SASS Documentation

Release 0.8.0

Marcel Stefko, Kyle M. Douglass

Jun 05, 2018

Contents

SMLM Acquisition Simulation Software

CHAPTER 1

Quickstart

Contents

1.1 Installation

SASS is both a standalone application and a Fiji plugin.

1.1.1 Standalone

1. Download the latest .jar file from the SASS releases page.
2. You will also need to download the latest ALICA_ACPack .jar, which contains the run-time components for control systems simulations.
3. Place both .jars in the folder of your choosing.

1.1.2 Fiji

1. Download the latest .jar file from the SASS releases page.
2. You will also need to download the latest ALICA_ACPack .jar, which contains the run-time components for control systems simulations.
3. Copy the SASS .jar file into your *~/Fiji.app/plugins/* folder, where *Fiji.app* is root directory for your Fiji installation.
4. Copy the ALICA_ACPack .jar file into your *~/Fiji.app/jars* folder.
5. Restart Fiji.

You should now see *SASS* appear as a menu item in the the *Plugins* menu.

1.2 Run a simulation

1.2.1 Standalone

Before starting, make sure that you have a copy of the file example_random_2d_fluorophores.bsh from the SASS respository's *scripts* folder. When using SASS in standalone mode, it is most commonly used as a command line application.

1. From the command line, navigate to the folder where you placed the SASS .jar file that you downloaded in the installation step.
2. Enter the command *java -jar SASS_-<VERSION>.jar -s example_random_2d_fluorophores.bsh*.
3. If you want to save the simulation's output, ensure that any call to the *saveStack(...)* method is uncommented inside the script and rerun the simulation.

1.2.2 Fiji

1. Launch Fiji. (If you're launch Fiji from the command line, ensure that you are first in the Fiji root directory.)
2. Navigate to *Plugins* > *SASS* > *Simulator*.
3. Ensure that **Manual** is selected in the *Controller* drop-down box.
4. Click the *Initialize* button.
5. Rearrange the windows so that you can find the dialog with the controller set point and the *Start* and *Stop* buttons.
6. Click *Start* to start the simulation. You should see images begin streaming into the simulation's image stack.
7. Click the *Stop* button to pause the simulation.
8. Change the *Controller setpoint* value and click *Start* again to resume the simulation with a new laser power.

CHAPTER 2

Simulation Models

2.1 Fluorescence dynamics

The fluorescence dynamics in SASS are modeled as memoryless state systems. Such systems are comprised of two or more states that a fluorophore may occupy at any given time. During the course of an experiment, the fluorophore may randomly transition from its current state m to a new state n, and the probability with which this transition occurs is determined partly by the so-called rate constant k_{mn}.

Memorylessness means that the probability to transition to any accessible state does not depend on the time that the fluorophore has already spent in its current state. This assumption is well-founded: it is unlikely that a fluorescent molecule possesses some mechanism to keep track of time. Under the assumption of memorylessness, the length of the time interval t that is spent by a fluorophore in its current state S_m before making a transition to state S_n is given by an exponential probability density function

$$p_{mn}(t) = k_{mn} e^{-k_{mn} t}$$

When multiple states are accessible from S_m, then it may be shown that the probability that the fluorophore will have transitioned to the specific state S_n is

$$P(S_n, t = \infty | S_m, t = 0) = \frac{k_{mn}}{K}$$

where $K \equiv \sum_n k_{mn}$. Thus, the rate constants determine the relative probabilities of the transitions to different states.

2.1.1 Algorithm for state system simulations

The algorithm for simulating the state transitions proceeds as follows:

1. The fluorescent molecule is assigned a pre-defined starting state S_m.
2. Next, a random transition time from the molecule's current state is drawn for each accessible state n from an exponential distribution, $\forall n : t_{mn} \sim \text{Exp}(\tau_{mn})$ where $\tau_{mn} \equiv 1/k_{mn}$ is the average of the distribution.
3. The smallest value from this set of transition times is computed and stored as the molecule's transition time $T \equiv \text{Min}(t_{mn})$. The corresponding molecular state S_n is stored for use in the next step.

4. The simulation time is advanced one time step. If, during this time, a total amount of time has elapsed that is greater than the previously calculated transition time T, then the molecule is transitioned into its next state. The new next state and its transition time are generated and stored in the manner just described.
5. This process is repeated as the simulation continues until a pre-determined number of time steps have occurred or it is stopped by the user.

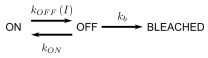
2.1.2 Non-stationary state transitions

In PALM/STORM type experiments, one or more rate constants depend on the light irradiance (power per area) of one or more light sources. Indeed, adjusting the power during an acquisition is a common way to optimize the quality of datasets derived from such experiments because it offers a direct way to tune the density of fluorophores in a light-emitting state.

When the laser irradiance varies with time, so too do the rate constants and, therefore, the relative numbers of the fluorophores found in each state. Fortunately, the memorylessness property makes it easy to adapt the above algorithm to account for a changing irradiance. At each time step of the simulation, a check is performed to see whether the laser irradiance has changed. If it has, new rate constants are computed and a new transition time and state are derived from the algorithm described above.

2.1.3 State system representations

As an example of how state systems are represented in SASS, consider the simplified three-state fluorophore model pictured below.



In this simple model, the fluorophore may be in a fluorescence emitting (ON) state, a non-emitting (OFF) state, and an irreversibly bleached state from which it may never recover. (This model is perhaps too simplistic as it does not account for the typically numerous non-emitting states that real fluorophores possess. It does, however, capture the essential behavior in a SMLM experiment.)

The transition rate from OFF to ON is a constant, k_{ON}, as is the rate k_b from the OFF to the BLEACHED state. The ON to OFF rate k_{OFF} is a function of the irradiance and may be expanded as

$$k_{OFF}(I) = k_{OFF,0} + k_{OFF,1} I + k_{OFF,2} I^2 + \cdots$$

Let's assume that k_{OFF} is at most linear with the irradiance. Then, the full dynamics of the fluorophore may be

specified by a $3 \times 3 \times 2$ matrix M

$$M_{:,:,1} = \begin{bmatrix} \\ k_{OFF,0} \\ 0 \\ k_{ON,0} \\ 0 \\ k_{b,0} \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

$$M_{:,:,2} = \begin{bmatrix} \\ k_{OFF,1} \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

(Note that some browsers may not render the first elements of the above matrices. Both elements are 0.)

The rows of each matrix represent the state being *transitioned from* (ON, OFF, and BLEACHED states respectively), while the columns represent the state that is *transitioned to* (in the same order). For example, the first row of $M_{:,:,1}$ indicates that $k_{OFF,0}$ is the zero-order term for the rate coefficient polynomial expansion in I from the ON state to the OFF state. Here, row number one corresponds to the ON state and column number 2 corresponds to the OFF state. The corresponding element in the second matrix $M_{:,:,2}$ is $k_{OFF,1}$ and indicates that the rate coefficient is linearly proportional to the irradiance. If there were a third matrix $M_{:,:,3}$ with a $k_{OFF,2}$ element, then this would indicate a second-order polynomial term for the dependence of k on I. Zeros for all the remaining elements in $M_{:,:,2}$ indicate that no other rates depend on the irradiance.

Any fluorophore state system may be implemented in SASS by specifying the matrix M.

2.2 Shot noise and sensor noise

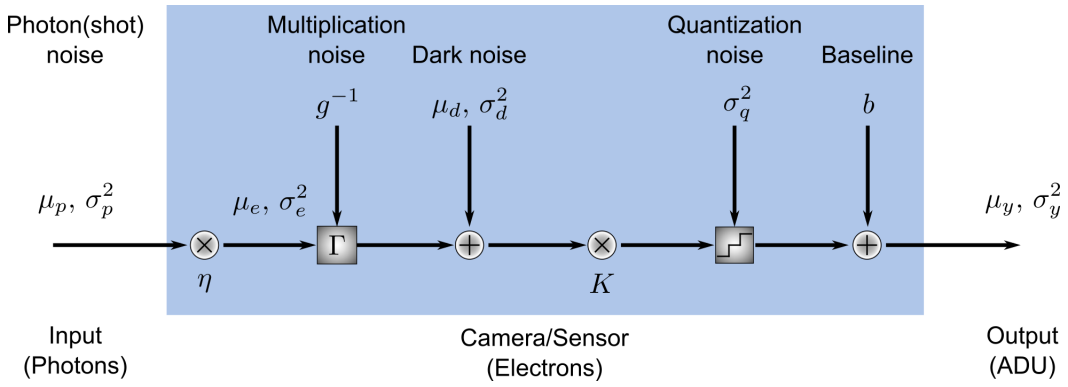
There are two noise models employed by SASS: photon shot noise—which accounts for the quantum nature of fluorescence emission—and sensor noise. Sensor noise is based on the models described in these two documents:

- Basden, Haniff, and Mackay, "Photon counting strategies with low-light-level CCDs," Mon. Not. R. Aston. Soc. 345, 1187-1197 (2003)
- The EMVA 1288 Standard

Sensor noise models in SASS currently do not account for spatial non-uniformities or defect pixels; each pixel is assumed independent from all other pixels. Furthermore, each pixel has identical statistical properties to all other pixels.

