

# Manual of *DataRail*

Flexible Informatics for Linking Diverse Data to Mathematical  
Models in MATLAB

Julio Saez-Rodriguez<sup>1,2,3</sup>, Arthur Goldsipe<sup>2,3</sup>,  
Jeremy Muhlich<sup>1,3</sup>, Leonidas G. Alexopoulos<sup>1,3</sup>, Bjorn Millard<sup>1,3</sup>,  
Douglas A. Lauffenburger<sup>2,3</sup> and Peter K. Sorger<sup>1,2,3</sup>

January 15, 2010

<sup>1</sup> Harvard Medical School, Department of Systems Biology, Boston, MA

<sup>2</sup> Massachusetts Institute of Technology, Department of Biological Engineering,  
Cambridge, MA

<sup>3</sup> Cell Decision Process Center, Massachusetts Institute of Technology,  
Cambridge, MA

---

Copyright 2007

Julio Saez-Rodriguez, Arthur Goldsipe  
sbpipeline@hms.harvard.edu

SBDataPipe is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 3 of the License, or (at your option) any later version.

SBDataPipe is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <http://www.gnu.org/licenses/>.

# Installation and Quick Start

Unzip the toolbox into a directory of your choice. Start MATLAB and change the working directory to the top directory of the toolbox. Run the file `startDataRail` to install the *DataRail* toolbox and run the main GUI.

## Contents

<b>1</b>	<b>Introduction</b>	<b>4</b>
<b>2</b>	<b>Data import, storage, and handling in MATLAB</b>	<b>5</b>
2.1	Importing Data . . . . .	5
2.2	The Compendium . . . . .	5
2.3	The Cubes . . . . .	6
2.4	The Labels . . . . .	7
2.4.1	Standard Labeling . . . . .	8
<b>3</b>	<b>User Interfaces</b>	<b>9</b>
<b>4</b>	<b>MATLAB functions</b>	<b>10</b>
4.1	Importing Data . . . . .	10
4.2	Processing Functions . . . . .	10
4.3	Plotting functions . . . . .	13
<b>5</b>	<b>Analysis</b>	<b>14</b>
5.1	Multiple Linear Regression . . . . .	14
5.2	PCA/PLSR . . . . .	14
5.3	Bayesian Inference (BNSL) . . . . .	14
5.4	Logic-based modeling (CNO) . . . . .	17
<b>6</b>	<b>MIDAS File generator</b>	<b>18</b>
6.1	Start-up . . . . .	18
6.2	Filling in Meta-data . . . . .	18
6.3	Adding Treatments . . . . .	19
6.4	Adding Measurements . . . . .	19
6.5	Saving the file . . . . .	20
6.6	Other options . . . . .	20

## **7 Acknowledgments**

**20**

# 1 Introduction

This short document is a documentation (manual) for *DataRail*. As complementary information, a tutorial, a presentation, and a paper can be found in <http://code.google.com/p/sbpipeline/downloads/>.

The goal of this toolbox is to provide an infrastructure to read, transform, store and document high throughput data. We use the format MIDAS (Minimal Amount of Information for Data Analysis in Systems Biology), a derivative of the (MIACA (Minimum Information About a Cellular Assay) formalism to store the data coming from various experimental devices. The MIDAS data is then transferred to MATLAB.

In MATLAB, the data is stored in data arrays (or cubes) and then transformed, visualized and exported to different analysis toolboxes. So far a Statistical toolbox to perform Principal Component Analysis (PCA) and Partial Least Squares Regression (PLSR) is available, as well as one to perform multiple linear regression analysis. Export to An Spotfire for visualization is also possible, as well as to *CellNetAnalyzer* for logical analysis and to *PottersWheel* for ODE-based simulation and parameter estimation. Several additional exports are planned, e.g. to BioConductor to perform different sorts of statistical analysis is also planned. For more information on *DataRail*, see [1].

## 2 Data import, storage, and handling in MATLAB

In this section we will roughly describe how the data is handled in *DataRail*. The user can do this either via Graphical User Interfaces (GUIs) or MATLAB scripting. To use the GUIs, no knowledge of MATLAB programming is required, but to write new scripts, some knowledge will be useful. Section 3 presents the GUIs, and Section 4 the MATLAB functions which can be called from the scripts.

