 989-953-307-355-1. In press.
- [6] Mészáros G, Szalay B, Toldi G, Kaposi A, Vásárhelyi B, Treszl A. Kinetic Measurements Using Flow Cytometry: New Methods for Monitoring Intracellular Processes. *Assay Drug Dev Technol.* Volume 10, Issue 1, Feb 2012, pp. 97-104.
- [7] Moore, A. W. Cross-validation for detecting and preventing overfitting (presentation). School of Computer Science, Carnegie Mellon University. URL: <http://www.autonlab.org/tutorials/overfit10.pdf> (accessed: 31 Mar 2012)
- [8] Murphy RF, Chused TM. A proposal for a flow cytometric data file standard. *Cytometry*, Volume 5, Issue 5, Sep 1984, pp. 553-5.
- [9] Nelder, J. A. and Mead, R. A simplex algorithm for function minimization. *Computer Journal*. No. 7, 1965, pp. 308-313.

- [10] Picard R.R., Cook, R.D. Cross-Validation of Regression Models. *Journal of the American Statistical Association*, Volume 79, Number 387, Sep 1984, pp. 575-583.
- [11] Roederer, M., Treister, A., Moore, W., Herzenberg, L.A. Probability binning comparison: a metric for quantitating univariate distribution differences. *Cytometry*, Volume 45, Issue 1, Sep 2001, pp. 37-46.
- [12] Roederer M. Spectral compensation for flow cytometry: Visualization artifacts, limitations, and caveats. *Cytometry A*. Volume 45, Issue 3, Nov 2001, pp. 194–205.
- [13] Seamer LC, Bagwell CB, Barden L, Redelman D, Salzman GC, Wood JC, Murphy RF. Proposed new data file standard for flow cytometry, version FCS 3.0. *Cytometry*. Volume 28, Issue 2, Jun 1997, pp. 118-22.

Websites

- [14] Apple Inc.: <http://www.apple.com> (accessed: 31 Mar 2012)
- [15] BD FACSDiva Software, BD Biosciences, 2350 Qume Drive, San Jose, California, USA 95131. <http://www.bdbiosciences.com/instruments/software/facsdiva> (accessed: 31 Mar 2012)
- [16] FCS format Version 3.1: <http://isac-net.org/Resources-for-Cytometrists/Data-Standards/Data-File-Standards/Flow-Cytometry-Data-File-Format-Standards.aspx> (accessed: 31 Mar 2012)
- [17] Flow Cytometry on Wikipedia: http://en.wikipedia.org/wiki/Flow_cytometry (accessed: 31 Mar 2012)
- [18] flowCore R package: <http://bioconductor.org/packages/2.2/bioc/html/flowCore.html> (accessed: 31 Mar 2012)
- [19] FlowJo Software, Tree Star, Inc. 340 A Street #101. Ashland, Oregon, USA 97520. <http://www.flowjo.com> (accessed: 31 Mar 2012)
- [20] Java, by Oracle Inc.: <http://www.java.com> (accessed: 31 Mar 2012)
- [21] Java Web Start: <http://docs.oracle.com/javase/6/docs/technotes/guides/javaws/> (accessed: 31 Mar 2012)

- [22] Department of Laboratory Medicine, Semmelweis University, Budapest, Hungary. <http://www.labmed.sote.hu> (accessed: 31 Mar 2012)
- [23] The Document Foundation LibreOffice: <http://www.libreoffice.org> (accessed: 31 Mar 2012)
- [24] NetBeans: <http://www.netbeans.org> (accessed: 31 Mar 2012)
- [25] NixOS Linux Distribution based on a lazy purely functional system configuration management language: <http://www.nixos.org> (accessed: 31 Mar 2012)
- [26] PNG format: http://en.wikipedia.org/wiki/Portable_Network_Graphics
- [27] R Development Core Team (2012). R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0, URL <http://www.R-project.org>.
- [28] Robert Green's DIY website: <http://www.rbggrn.net/content/43-java-single-application-instance> (accessed: 31 Mar 2012)
- [29] SendEmail Perl script by Brandon Zehm. Distributed under GPL licence. <http://caspian.dotconf.net/menu/Software/SendEmail> (accessed: 31 Mar 2012)
- [30] Ubuntu by Canonical Inc.: <http://www.ubuntu.com> (accessed: 31 Mar 2012)
- [31] Windows by Microsoft Inc.: <http://www.microsoft.com> (accessed: 31 Mar 2012)