Additional assumptions employed in SASS include:

- The sensor is linear.
- Noise sources are wide sense stationary with respect to time and space.
- Only quantum efficiency is wavelength-dependent.
- Only dark current is temperature dependent.



2.2.1 Shot noise

Photon shot noise (or just shot noise) represents fluctuations in the number of photons incident on a pixel between different frame exposures. It is due to the quantum nature of fluorescence emission and is not dependent upon any properties of the image sensor.

Let μ_p represent the mean number of photons incident upon a pixel during the exposure of a given frame. The number of photoelectrons μ_e generated by these photons is given by

$$\mu_e = \eta\mu_p$$

where η is the quantum efficiency of the sensor and, in general, depends on the wavelength of the light.

Fluorescence emission is well-modeled as a Poisson process. Under this condition, the mean number of photoelectrons will be equivalent to the variance $sigma_e^2$ of the number of photoelectrons generated over time.

$$\sigma_e^2 = \mu_e$$

2.2.2 Sensor temporal noise

Within the sensor, photoelectrons are converted to analog-to-digital units (ADU) through a step-wise process involving

1. the amplification of the signal and the addition of multiplication noise (for cameras possessing a multiplication register),
2. the addition of dark noise, which consists of readout noise and dark current noise,
3. the conversion of electrons to voltages by multiplication with a constant system gain factor,
4. and quantization of the voltage to discrete ADU values and summation with a constant baseline value.

The number of photoelectrons that is generated within the pixels of an electron multiplying CCD (EMCCD) is amplified within a serial register via electron avalanche multiplication. This process is random and introduces a multiplicative noise that is modeled as a gamma distribution $\Gamma\left(\mu_e, g^{-1}\right)$ where g^{-1} is the inverse value of the camera's EM gain. (Note that in some notations the second parameter of the gamma distribution is denoted directly by the gain, not its inverse.) Sensors such as sCMOS cameras that lack a serial multiplication register are modeled in SASS by setting the EM gain value to 0.

Following the multiplication register, dark current noise is added to the signal to account for thermally excited electrons within the pixels. Dark current is modeled as a zero-mean Gaussian distribution whose standard deviation is a free parameter. Typically, the value for this parameter is found by assuming that dark current is also a Poisson process whose variance is equivalent to the mean number of dark current electrons $\mu_I t_{exp}$. Here, μ_I is the dark current in electrons per time and t_{exp} is the exposure time of the frame. μ_I is dependent on temperature in general. Dark current is often negligible in microscopy experiments, so it may often be safely ignored.

The total number of amplified photoelectrons and dark current electrons are then readout as a voltage, which introduces a readout noise. Readout noise is modeled as a zero-mean Gaussian distribution whose standard deviation is also a free parameter. The value for this parameter is often given on camera specification sheets as a median or root-mean-square (RMS) number of electrons. (RMS readout noise is preferred for sCMOS cameras because of pixel-to-pixel variation in the values.) Some camera manufacturers will combine dark current and readout noise into a single noise source known as dark noise with mean μ_d and variance σ_d^2.

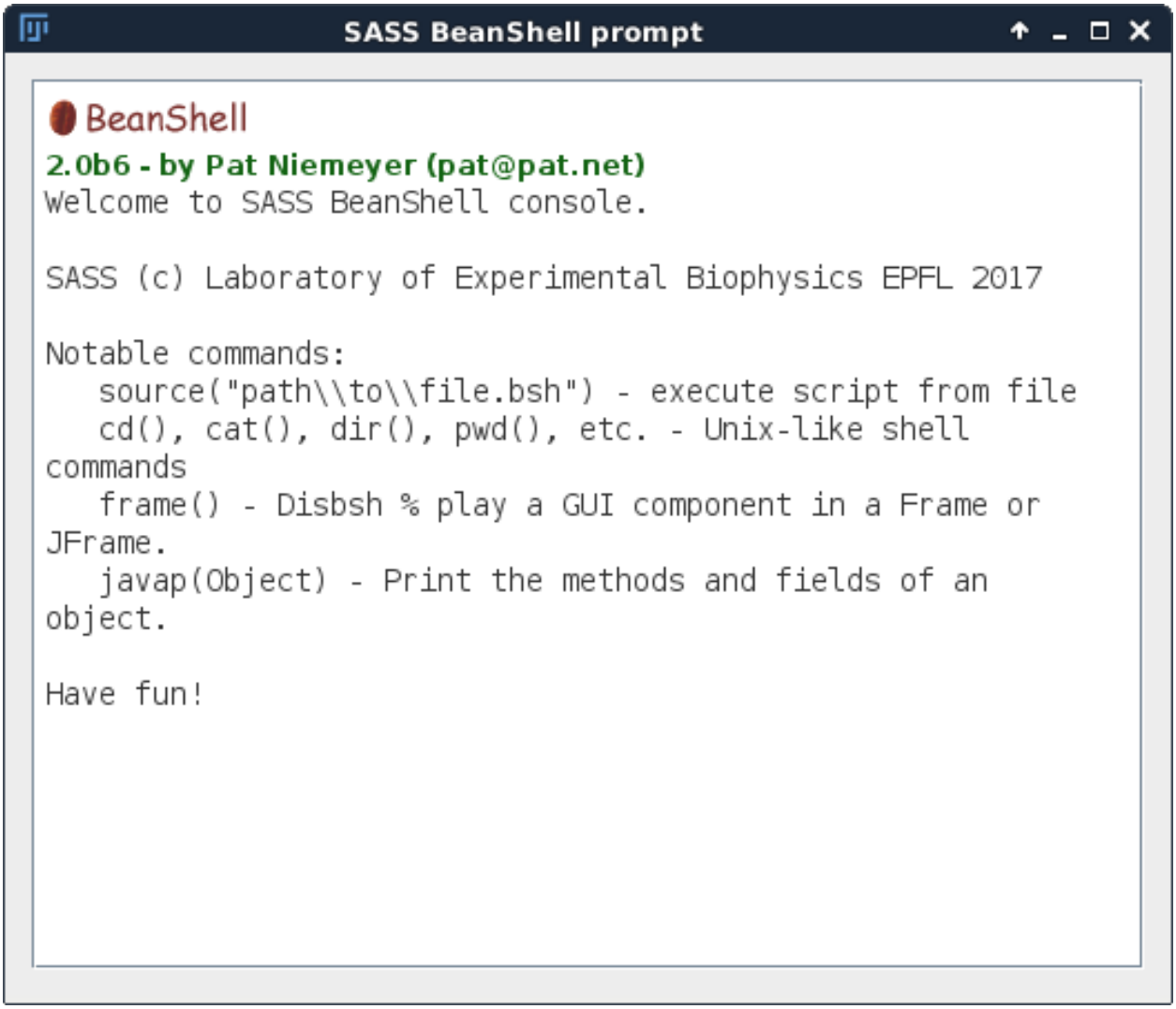
After addition of the readout noise, the voltage signal is amplified by another free parameter found on camera specification sheets, the system gain K. Finally, the signal is quantized into discrete ADUs and optionally summed with a constant baseline b to prevent negative pixel values. This baseline is often about 100 ADU. The quantization noise is a uniform distribution with variance $\sigma_q^2 = \frac{1}{12}\,ADU^2$. It is automatically accounted for in the code by converting from double to integer data types.