### 2.1 Importing Data

In *DataRail*, you can either fill in a MIDAS file (as a comma separated values or Excel file; make sure when creating a csv file that you dont use quotes as text limiters) to `MidasImporter` or import from a particular experimental device. So far only the `biopleximporter` to import data from the Bioplex device is implemented. In both cases, one creates a `Compendium`.

The MIDAS format is a tabular format to describe data sets. Each row corresponds to an experiment, and the columns define the experimental conditions (with columns named starting with TR: plus the treatment, e.g. TR:EGF), times of measurements (columns named DA: measurement, e.g. DA:ERK) and values of measurements (in columns named with DV:, e.g. DV:ERK). If all the measurement for a certain experiment are performed at the same time, you can used a compress form where the time is only defined once in a column denoted with DA:All. The MIDAS format is described in detail in [1].

### 2.2 The Compendium

Our starting point is a set of experimental data. Here we use the term `Compendium` to refer to a set of data cubes arising from a common MIDAS file (and thus a common experiment(s)). Each `Compendium` is organized in a MATLAB structure with fields, `Name`, `Info`, and `data`, e.g.: A set of `compendia` can be collected inside a structure called `Project`.

```
Compendium =  
  Name: 'MC-LGA-11111-CytoInh50'  
  Info: 'Data produced from Leonidas Alexopoulos in Feb 07  
        with Primary hepatocytes and HepG2s'  
  data: [1x8 struct]
```

The `Name` usually contains the UID of the Compendium (see [1] and [2]). The `Info` field is a string with a brief documentation on the project.

## 2.3 The Cubes

The key mathematical structure in our pipeline is a data array or cube. Data cube is the word used in computer programming; this is referred to as a tensor in mathematics, and hypercube in geometry.

Each Compendium contains an array of data cube structures. The first data cube in the array is normally obtained by importing experimental data stored in the MIDAS format. Subsequent cubes are normally obtained by applying some function (normalization, discretization, etc, see Section 4.) to a previous data cube.

Each cube (e.g. `Compendium.data(1)`) is a structure that contains the actual data (the `Value` field), a short `Name` field, a longer `Info` field, the name of the cube from which it is derived (the `SourceData` field), the name of the function used to create the cube (the `Code` field), and a snapshot of the function for documentation and re-computing purposes, e.g.

```
Compendium.data(2) =  
  Value: [5-D double]  
  Name: 'BackgroundSubtracted'  
  Info: 'Removed the experimental background error from Raw Data'  
  Code: [1x1 struct]  
  Parameters: [1x1 struct]  
  SourceData: 'Raw'
```

The first cube is typically the raw data as it is imported from the MIDAS file. Accordingly, the raw data cube has as `sourcedata` the MIDAS file it was converted from:

```
Compendium.data(1)=  
  Value: [5-D double]  
  Name: 'Raw'  
  Info: 'Raw Data as it comes from Leo'  
  Code: [1x1 struct]  
  Parameters: [0x0 struct]  
  SourceData: 'MD-LGA-11111-CytoInh17phFI-BLK.csv'
```

To summarize, we use the following terminology:

- Compendium structure is the whole Compendium, e.g. `Data`,
- Cube structure is all the information related to a data cube, e.g. `Compendium.data(1)`
- Data cube is the actual numerical N-dimensional cube, `Compendium.data(1).Value`.

## 2.4 The Labels

`Labels` defines the Labels of the different dimensions of the data cubes. Typically, we define 5 different dimensions: the time, readout measured, and 3 other independent variables (e.g. Cell type, stimuli, and inhibitors). Most cubes have the same dimensions; as it is not necessarily the case, each cube has its own dimensions to ensure consistency.

```
Compendium.data(1).Labels=
1x5 struct array with fields:
    Names
    Value
```

Often, we use the so-called 'canonical form', with Time as the 2nd dimension and the Signals (or Measurements) as the 5th dimension, e.g.

```
Compendium.Labels.Names='CellTypes' 'Times' 'Cues'
                        'Inhibitors' 'Measurements'
```

Many functions work only on this canonical form.