CHAPTER 3

Scripting Interface

3.1 The SASS Command Prompt

SASS includes a beanshell scripting interface that supports the execution of either pre-made or *ad hoc* scripts for easily running and repeating simulations. The interface is accessed through the Plugins menu bar via *Plugins > SASS > Command Prompt*.



Inside this prompt you have access to a few Unix-like shell commands by appending `();` to the command name. For example:

```
pwd();
```

prints the current working directory.

3.2 Running Beanshell Scripts

Beanshell scripts that setup and launch localization microscopy simulations may be run with the `source()` command. For security reasons, you will want to ensure that the file you are sourcing is trustworthy because the Beanshell interpreter will run whatever code is contained within the file.

Here is how one would launch the **example_run_generator.bsh** example script from within the command prompt and which launches a basic PALM simulation:

```
source("/path/to/examples/example_run_generator.bsh");
```

Please be sure to change the path argument above to one for your specific machine, which includes changing / to \ if you are using Windows.

3.2.1 From the shell/command line

To better facilitate batch processing and complex workflows, we made it possible to run a Beanshell script directly from the command line by invoking the SASS .jar directly through the Java Virtual Machine:

```
java -jar path/to/SASS/SASS.jar -s path/to/examples/example_run_generator.bsh
```

As you can see, you only need to pass the path to the .jar file on your machine and a **-s** argument followed by the path to the Beanshell script.

3.3 Example Scripts

Example scripts for performing 2D and 3D simulations with PALM and STORM models may be found in the examples folder in the SASS parent directory.

https://github.com/LEB-EPFL/SASS/tree/master/scripts

CHAPTER 4

Remote Procedure Calls and the SASS Server

4.1 Introduction

It is possible to control a SASS simulation from a programming language other than Java or even remotely over a network. This feature is enabled by the SASS remote procedure call (RPC) server. The idea of the RPC server is simple: it listens on a network port for commands sent by other languages and/or computers. When it receives a command, it performs the requested operation and returns any data that is associated with the command.

For example, after initializing a simulation and starting the server, a Python script on the same PC could adjust the laser power on the simulated microscope. It could then ask the server to simulate five new images and return them to the Python interpreter for further processing.

As another example, a C++ program could run a simulation by connecting to the server remotely over a network. The details of setting up your networked, such as ensuring the correct ports are open in your firewall, are beyond the scope of this documentation.

The RPC service was created using Apache Thrift.

4.2 Starting the server

There are three ways to start the server: via the command line, inside the ImageJ GUI, and via a Beanshell script.

4.2.1 Command line

Enter the following command in a console window to start the server from the command line

```
java -jar PATH_TO_SASS_JAR -r CONFIGURATION_FILE
```

The above command requires two arguments. **PATH_TO_SASS_JAR** is the path and name of the SASS .jar file, which can be downloaded from the releases page of the GitHub repository. **CONFIGURATION_FILE** is a file that specifies the simulation configuration. This file can be created created and saved from inside the SASS ImageJ GUI.

The command will start the server on the default port, which was 9090 at the time of this writing. If instead you wish to specify the port number, use

```
java -jar PATH_TO_SASS_JAR -p PORT -r CONFIGURATION_FILE
```

4.2.2 ImageJ

1. Open the server configuration dialog from the menu bar by clicking **Plugins > SASS > Server**.
2. Enter the port number you wish to use for communications with the server. Usually the default (9090) is fine.
3. Next, you will need a configuration file that defines your simulation parameters. This should be a *.sass* file containing the simulation details. You can create one by navigating to **Plugins > SASS > Simulator**, adjusting the simulation parameters as desired, then clicking the **Save...** button.
4. Once you have a configuration file, click the **Select configuration...** button, navigate to your file, and open it.
5. The **Start** button should now be enabled. Click it and the simulation will initialize. (This may take a few seconds depending on the size of your simulation.)
6. When the server has started, you should see the **Server running** message in the status field.
7. To stop the server, either click the **Stop** button or exit the server control window.

If you are using Fiji, then you can see status updates from the server by navigating to **Window >> Console** on the menu bar.

4.2.3 Beanshell script

There is an example script called **example_server.bsh** in the scripts folder of SASS that demonstrates how to launch the server through a Beanshell script. After creating a Microscope instance named *microscope*, simply create and launch the server with these lines

```
RPCServer server = new RPCServer(microscope, 9090);
server.serve();
```

Note that you will need to first import RPCServer with the command

```
import ch.epfl.leb.sass.server.RPCServer;
```

This code will initialize the server to listen on port 9090 and launch it. If you run the script from the command line, then you can kill the server by typing **Ctrl-C**.

4.3 Server communications

4.3.1 Services

The RPC server works by providing clearly-defined services to clients. Roughly speaking, a service is just a command made by a client that changes the simulation state and/or returns some data. A client must therefore know what services are provided by the server.

The SASS RPC server is implemented using Apache Thrift. The types of services that are provided by the server are defined in the RPCServer.thrift file in the *thrift* folder of the SASS root directory. Here is what the very first RPCServer.thrift file looked like

```
namespace java ch.epfl.leb.sass.server
namespace py remotesim

service RemoteSimulationService {

  /**
   * Returns the simulation server's current status.
   */
  string getServerStatus(),

  /**
   * Increments the simulation by one time step and returns an image.
   */
  binary getNextImage(),

  /**
   * Changes the simulation'ss fluorescence activation laser power.
   */
  void setActivationLaserPower(1: double power),

  /**
   * Returns information about the current state of each emitter in
   * a JSON string.
   */
  string getSimulationState()

}
```

This script defines the package names for the Java and Python code, respectively, and then defines the service that the server provides. There are four method calls:

1. getServerStatus()
2. getNextImage()
3. setActivationLaserPower
4. getSimulationState

The comments above the method definitions describe what each method does, and the data type that the method returns (string, binary, or void) is specific to Thrift's IDL language. After this script is compiled by the Thrift compiler into Java and Python code, they are converted into the corresponding data types in each language.

Note that the SASS RPC server sends images as tif-encoded byte strings and the simulation state as JSON strings. You will need to decode this information after its received in your target language.

4.3.2 A Python client

The general problem of setting up a client to interact with the simulation is not so much a SASS problem but is rather more within the scope of working with Apache Thrift. There are many excellent tutorials on their website on how to do this in a number of different languages.

To get you started, we provide here a basic workflow to setup a rudimentary Python client to control a SASS simulation.

1. Get Apache Thrift.
2. Navigate into the folder containing the RPCServer.thrift file and open it. Add the namespace for your target lanuage. For Python, this has already been done for you.
3. Compile the thrift file into Python with the command *thrift -r –gen py RPCServer.thrift*.

4. Install the Thrift bindings for Python, preferably inside a virtual environment. *pip install thrift*
4. Enter the folder **gen-py** (or move it to a convenient directory).
5. Create an emtpy file named client.py.

Inside the client.py file, you will need to add the following code

```
from thrift import Thrift
from thrift.transport import TSocket
from thrift.transport import TTransport
from thrift.protocol import TBinaryProtocol
from remotesim import RemoteSimulationService
from PIL import Image
from io import BytesIO

def main():
    # Make socket
    transport = TSocket.TSocket('localhost', 9090)

    # Buffering is critical. Raw sockets are very slow
    transport = TTransport.TBufferedTransport(transport)

    # Wrap in a protocol
    protocol = TBinaryProtocol.TBinaryProtocol(transport)

    # Create a client to use the protocol encoder
    client = RemoteSimulationService.Client(protocol)

    # Connect!
    transport.open()

    try:
         x = client.getNextImage()
         img = Image.open(BytesIO(x))
         img.load()
         img.show()
    finally:
         transport.close()

if __name__ == '__main__':
     main()
```

This will create the client and request the next image from the simulation. **By default, the RPC Server will return images as tif-encoded byte strings.** You therefore will need the libtiff library in your target language to decode them. In Python, this can be provided by pillow.

CHAPTER 5

Frequently Asked Questions

Contents

5.1 General

5.1.1 What are the advantages of SASS over other SMLM simulators?

- Complete integration with ImageJ/Fiji.
- Incorporates automated control systems into the simulation environment.
- Allows for dynamic adjustment of the illumination *during* a simulation.
- Easy-to-use script interface via Beanshell and the example scripts.
- Interfaces are available for extending simulation attributes, such as PSF generation, background, fiducial markers, and custom fluorophore photophysics.

5.1.2 What does SASS stand for?

SMLM Acquisition Simulation Software.

(SMLM stands for Single Molecule Localization Microscopy.)

5.2 Running Simulations

5.2.1 How is the coordinate system in SASS defined?

Coordinates in SASS are typically in units of pixels unless otherwise noted in the documentation. **Please note that the origin of the Cartesian coordinate system lies at the center of a pixel, not at a corner.**

5.2.2 How are custom fluorophore position lists formatted?

Customized fluorophore positions are imported into SASS from an externally-generated file that you create yourself. This file should contain two columns (*optionally three*) of comma-separated numerical values (for example, a .csv file). Each row represents the position of one fluorophore; the first column represents the fluorophore's x-position, while the second column represents the fluorophore's y-position. If you want to do 3D simulations, there should be a third column for the z-position. The file should contain no header or comments.

Example

The column labels **should not** be included in the file; they are illustrated here only to indicate which columns correspond to x and y.

x, pixels	y, pixels	(optional) z, arb. units
1.2376	4.2340	0.0000
2.7300	3.7105	0.0000
2.4360	1.2887	0.0000
...	...	...

The units of the values are in pixels, and, **for imports from CSV files only**, the origin is in the upper left-hand corner of the generated image stacks, not the center of the upper left pixel. After import into SASS, there is an implicit subtraction of half a pixel from the x- and y-coordinate values which shifts the coordinate system into the one used by SASS. This is done to preserve the same relative pixel locations when importing from the same file into SASS or ThunderSTORM.

For example, a fluorophore with a position in the CSV file at (15.5, 15.5) will lie at the center of a pixel in ThunderSTORM. To get it to lie at the center of a pixel in SASS, 0.5 is subtracted from each coordinate to make the resulting position (15, 15). Because the origin is at a pixel center in SASS, so to will be this fluorophore's position.

Check out ThunderSTORM for more information.

5.2.3 What are the units for the axial (z) direction?

The units of the values in the z-column of the fluorophore position lists can be any unit that you want, so long as you are consistent in your choice of units for the properties of the various simulation components.

For example, if the you specify the fluorophore z-positions in microns, then you should use microns for the fluorescence wavelength, stage displacement, and other values that require a length.

5.2.4 How is the stage z-displacement property used?

The z-displacement of the stage is used for some 3D point spread functions that depend on the emitter's distance from the coverslip.

- z = 0 corresponds to the coverslip surface.
- Negative z-positions correspond to moving the stage downwards on an inverted microscope. For example, a stage z-position of -2 microns corresponds to a focal volume that is located +2 microns above the coverslip surface.

CHAPTER 6

Javadoc

6.1 ch.epfl.leb.sass.commandline

6.1.1 BeanShellConsole

public class **`BeanShellConsole`** extends PlugInFrame
BeanShell console for execution of SASS simulation scripts

Author Marcel Stefko

Constructors

BeanShellConsole

public **`BeanShellConsole`** (String *title*)
Initialize the new frame

Parameters

- **`title`** – name of the frame

Methods

getInterpreter

public Interpreter **`getInterpreter`** ()

Returns BeanShell interpreter associated with this BeanShellConsole

6.1.2 CommandLineInterface

public final class **`CommandLineInterface`**
Main class of the project, launches the BeanShell script interface.

Author Marcel Stefko

Methods

constructOptions

public static Options **`constructOptions`**()

Returns all understood options for ALICA execution

main

public static void **`main`**(String[] *args*)
Shows help, launches the interpreter and executes scripts according to input args.

Parameters

- **`args`** – input arguments

printWelcomeText

public static void **`printWelcomeText`**(PrintStream *out*)
Reads the welcome_text file and prints it to a PrintStream.

Parameters

- **`out`** – stream to print to

6.2 ch.epfl.leb.sass.ijplugin

6.2.1 App

public class **`App`** extends *ImageJSimulator*
Backend for the FIJI plugin GUI

Author Marcel Stefko

Constructors

App

public **`App`**(*Microscope microscope*, Analyzer *analyzer*, Controller *controller*, int *controller_tickrate*)
Assemble the App from custom components.

Parameters

- **`microscope`** – The microscope to be simulated.

- **analyzer** – An analyzer for processing images from the microscope.
- **controller** – A controller that adjusts the state of the microscope.

Methods

getAnalyzerOutput

public ArrayList<Double> **getAnalyzerOutput**()

getControllerOutput

public ArrayList<Double> **getControllerOutput**()

getControllerSetpoint

public ArrayList<Double> **getControllerSetpoint**()

getControllerTickrate

public int **getControllerTickrate**()

getGeneratorTrueSignal

public ArrayList<Double> **getGeneratorTrueSignal**()

getStatusFrame

public *SimulatorStatusFrame* **getStatusFrame**()
Return the handle for the status frame.

Returns Plots with the simulation history.

setSetpoint

public void **setSetpoint**(double *value*)
Set new setpoint for the controller

Parameters

- **value** – new setpoint value

startSimulating

public void **startSimulating**()
Start continuously generating new images until stopped.

stopSimulating

public void **stopSimulating**()
Stop generating new images.

6.2.2 ButtonGroupUtils

public class **ButtonGroupUtils**
Utilities for working with button groups. See https://stackoverflow.com/questions/201287/how-do-i-get-which-jradiobutton-is-selected-from-a-buttongroup#13232816

Author Kyle M. Douglass

Methods

getSelectedButtonText

public static String **getSelectedButtonText**(ButtonGroup *buttonGroup*)
Determines the label of the current selected button.

Parameters

- **buttonGroup** –

Returns The text label of the selected button.

selectButtonModelFromText

public static void **selectButtonModelFromText**(ButtonGroup *buttonGroup*, String *text*)
Selects the button in a button group based on its text label.

Parameters

- **buttonGroup** –
- **text** – The text label of the desired button to select.

6.2.3 CommandPrompt

public class **CommandPrompt** implements PlugIn
Wrapper for initialization of BeanShell console

Author Marcel Stefko

Constructors

CommandPrompt

public **CommandPrompt**()
Initializes new BeanShell console

Methods

run

public void **run** (String *string*)
Set input and output streams, and print welcome text.

Parameters

- **string** –

6.2.4 GUI

public class **GUI** extends PlugInFrame
Main FIJI plugin frame.

Author Marcel Stefko

Fields

app

App **app**

Constructors

GUI

public **GUI** (String *title*)
Creates new form MainFrame

Parameters

- **title** – title of the window

GUI

public **GUI** ()
Initialize the new frame

Methods

run

public void **run** (String *arg*)
Show the frame and initialize backend.

Parameters

- **arg** –

setApp

public void **setApp** (*App app*)
Set the App which this GUI should control

Parameters

- **app** –

6.2.5 InitializeSimulation

public class **InitializeSimulation** extends java.awt.Dialog
Frame for basic setup of a simulation.

Author Marcel Stefko

Fields

backgroundTifFile

File **backgroundTifFile**

emittersCsvFile

File **emittersCsvFile**

main

GUI **main**

model

Model **model**

Constructors

InitializeSimulation

public **InitializeSimulation** (java.awt.Frame *parent*, boolean *modal*, *GUI main*)
Assemble the frame and display it

Parameters

- **parent** –
- **modal** – should the window be persistent
- **main** – GUI to notify

6.2.6 InteractionWindow

public class **InteractionWindow** extends javax.swing.JFrame

Author stefko

Constructors

InteractionWindow

public **InteractionWindow**(Analyzer *analyzer*, Controller *controller*)
Creates new form InteractionWindow

6.2.7 Model

public class **Model** implements Serializable
Model for the InitializeSimulation window.

Author Kyle M. Douglass

Methods

build

public *Microscope* **build**()
Builds a microscope from the model parameters.