The labels for most dimensions are saved as strings, but time is stored as a numerical vector.

```
Compendium.data(1).Labels(1).Value = 'HepG2' 'Primary'
```

```
Compendium.data(1).Labels(2).Value = 0 30 180
```

```
Compendium.data(1).Labels(3).Value = 'DMSO' 'FASL' 'IFNg' 'TNF'
    'IL1a' 'IL6' 'IGF1' 'TGFa' 'TGFB' 'HGF' 'LPS'
```

```
Compendium.data(1).Labels(4).Value = 'DMSOi' 'MEK12i' 'p38i' 'PI3Ki'
    'IKKabi' 'mTORi' 'GSK3i' 'JNK12i'
```

```
Compendium.data(1).Labels(5).Value = 'AKT' 'ERK12' 'GSK3' 'Ikb'
    'JNK12' 'p38' 'p70S6' 'p90RSK' 'STAT3' 'cJUN' 'CREB' 'HistH3'
    'HSP27' 'IRS1s' 'MEK12' 'p53' 'STAT6'
```

### 2.4.1 Standard Labeling

While the framework of the pipeline is flexible, it is recommended to be consistent in the labeling of the dimensions. This avoid compatibility problems, e.g. if you call your data about ERK phosphorylation p-ERK12 but in your Boolean model you call this state ERK. We use as standard the following names:

**Cytokines-** Either when they are cues or readouts, we use the following names, alphabetically ordered:

'bNGF',  
'CTACK',  
'DMSO',  
'Eotaxin',  
'FASL', 'FGFbasic',  
'GCSF', 'GMCSF', 'GROa',  
'HGF',  
'ICAM1', 'IGF1', 'IFNg', 'IFNa2', 'IL1a', 'IL1b', 'IL1ra', 'IL2', 'IL2ra', 'IL3',  
'IL4', 'IL5', 'IL6', 'IL7', 'IL8', 'IL9', 'IL10', 'IL12p40', 'IL12p70', 'IL13',  
'IL15', 'IL16', 'IL17', 'IL18', 'IP10',  
'LIF', 'LPS',  
'MCP1', 'MCP3', 'MCSF', 'MIF', 'MIG', 'MIP1a', 'MIP1b',  
'NOCYTO',  
'PDGFbb',  
'RANTES',  
'SCF', 'SCGFb', 'SDF1a',  
'TGFa', 'TGFb', 'TNFb', 'TNFa', 'TRAIL',  
'VCAM1', 'VEGF'.

Note that both 'NOCYTO' and 'DMSO' are used to determine the absence of cytokine

**Phosphorylation readouts-** 'AKT',

'cJUN' 'CREB',  
'ERK12',  
'GSK3',  
'HistH3', 'HSP27',  
'Ikb', 'IRS1s',  
'JNK12,'

'MEK12',  
'p38', 'p53', 'p70S6', 'p90RSK',  
'STAT3', 'STAT6'.

**Inhibitors-** We use the syntax: 'Protein inhibited1' 'i' 'Protein inhibited2' 'i' '-free text'. Do not use - inside the free text. Use the free text particularly if you have different inhibitors for the same protein.

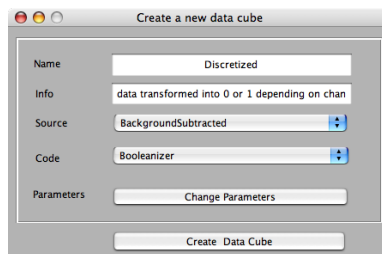
E.g. PI3Ki-inh1, PI3Ki-inh2. If you have only one, you can call it PI3Ki.

You can use the label 'NOINHIB' to define the absence of inhibitor.

### 3 User Interfaces

A simple set of MATLAB user interfaces have been coded to assist users who are not familiar with MATLAB programming. GUIs have been created that support:

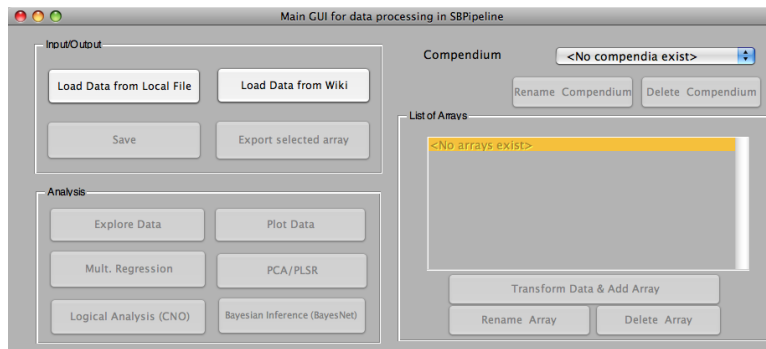
- Loading data from MIDAS or MATLAB files `GuiMidasImporter`
- Creating of different cubes `GuiCreateDataCube` (see Figure 1). Most of the functions you can choose here have their own GUI which pops up if you want to change parameters.