Returns A new microscope built from the model parameters.

getAnalyzerCurrentSelection

public String **getAnalyzerCurrentSelection**()

getBackgroundCurrentSelection

public String **getBackgroundCurrentSelection**()

getBackgroundRandomButtonText

public String **getBackgroundRandomButtonText**()

getBackgroundRandomFeatureSize

public double **getBackgroundRandomFeatureSize**()

getBackgroundRandomMaxValue

public float **getBackgroundRandomMaxValue**()

getBackgroundRandomMinValue

public float **getBackgroundRandomMinValue**()

getBackgroundRandomSeed

public int **getBackgroundRandomSeed**()

getBackgroundTifFile

public String **getBackgroundTifFile**()

getBackgroundTifFileButtonText

public String **getBackgroundTifFileButtonText**()

getBackgroundUniformButtonText

public String **getBackgroundUniformButtonText**()

getBackgroundUniformSignal

public float **getBackgroundUniformSignal**()

getCameraAduPerElectron

public double **getCameraAduPerElectron**()

getCameraBaseline

public int **getCameraBaseline**()

getCameraDarkCurrent

public double **getCameraDarkCurrent**()

getCameraEmGain

public int **getCameraEmGain**()

getCameraNX

public int **getCameraNX**()

getCameraNY

public int **getCameraNY**()

getCameraPixelSize

public double **getCameraPixelSize**()

getCameraQuantumEfficiency

public double **getCameraQuantumEfficiency**()

getCameraReadoutNoise

public double **getCameraReadoutNoise**()

getCameraThermalNoise

public double **getCameraThermalNoise**()

getControllerCurrentSelection

public String **getControllerCurrentSelection**()

getEmitters3DCheckBoxEnabled

public boolean **getEmitters3DCheckBoxEnabled**()

getEmitters3DMaxZ

public double **getEmitters3DMaxZ**()

getEmitters3DMinZ

public double **getEmitters3DMinZ**()

getEmittersCsvFile

public String **getEmittersCsvFile**()

getEmittersCsvFileButtonText

public String **getEmittersCsvFileButtonText**()

getEmittersCurrentSelection

public String **getEmittersCurrentSelection**()

getEmittersGridButtonText

public String **getEmittersGridButtonText**()

getEmittersGridSpacing

public int **getEmittersGridSpacing**()

getEmittersRandomButtonText

public String **getEmittersRandomButtonText**()

getEmittersRandomNumber

public int **getEmittersRandomNumber**()

getFiducialsNumber

public int **getFiducialsNumber**()

getFiducialsSignal

public double **getFiducialsSignal**()

getFluorophoreCurrentSelection

public String **getFluorophoreCurrentSelection**()

getFluorophorePalmText

public String **getFluorophorePalmText**()

getFluorophoreSignal

public double **getFluorophoreSignal**()

getFluorophoreSimpleText

public String **getFluorophoreSimpleText**()

getFluorophoreStormText

public String **getFluorophoreStormText**()

getFluorophoreTBl

public double **getFluorophoreTBl**()

getFluorophoreTOff

public double **getFluorophoreTOff**()

getFluorophoreTOn

public double **getFluorophoreTOn**()

getFluorophoreWavelength

public double **getFluorophoreWavelength**()

getLaserCurrentPower

public double **getLaserCurrentPower**()

getLaserMaxPower

public double **getLaserMaxPower**()

getLaserMinPower

public double **getLaserMinPower**()

getObjectiveMag

public double **getObjectiveMag**()

getObjectiveNa

public double **getObjectiveNa**()

getPalmKA

public double **getPalmKA**()

getPalmKB

public double **getPalmKB**()

getPalmKD1

public double **getPalmKD1**()

getPalmKD2

public double **getPalmKD2**()

getPalmKR1

public double **getPalmKR1**()

getPalmKR2

public double **getPalmKR2**()

getPalmSignal

public double **getPalmSignal**()

getPalmWavelength

public double **getPalmWavelength**()

getPsfCurrentSelection

public String **getPsfCurrentSelection**()

getPsfGaussian2dText

public String **getPsfGaussian2dText**()

getPsfGaussian3dText

public String **getPsfGaussian3dText**()

getPsfGibsonLanniMaxRadius

public int **getPsfGibsonLanniMaxRadius**()

getPsfGibsonLanniNg

public double **getPsfGibsonLanniNg**()

getPsfGibsonLanniNg0

public double **getPsfGibsonLanniNg0**()

getPsfGibsonLanniNi

public double **getPsfGibsonLanniNi**()

getPsfGibsonLanniNi0

public double **getPsfGibsonLanniNi0**()

getPsfGibsonLanniNs

public double **getPsfGibsonLanniNs**()

getPsfGibsonLanniNumBasis

public int **getPsfGibsonLanniNumBasis**()

getPsfGibsonLanniNumSamples

public int **getPsfGibsonLanniNumSamples**()

getPsfGibsonLanniOversampling

public int **getPsfGibsonLanniOversampling**()

getPsfGibsonLanniResPsf

public double **getPsfGibsonLanniResPsf**()

getPsfGibsonLanniResPsfAxial

public double **getPsfGibsonLanniResPsfAxial**()

getPsfGibsonLanniSizeX

public int **getPsfGibsonLanniSizeX**()

getPsfGibsonLanniSizeY

public int **getPsfGibsonLanniSizeY**()

getPsfGibsonLanniSolver

public String **getPsfGibsonLanniSolver**()

getPsfGibsonLanniText

public String **getPsfGibsonLanniText**()

getPsfGibsonLanniTg

public double **getPsfGibsonLanniTg**()

getPsfGibsonLanniTg0

public double **getPsfGibsonLanniTg0**()

getPsfGibsonLanniTi0

public double **getPsfGibsonLanniTi0**()

getStageX

public double **getStageX**()

getStageY

public double **getStageY**()

getStageZ

public double **getStageZ**()

getStormKBl

public double **getStormKBl**()

getStormKDark

public double **getStormKDark**()

getStormKDarkRecovery

public double **getStormKDarkRecovery**()

getStormKDarkRecoveryConstant

public double **getStormKDarkRecoveryConstant**()

getStormKTriplet

public double **getStormKTriplet**()

getStormKTripletRecovery

public double **getStormKTripletRecovery**()

getStormSignal

public double **getStormSignal**()

getStormWavelength

public double **getStormWavelength**()

read

public static *Model* **read**(FileInputStream *fileIn*)
Loads a model from a file.

Parameters

- **fileIn** – The input stream from the file.

setAnalyzerCurrentSelection

public void **setAnalyzerCurrentSelection**(String *text*)

setBackgroundCurrentSelection

public void **setBackgroundCurrentSelection**(String *currentSelection*)

setBackgroundRandomButtonText

public void **setBackgroundRandomButtonText**(String *text*)

setBackgroundRandomFeatureSize

public void **setBackgroundRandomFeatureSize** (double *featureSize*)

setBackgroundRandomMaxValue

public void **setBackgroundRandomMaxValue** (float *maxValue*)

setBackgroundRandomMinValue

public void **setBackgroundRandomMinValue** (float *minValue*)

setBackgroundRandomSeed

public void **setBackgroundRandomSeed** (int *seed*)

setBackgroundTifFile

public void **setBackgroundTifFile** (String *filename*)

setBackgroundTifFileButtonText

public void **setBackgroundTifFileButtonText** (String *text*)

setBackgroundUniformButtonText

public void **setBackgroundUniformButtonText** (String *text*)

setBackgroundUniformSignal

public void **setBackgroundUniformSignal** (float *signal*)

setCameraAduPerElectron

public void **setCameraAduPerElectron** (double *aduPerElectron*)

setCameraBaseline

public void **setCameraBaseline** (int *baseline*)

setCameraDarkCurrent

public void **setCameraDarkCurrent** (double *darkCurrent*)

setCameraEmGain

public void **setCameraEmGain** (int *emGain*)

setCameraNX

public void **setCameraNX** (int *nX*)

setCameraNY

public void **setCameraNY** (int *nY*)

setCameraPixelSize

public void **setCameraPixelSize** (double *pixelSize*)

setCameraQuantumEfficiency

public void **setCameraQuantumEfficiency** (double *quantumEfficiency*)

setCameraReadoutNoise

public void **setCameraReadoutNoise** (double *readoutNoise*)

setCameraThermalNoise

public void **setCameraThermalNoise** (double *thermalNoise*)

setControllerCurrentSelection

public void **setControllerCurrentSelection** (String *text*)

setEmitters3DCheckBoxEnabled

public void **setEmitters3DCheckBoxEnabled** (boolean *enabled*)

setEmitters3DMaxZ

public void **setEmitters3DMaxZ** (double *max*)

setEmitters3DMinZ

public void **setEmitters3DMinZ** (double *min*)

setEmittersCsvFile

public void **setEmittersCsvFile** (String *filename*)