**Figure 1: Screenshot of GuiCreateDataCube.**

- Plotting data (`GuiPlotAllSignalsCompact`).
- Multiple Regression (`GuiMultipleReg`).
- PCA/PLSR analysis (`GuiPLSR`)

- In addition, a main GUI `GuiMain` was created to give simple access to the other GUIs (see Figure 3). When you load data, you can choose either a MIDAS file in csv (comma separated values) format, a Compendium in MATLAB's mat format, or a Bioplex data file as an excel spreadsheet (.xls). If you already have loaded a Compendium, `SBDDataPipe` will ask you to either replace it with the new one or to append new data using `AppendCompendium`.



**Figure 2: Screenshot of GuiMain.**

## 4 MATLAB functions

### 4.1 Importing Data

The main function is `MidasImporter` which imports a data in MIDAS format into a datacube. If your time points are the same for all your data in your MIDAS file, it is sufficient if you have one column called `DA:ALL`. The importer will recognize it and handle the file correctly.

If your data is in bioplex form, you can load it using the function `bioplex2midas`.

### 4.2 Processing Functions

The master function `createDataCube` transforms cubes into other cubes. The function gets as input the fields required to define a new cube structure (Name, Info, Code, Parameters, and SourceData). For example:

```
Compendium.data(cub) = createDataCube(...  
  'Name', 'RelativeMaxSignal', ...  
  'Info', 'Normalized data where each measurement  
          is relative to the maximal value for  
          the same signal', ...  
  'Code', @MaxSignalRelativator, ...  
  'Parameters', {'ExpNoise', 0}, ...  
  'CodeHashArray', Compendium.data(Compendium.v.Relat).CodeHashArray, ...  
  'SourceData', Compendium.data(Compendium.v.BackgroundSubtracted));
```

A suite of functions to transform cubes has been created:

- `averageReplicates` averages data across the replicate dimension.
- `Booleanizer` discretizes a 5-D canonical data cube into a 5D 0,1 cube.
- `centerAndScale` is a DataRail interface to nprocess of the nway toolbox.
- `collapseDataCube` collapses dimensions and labels in a datacube.
- `CreateCNADData` converts Cube of Boolean Data into CellNetAnalyzer Format.
- `createSubcube` to slice a subcube out of cube.
- `CubicalizeCNAArray` takes an array from CNA computations and reconverts it into a cube.
- `CustomMath` allows to customize simple mathematical functions to normalize the data.
- `Discretize` discretizes the dataset with various methods (quantile/interval, deterministic/stochastic, with and without using TMI).
- `ExpandCubeto2D` converts a 5D data array into a 2D matrix for e.g. Multiple regression or PLSR.
- `GetConcentrations` converts a canonical 5-D datacube from 50-plex signal to concentration of cytokines.
- `GetRelative` normalizes a datacube with respect to a particular treatment.
- `GetTimeCompressed` compresses data cube of n time points into 2 or 3 + 0 timepoint.
- `joinCubes` joins two cubes together.
- `Normalize` normalizes data in various ways (initial time point, maximum value, specified value, respective value of certain cell type or total amount of protein).
- `RemoveBasalLevel` subtracts to every signal the value for t=0.

- `RemoveInputEqOutput` removes data from experiments where the input is the same as the output.

A couple of auxiliary functions are also available:

- `RefreshCompendium` allows to refresh a `Compendium` after changing something (e.g. some data, a parameter). It is also useful to create copies of `Compendia`.
- `AppendCompendium` to add new experimental data to a `Compendium`.
- `CreateShortcuts` creates an structure of shortcuts to access the elements in the cubes. These shortcuts allow you to easily find particular positions in a certain dimension, e.g.

```
Compendium.data(1).Value(v.Prim,1,v.EGF,v.PI3Ki,v.ERK12)
```

- `CreateCalibration` creates the calibration curves for the cytokine concentrations.
- `PolishLabels` removes the MIDAS tags (TR, DA, DV) and other strings.
- `setParameters` sets the parameters for a particular function using explicit values wherever given or the default elsewhere.

### 4.3 Plotting functions

Several plotting routines allow different visualization of the data. Each of them can be customized by parameters (see particular functions for details). Here, the general plotting functions are described; each analysis toolbox (e.g. Boolean or Multiple Regression) may have its specific plotting functions.

- `PlotAllSignalsCompact` plots a data cube in a set of subplots. Through the parameters many different plots can be created.
- `PlotCubesStructure` creates a graph with the structure of cubes included in particular `Compendium`.

=====

## 5 Analysis

### 5.1 Multiple Linear Regression

*DataRail* allows Multiple Linear Regression (MLR) analysis of data both through a GUI and through script-based functions. Before the analysis is performed, the compendium must contain a 2-dimensional array, which will be derived from a variation of your original 5-dimensional array. If the Compendium does not already contain a 2D array, a prompt to collapse the cube will be displayed before the analysis is performed. Clicking the `Mult. Regression` box opens a new GUI, termed `GuiMultipleReg`, which allows the user to choose the inputs and outputs for the MLR. `GuiMultipleReg` allows the inputs and outputs to be either from the same or different compendia. Designate the specific subset of data from the inputs and outputs by clicking the pulldown menu and choosing the appropriate data set (i.e. cues, inhibitors, readouts). To plot the weights, plot the fit of the data, or export the weights to visualize the entire network in Cytoscape, click the appropriate check boxes. See Figure 3 for an example.

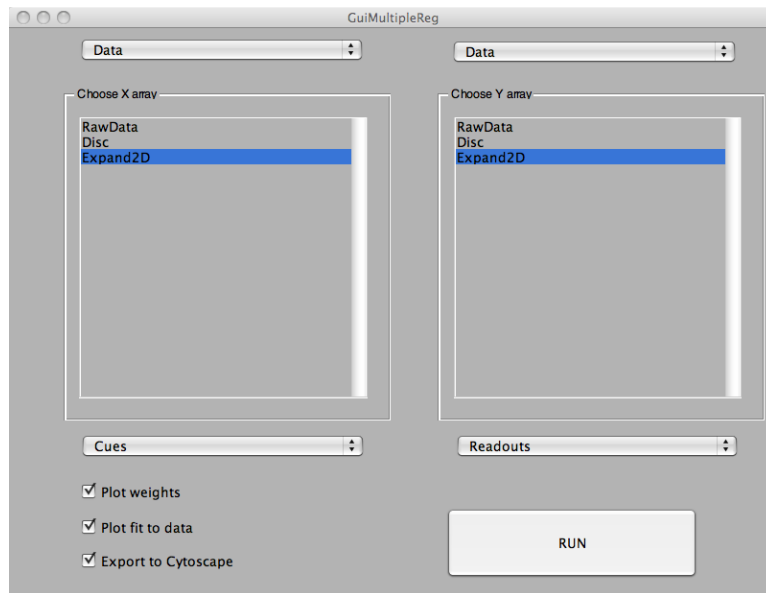
Below is an example of the plotted weights.

### 5.2 PCA/PLSR

PCA and PLSR are techniques for reducing the dimensionality of data by identifying correlations between variables. PCA (principle component analysis) is applied to a single array of data. PLSR (partial least-squares regression) is used to predict one array from another array (see Janes, K., et al. A Systems Model of Signaling Identifies a Molecular Basis Set for Cytokine-Induced Apoptosis. *Science*, 310, 1646-1653 (2005) doi:10.1126/science.1116598). You can find in the tutorial an example of doing PCA and PLSR analysis with *DataRail*.