setEmittersCsvFileButtonText

public void **setEmittersCsvFileButtonText** (String *text*)

setEmittersCurrentSelection

public void **setEmittersCurrentSelection** (String *currentSelection*)

setEmittersGridButtonText

public void **setEmittersGridButtonText** (String *text*)

setEmittersGridSpacing

public void **setEmittersGridSpacing** (int *spacing*)

setEmittersRandomButtonText

public void **setEmittersRandomButtonText** (String *text*)

setEmittersRandomNumber

public void **setEmittersRandomNumber** (int *number*)

setFiducialsNumber

public void **setFiducialsNumber** (int *number*)

setFiducialsSignal

public void **setFiducialsSignal** (double *signal*)

setFluorophoreCurrentSelection

public void **setFluorophoreCurrentSelection** (String *text*)

setFluorophorePalmText

public void **setFluorophorePalmText** (String *text*)

setFluorophoreSignal

public void **setFluorophoreSignal** (double *signal*)

setFluorophoreSimpleText

public void **setFluorophoreSimpleText** (String *text*)

setFluorophoreStormText

public void **setFluorophoreStormText** (String *text*)

setFluorophoreTBl

public void **setFluorophoreTBl** (double *tBl*)

setFluorophoreTOff

public void **setFluorophoreTOff** (double *tOff*)

setFluorophoreTOn

public void **setFluorophoreTOn** (double *tOn*)

setFluorophoreWavelength

public void **setFluorophoreWavelength** (double *wavelength*)

setLaserCurrentPower

public void **setLaserCurrentPower** (double *currentPower*)

setLaserMaxPower

public void **setLaserMaxPower** (double *maxPower*)

setLaserMinPower

public void **setLaserMinPower** (double *minPower*)

setObjectiveMag

public void **setObjectiveMag** (double *mag*)

setObjectiveNa

public void **setObjectiveNa** (double *na*)

setPalmKA

public void **setPalmKA** (double *kA*)

setPalmKB

public void **setPalmKB** (double *kB*)

setPalmKD1

public void **setPalmKD1** (double *kD1*)

setPalmKD2

public void **setPalmKD2** (double *kD2*)

setPalmKR1

public void **setPalmKR1** (double *kR1*)

setPalmKR2

public void **setPalmKR2** (double *kR2*)

setPalmSignal

public void **setPalmSignal** (double *signal*)

setPalmWavelength

public void **setPalmWavelength** (double *wavelength*)

setPsfCurrentSelection

public void **setPsfCurrentSelection** (String *text*)

setPsfGaussian2dText

public void **setPsfGaussian2dText** (String *text*)

setPsfGaussian3dText

public void **setPsfGaussian3dText** (String *text*)

setPsfGibsonLanniMaxRadius

public void **setPsfGibsonLanniMaxRadius** (int *maxRadius*)

setPsfGibsonLanniNg

public void **setPsfGibsonLanniNg** (double *ng*)

setPsfGibsonLanniNg0

public void **setPsfGibsonLanniNg0** (double *ng0*)

setPsfGibsonLanniNi

public void **setPsfGibsonLanniNi** (double *ni*)

setPsfGibsonLanniNi0

public void **setPsfGibsonLanniNi0** (double *ni0*)

setPsfGibsonLanniNs

public void **setPsfGibsonLanniNs** (double *ns*)

setPsfGibsonLanniNumBasis

public void **setPsfGibsonLanniNumBasis** (int *numBasis*)

setPsfGibsonLanniNumSamples

public void **setPsfGibsonLanniNumSamples** (int *numSamples*)

setPsfGibsonLanniOversampling

public void **setPsfGibsonLanniOversampling** (int *oversampling*)

setPsfGibsonLanniResPsf

public void **setPsfGibsonLanniResPsf** (double *resPsf*)

setPsfGibsonLanniResPsfAxial

public void **setPsfGibsonLanniResPsfAxial** (double *resPsfAxial*)

setPsfGibsonLanniSizeX

public void **setPsfGibsonLanniSizeX** (int *sizeX*)

setPsfGibsonLanniSizeY

public void **setPsfGibsonLanniSizeY** (int *sizeY*)

setPsfGibsonLanniSolver

public void **setPsfGibsonLanniSolver** (String *solver*)

setPsfGibsonLanniText

public void **setPsfGibsonLanniText** (String *text*)

setPsfGibsonLanniTg

public void **setPsfGibsonLanniTg** (double *tg*)

setPsfGibsonLanniTg0

public void **setPsfGibsonLanniTg0** (double *tg0*)

setPsfGibsonLanniTi0

public void **setPsfGibsonLanniTi0** (double *ti0*)

setStageX

public void **setStageX** (double *x*)

setStageY

public void **setStageY** (double *y*)

setStageZ

public void **setStageZ** (double *z*)

setStormKBl

public void **setStormKBl** (double *kBl*)

setStormKDark

public void **setStormKDark** (double *kDark*)

setStormKDarkRecovery

public void **setStormKDarkRecovery** (double *kDarkRecovery*)

setStormKDarkRecoveryConstant

public void **setStormKDarkRecoveryConstant** (double *kDarkRecoveryConstant*)

setStormKTriplet

public void **setStormKTriplet** (double *kTriplet*)

setStormKTripletRecovery

public void **setStormKTripletRecovery** (double *kTripletRecovery*)

setStormSignal

public void **setStormSignal** (double *signal*)

setStormWavelength

public void **setStormWavelength** (double *wavelength*)

write

public void **write** (FileOutputStream *fileOut*)
Saves the model's state to a file.

Parameters

- **fileOut** – The output stream to the file.

6.2.8 ModelTest

public class **ModelTest**

Author Kyle M. Douglass

Constructors

ModelTest

public **ModelTest**()

Methods

testGetAnalyzerCurrentSelection

public void **testGetAnalyzerCurrentSelection**()
Test of getAnalyzerCurrentSelection method, of class Model.

testGetBackgroundCurrentSelection

public void **testGetBackgroundCurrentSelection**()
Test of getBackgroundCurrentSelection method, of class Model.

testGetBackgroundRandomButtonText

public void **testGetBackgroundRandomButtonText**()
Test of getBackgroundRandomButtonText() {

testGetBackgroundRandomFeatureSize

public void **testGetBackgroundRandomFeatureSize**()
Test of getBackgroundRandomFeatureSize method, of class Model.

testGetBackgroundRandomMaxValue

public void **testGetBackgroundRandomMaxValue**()
Test of getBackgroundRandomMaxValue method, of class Model.

testGetBackgroundRandomMinValue

public void **testGetBackgroundRandomMinValue**()
Test of getBackgroundRandomMinValue method, of class Model.

testGetBackgroundRandomSeed

public void **testGetBackgroundRandomSeed**()
Test of getBackgroundRandomFeatureSize method, of class Model.

testGetBackgroundTifFile

public void **testGetBackgroundTifFile**()
Test of getBackgroundTifFile method, of class Model.

testGetBackgroundTifFileButtonText

public void **testGetBackgroundTifFileButtonText**()
Test of getBackgroundTifFileButtonText method, of class Model.

testGetBackgroundUniformButtonText

public void **testGetBackgroundUniformButtonText**()
Test of getBackgroundUniformButtonText method, of class Model.

testGetBackgroundUniformSignal

public void **testGetBackgroundUniformSignal**()
Test of getBackgroundUniformSignal method, of class Model.

testGetCameraAduPerElectron

public void **testGetCameraAduPerElectron**()
Test of getCameraAduPerElectron method, of class Model.

testGetCameraBaseline

public void **testGetCameraBaseline**()
Test of getCameraBaseline method, of class Model.

testGetCameraDarkCurrent

public void **testGetCameraDarkCurrent**()
Test of getCameraDarkCurrent method, of class Model.

testGetCameraEmGain

public void **testGetCameraEmGain**()
Test of getCameraEmGain method, of class Model.

testGetCameraNX

public void **testGetCameraNX**()
Test of getCameraNX method, of class Model.

testGetCameraNY

public void **testGetCameraNY**()
Test of getCameraNY method, of class Model.

testGetCameraPixelSize

public void **testGetCameraPixelSize**()
Test of getCameraPixelSize method, of class Model.

testGetCameraQuantumEfficiency

public void **testGetCameraQuantumEfficiency**()
Test of getCameraQuantumEfficiency method, of class Model.

testGetCameraReadoutNoise

public void **testGetCameraReadoutNoise**()
Test of getCameraReadoutNoise method, of class Model.

testGetCameraThermalNoise

public void **testGetCameraThermalNoise**()
Test of getCameraThermalNoise method, of class Model.

testGetControllerCurrentSelection

public void **testGetControllerCurrentSelection**()
Test of getControllerCurrentSelection method, of class Model.

testGetEmitters3DCheckBoxEnabled

public void **testGetEmitters3DCheckBoxEnabled**()
Test of getEmitters3DCheckBoxEnabled method, of class Model.

testGetEmitters3DMaxZ

public void **testGetEmitters3DMaxZ**()
Test of getEmitters3DMaxZ method, of class Model.

testGetEmitters3DMinZ

public void **testGetEmitters3DMinZ**()
Test of getEmitters3DMinZ method, of class Model.

testGetEmittersCsvFile

public void **testGetEmittersCsvFile**()
Test of getEmittersCsvFile method, of class Model.

testGetEmittersCsvFileButtonText

public void **testGetEmittersCsvFileButtonText**()
Test of getEmittersCsvFileButtonText method, of class Model.

testGetEmittersCurrentSelection

public void **testGetEmittersCurrentSelection**()
Test of getEmittersCurrentSelection method, of class Model.

testGetEmittersGridButtonText

public void **testGetEmittersGridButtonText**()
Test of getEmittersGridButtonText method, of class Model.

testGetEmittersGridSpacing

public void **testGetEmittersGridSpacing**()
Test of getEmittersGridSpacing method, of class Model.

testGetEmittersRandomButtonText

public void **testGetEmittersRandomButtonText**()
Test of getEmittersRandomButtonText method, of class Model.

testGetEmittersRandomNumber

public void **testGetEmittersRandomNumber**()
Test of getEmittersRandomNumber method, of class Model.

testGetFiducialsNumber

public void **testGetFiducialsNumber**()
Test of getFiducialsNumber method, of class Model.

testGetFiducialsSignal

public void **testGetFiducialsSignal**()
Test of getFiducialsSignal method, of class Model.

testGetFluorophoreCurrentSelection

public void **testGetFluorophoreCurrentSelection**()
Test of getFluorophoreCurrentSelection method, of class Model.

testGetFluorophorePalmText

public void **testGetFluorophorePalmText**()
Test of getFluorophorePalmText method, of class Model.

testGetFluorophoreSignal

public void **testGetFluorophoreSignal**()
Test of getFluorophoreSignal method, of class Model.

testGetFluorophoreSimpleText

public void **testGetFluorophoreSimpleText**()
Test of getFluorophoreSimpleText method, of class Model.

testGetFluorophoreStormText

public void **testGetFluorophoreStormText**()
Test of getFluorophoreStormText method, of class Model.

testGetFluorophoreTBl

public void **testGetFluorophoreTBl**()
Test of getFluorophoreTBl method, of class Model.

testGetFluorophoreTOff

public void **testGetFluorophoreTOff**()
Test of getFluorophoreTOff method, of class Model.

testGetFluorophoreTOn

public void **testGetFluorophoreTOn**()
Test of getFluorophoreTOn method, of class Model.

testGetFluorophoreWavelength

public void **testGetFluorophoreWavelength**()
Test of getFluorophoreWavelength method, of class Model.

testGetLaserCurrentPower

public void **testGetLaserCurrentPower**()
Test of getLaserCurrentPower method, of class Model.

testGetLaserMaxPower

public void **testGetLaserMaxPower**()
Test of getLaserMaxPower method, of class Model.

testGetLaserMinPower

public void **testGetLaserMinPower**()
Test of getLaserMinPower method, of class Model.

testGetObjectiveMag

public void **testGetObjectiveMag**()
Test of getObjectiveMag method, of class Model.

testGetObjectiveNa

public void **testGetObjectiveNa**()
Test of getObjectiveNa method, of class Model.

testGetPalmKA

public void **testGetPalmKA**()
Test of getPalmKA method, of class Model.

testGetPalmKB

public void **testGetPalmKB**()
Test of getPalmKB method, of class Model.

testGetPalmKD1

public void **testGetPalmKD1**()
Test of getPalmKD1 method, of class Model.

testGetPalmKD2

public void **testGetPalmKD2**()
Test of getPalmKD2 method, of class Model.

testGetPalmKR1

public void **testGetPalmKR1**()
Test of getPalmKR1 method, of class Model.

testGetPalmKR2

public void **testGetPalmKR2**()
Test of getPalmKR2 method, of class Model.

testGetPalmSignal

public void **testGetPalmSignal**()
Test of getPalmSignal method, of class Model.

testGetPalmWavelength

public void **testGetPalmWavelength**()
Test of getPalmWavelength method, of class Model.

testGetPsfCurrentSelection

public void **testGetPsfCurrentSelection**()
Test of getPsfCurrentSelection method, of class Model.

testGetPsfGaussian2dText

public void **testGetPsfGaussian2dText**()
Test of getPsfGaussian2dText method, of class Model.

testGetPsfGaussian3dText

public void **testGetPsfGaussian3dText**()
Test of getPsfGaussian3dText method, of class Model.

testGetPsfGibsonLanniMaxRadius

public void **testGetPsfGibsonLanniMaxRadius**()
Test of getPsfGibsonLanniMaxRadius, of class Model.

testGetPsfGibsonLanniNg

public void **testGetPsfGibsonLanniNg**()
Test of getPsfGibsonLanniNg, of class Model.

testGetPsfGibsonLanniNg0

public void **testGetPsfGibsonLanniNg0**()
Test of getPsfGibsonLanniNg0, of class Model.

testGetPsfGibsonLanniNi

public void **testGetPsfGibsonLanniNi**()
Test of getPsfGibsonLanniNi, of class Model.

testGetPsfGibsonLanniNi0

public void **testGetPsfGibsonLanniNi0**()
Test of getPsfGibsonLanniNi0, of class Model.

testGetPsfGibsonLanniNs

public void **testGetPsfGibsonLanniNs**()
Test of getPsfGibsonLanniNs, of class Model.

testGetPsfGibsonLanniNumBasis

public void **testGetPsfGibsonLanniNumBasis**()
Test of getPsfGibsonLanniNumBasis, of class Model.

testGetPsfGibsonLanniNumSamples

public void **testGetPsfGibsonLanniNumSamples**()
Test of getPsfGibsonLanniNumSamples, of class Model.

testGetPsfGibsonLanniOversampling

public void **testGetPsfGibsonLanniOversampling**()
Test of getPsfGibsonLanniOversampling, of class Model.

testGetPsfGibsonLanniResPsf

public void **testGetPsfGibsonLanniResPsf**()
Test of getPsfGibsonLanniResPsf, of class Model.

testGetPsfGibsonLanniResPsfAxial

public void **testGetPsfGibsonLanniResPsfAxial**()
Test of getPsfGibsonLanniResPsfAxial, of class Model.

testGetPsfGibsonLanniSizeX

public void **testGetPsfGibsonLanniSizeX**()
Test of getPsfGibsonLanniSizeX, of class Model.

testGetPsfGibsonLanniSizeY

public void **testGetPsfGibsonLanniSizeY**()
Test of getPsfGibsonLanniSizeY, of class Model.

testGetPsfGibsonLanniSolver

public void **testGetPsfGibsonLanniSolver**()
Test of getPsfGibsonLanniSolver, of class Model.

testGetPsfGibsonLanniTg

public void **testGetPsfGibsonLanniTg**()
Test of getPsfGibsonLanniTg, of class Model.

testGetPsfGibsonLanniTg0

public void **testGetPsfGibsonLanniTg0**()
Test of getPsfGibsonLanniTg0, of class Model.

testGetPsfGibsonLanniTi0

public void **testGetPsfGibsonLanniTi0**()
Test of getPsfGibsonLanniTi0, of class Model.

testGetStageX

public void **testGetStageX**()
Test of getStageX method, of class Model.

testGetStageY

public void **testGetStageY**()
Test of getStageY method, of class Model.

testGetStageZ

public void **testGetStageZ**()
Test of getStageZ method, of class Model.

testGetStormKBl

public void **testGetStormKBl**()
Test of getStormKBl method, class Model.

testGetStormKDark

public void **testGetStormKDark**()
Test of getStormKDark method, class Model.

testGetStormKDarkRecovery

public void **testGetStormKDarkRecovery**()
Test of getStormKDarkRecovery method, class Model.

testGetStormKDarkRecoveryConstant

public void **testGetStormKDarkRecoveryConstant**()
Test of getStormKDarkRecoveryConstant method, class Model.

testGetStormKTriplet

public void **testGetStormKTriplet**()
Test of getStormKTriplet method, class Model.

testGetStormKTripletRecovery

public void **testGetStormKTripletRecovery**()
Test of getStormKTripletRecovery method, class Model.