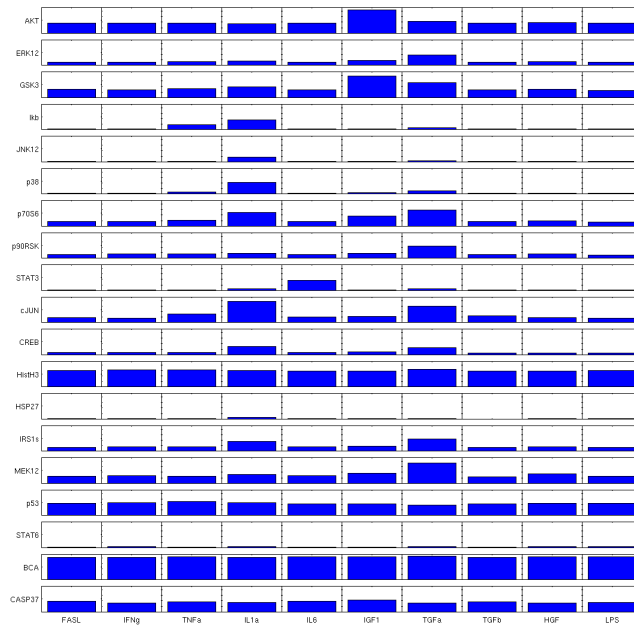
### 5.3 Bayesian Inference (BNSL)

Bayesian inference methods based on functions and code of BNSL (Bayesian Network Structure Learning), developed by D. Eaton and K. Murphy, see <http://www.cs.ubc.ca/murphy/>. BNSL can be launched by pressing the button Bayesian Inference (BNSL)

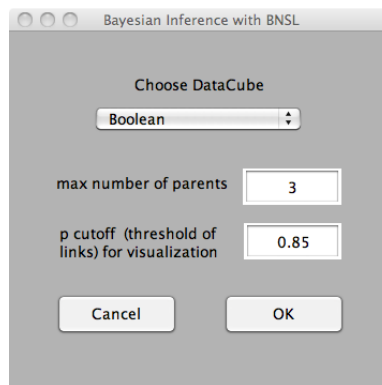


**Figure 3: Screenshot of GuiMultipleReg.**

in the main DataRail GUI. This will show a menu (Figure 5.3 where you can choose which data use to infer a network (it has to be discretized data), the maximum number of parents and a cut-off of the p-value for visualization. After the inference has been run, DataRail will visualize the resulting network and offer the option to store the results back in the Compendium.



**Figure 4: Screenshot of weight output.**



**Figure 5: Screenshot of interface to launch BNSL for bayesian inference.**

## 5.4 Logic-based modeling (CNO)

DataRail can also be linked to CellNetOptimizer (CNO), a MATLAB toolbox for creating logic-based models of signal transduction networks, and training them against high-throughput biochemical data (see Saez-Rodriguez et al., Discrete logic modeling as a means to link protein signaling networks with functional analysis of mammalian signal transduction, *Molecular Systems Biology*, 5:331, 2009.). CNO can be downloaded from

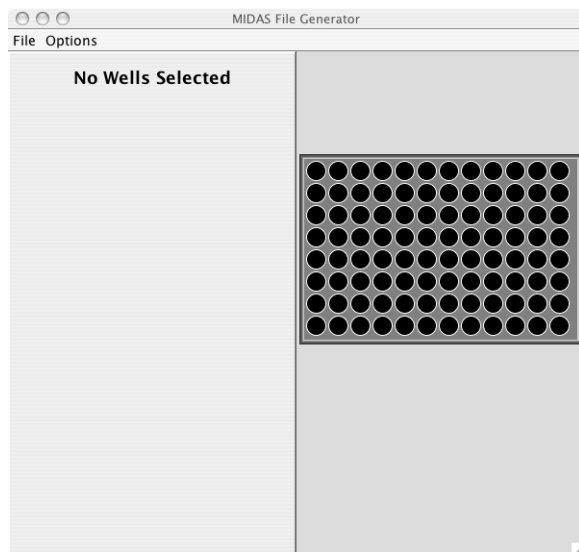
<http://www.cdpcenter.org/resources/software/cellnetoptimizer/>. Unzip the CNO folder in your main DataRail folder, and then, from DataRail, click the button Logical Analysis (CNO) to launch CellNetOptimizer. CNO comes with a dedicated tutorial about how to use it.

## 6 MIDAS File generator

A java stand alone application, which can be used together with *DataRail* is available.

### 6.1 Start-up

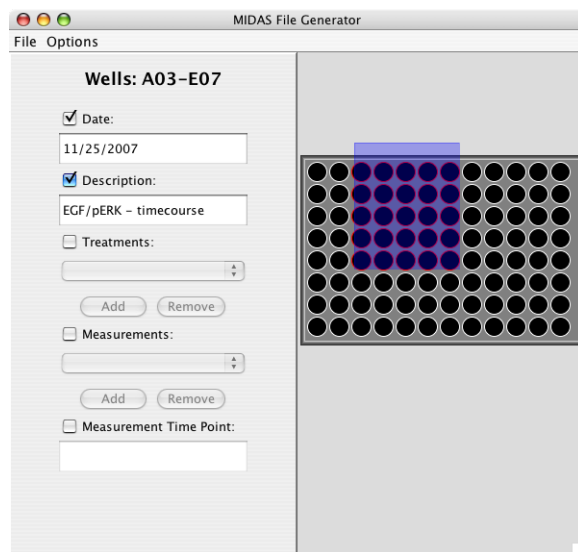
Upon starting the application a default 96-well plate interface will be displayed in the right panel. The left panel contains input fields for filing in and browsing MIDAS meta-data for selected wells. At startup, there are no wells selected and thus the left panel is blank (see Figure 6.1).



**Figure 6: Screenshot of the MIDAS File Generator**

### 6.2 Filling in Meta-data

To start filling in the meta-data, highlight and/or click on the desired wells at which time the left panel will become active. The left panel will display input fields for information describing the experiment, including: date, description, well treatment, targets measured, and time of well measurement. Click on the check boxes for the fields you wish to update then enter the desired data (see Figure 7).



**Figure 7: Filling Metata data in the MIDAS File Generator**

### 6.3 Adding Treatments

Click on the **Treatments Checkbox** to activate its fields. Click the **Add** button, which will pop-up a dialog window that allows you to create a new treatment or add an existing treatment that already exists in the current plate. When creating a new treatment, enter the name of the treatment and a value. For example, if you are stimulating a particular well with 50 ng/mL EGF, the name would be 'EGF' and value '50'. Upon entering the data, press the **Enter** button to add it to the appropriate wells. A treatment may be removed by selecting the desired treatment in the main GUIs combo box and pressing the **Remove** button (see Figure 8).

### 6.4 Adding Measurements

Adding a feature you are measuring in a well is similar to adding a treatment except the only input text field is the name of the measured feature. The experimenter will fill in the actual value after the MIDAS file template has been generated and experimental data collected. A second difference is that each measurement has an associated time of acquisition. This time can be entered in the last text input field in the main GUI.

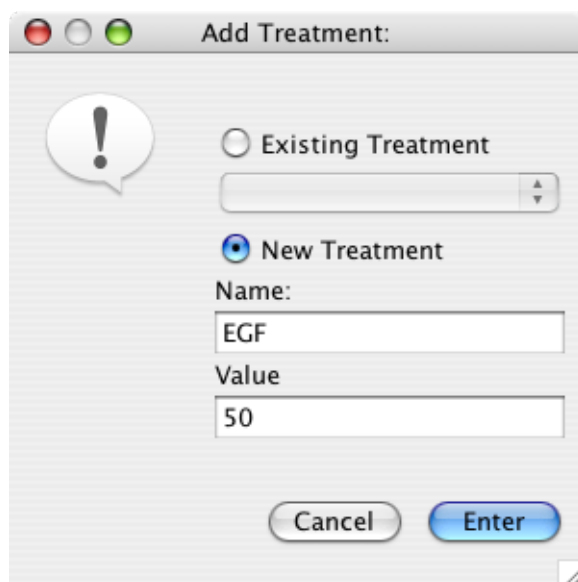


Figure 8: Adding treatments to the MIDAS File Generator

## 6.5 Saving the file

To export the final CSV (comma separated values) MIDAS file to disk, select 'Save file as' in the **File** menu.

## 6.6 Other options

The Option **Plate size**, menu allows the user to select different plate dimensions that are commonly used in the lab (6, 12, 24, 96, 384-well).

## 7 Acknowledgments

We thank MingSheng Zhang and Mark Fleury for testing *DataRail*, and Nickel Dittrich and Jooho Chung for contributing to *DataRail* and to this manual.

## References

- [1] Saez-Rodriguez J, Goldsipe A, Muhlich J, Alexopoulos L, Millard B, Laufenburger DA, Sorger PK Flexible Informatics for Linking Diverse Data to Mathematical Models via DataRail. *Bioinformatics*.
- [2] Muhlich J, et al. 24(6):840-847..