6.2.9 Server

public class **Server** extends PlugInFrame
The form for configuring the SASS server from within ImageJ.

Author Kyle M. Douglass

Constructors

Server

public **Server**(String *title*)
Creates new form Server

Parameters

- **title** – The title of the form.

Server

public **Server**()
Creates new form Server

Methods

run

public void **run**(String *arg*)
Show the frame and initialize backend.

Parameters

- **arg** –

6.2.10 ServerModel

public class **ServerModel**
Contains the GUI form data for the SASS server.

Author Kyle M. Douglass

Methods

getConfigFile

public String **getConfigFile**()

getPort

public int **getPort**()

getPortTextEnabled

public boolean **getPortTextEnabled**()

getSelectConfigButtonEnabled

public boolean **getSelectConfigButtonEnabled**()

getServer

public *RPCServer* **getServer**()

getSimulationModel

public *Model* **getSimulationModel**()

getStartButtonEnabled

public boolean **getStartButtonEnabled**()

getStopButtonEnabled

public boolean **getStopButtonEnabled**()

setConfigFile

public void **setConfigFile**(String *filename*)

setPort

public void **setPort**(int *port*)

setPortTextEnabled

public void **setPortTextEnabled**(boolean *enabled*)

setSelectConfigButtonEnabled

public void **setSelectConfigButtonEnabled**(boolean *enabled*)

setServer

public void **setServer**(*RPCServer server*)

setSimulationModel

public void **setSimulationModel**(*Model simulationModel*)

setStartButtonEnabled

public void **setStartButtonEnabled**(boolean *enabled*)

setStopButtonEnabled

public void **setStopButtonEnabled**(boolean *enabled*)

6.2.11 SimulatorStatusFrame

public class **SimulatorStatusFrame** extends javax.swing.JFrame

Frame that displays the current status and recent history of the simulation. The layout for the status frame was inspired by Karl Bellve's pgFocus GUI: http://big.umassmed.edu/wiki/index.php/PgFocus

Author Kyle M. Douglass

Fields

SUBPLOT_COUNT

public final int **SUBPLOT_COUNT**

Constructors

SimulatorStatusFrame

public **SimulatorStatusFrame**(String *groundTruthYLabel*, String *analyzerYLabel*, String *setpointYLabel*, String *outputYLabel*)

Creates a new status frame.

Parameters

- **groundTruthYLabel** – The y-axis label for the ground truth signal.
- **analyzerYLabel** – The units output by the analyzer.
- **setpointYLabel** – The units of the controller setpoint.
- **outputYLabel** – The units output by the controller.

Methods

updateGraph

public void **updateGraph**(int *frame*, double *trueCount*, double *estimate*, double *setpoint*, double *laser*)

Adds a single new time point to the plot.

Parameters

- **frame** – The frame number
- **trueCount** – The true number of emitting molecules.
- **estimate** – Analyzer's estimate of the number of emitting molecules.
- **setpoint** – The controller's setpoint value.
- **laser** – The output of the laser.

6.2.12 Worker

class **Worker** extends Thread

Fields

stop

public boolean **stop**

Constructors

Worker

public **Worker** (*App app*, Controller *controller*, Analyzer *active_analyzer*, *ImageS imp*)

Methods

run

public void **run** ()

6.3 ch.epfl.leb.sass.loggers

6.3.1 AbstractLogger

public abstract class **AbstractLogger**
Abstract class for logging simulation results.

Author Kyle M. Douglass

Fields

filename

protected String **filename**
The name of the log file.

performLogging

protected boolean **performLogging**
Determines whether the StateLogger is active or not.

Methods

getFilename

public String **getFilename** ()
Return the current filename for the log file.

Returns The filename of the log file.

getPerformLogging

public boolean **getPerformLogging**()
Indicates whether the logger is active.

Returns A boolean indicating whether the logger is active.

reset

public abstract void **reset**()
Resets the logger to its initial state.

saveLogFile

public abstract void **saveLogFile**()
Saves the state of the logger to a file.

Throws

- **java.io.IOException** –

setFilename

public void **setFilename**(String *inFilename*)
Set the filename for logging the fluorophore state transitions and create the file.

Parameters

- **inFilename** – The full path and filename of the log file

Throws

- **IOException** –

setPerformLogging

public void **setPerformLogging**(boolean *isActive*)
Activates and deactivates the logger.

Parameters

- **isActive** – Indicates whether the logger should be active.

6.3.2 FrameInfo

public class **FrameInfo**
Stores data from the FrameLogger.

Author Kyle M. Douglass

Fields

brightness

public double **brightness**

frame

public int **frame**

id

public int **id**

timeOn

public double **timeOn**

x

public double **x**

y

public double **y**

z

public double **z**

Constructors

FrameInfo

public **FrameInfo**()
Creates a new FrameInfo object with all field values set to zero.

FrameInfo

public **FrameInfo**(int *frame*, int *id*, double *x*, double *y*, double *z*, double *brightness*, double *timeOn*)
Creates a new FrameInfo object from the desired values.

Parameters

- **frame** –
- **id** –

- **x** –
- **y** –
- **z** –
- **brightness** –
- **timeOn** –

6.3.3 FrameLogger

public class **FrameLogger** extends *AbstractLogger*

Reports the positions of all fluorophores visible in each frame in a file The FrameLogger is a singleton.

Author Baptiste Ottino

Methods

getBrightness

public ArrayList<Double> **getBrightness**()

getFrame

public ArrayList<Integer> **getFrame**()

getFrameInfo

public ArrayList<*FrameInfo*> **getFrameInfo**()

Returns all the logged arrays in a single data structure. This method is provided for convenience when all frame information is required.

Returns A FramInfo data structure containing all the logged data.

getId

public ArrayList<Integer> **getId**()

getInstance

public static *FrameLogger* **getInstance**()

Returns An instance of the singleton.

getLogCurrentFrameOnly

public boolean **getLogCurrentFrameOnly**()

Indicates whether only the current frame or all frames are logged.

Returns If return value is true, only information about the current frame is returned.

getTimeOn

public ArrayList<Double> **getTimeOn**()

getX

public ArrayList<Double> **getX**()

getY

public ArrayList<Double> **getY**()

getZ

public ArrayList<Double> **getZ**()

logFrame

public void **logFrame**(int *frame*, int *id*, double *x*, double *y*, double *z*, double *brightness*, double *timeOn*)

Logs emitter information for each full frame. Correct operation of this method when logCurrentFrameOnly is true assumes that values for the frame argument either are the same as previous calls to this method or monotonically increasing.

Parameters

- **frame** – The current frame
- **id** – The emitter's unique ID.
- **x** – x-position of the emitter
- **y** – y-position of the emitter
- **z** – z-position of the emitter
- **brightness** – the apparent brightness of the fluorophore on the frame in number of photons
- **timeOn** – the amount of time the emitter "id" stays on in the current frame

reset

public void **reset**()

Resets the logger to its initial state.

saveLogFile

public void **saveLogFile**()

Saves the state of the logger to a file.

Throws

- **IOException** –

setLogCurrentFrameOnly

public void **setLogCurrentFrameOnly**(boolean *logCurrentFrame*)

Toggles whether only the current frame or all frames should be logged. Setting this to true will erase any information already held by the FrameLogger.

Parameters

- **logCurrentFrame** – If true, only information on the current frame is retained.

6.3.4 FrameLoggerTest

public class **FrameLoggerTest**

Logs all per-frame positions

Author Baptiste Ottino

Fields

tempDir

public TemporaryFolder **tempDir**

Constructors

FrameLoggerTest

public **FrameLoggerTest**()

Methods

setUp

public void **setUp**()

testGetFrameInfo

public void **testGetFrameInfo**()

Test of getFrameInfo method, of class FrameLogger.

testLogFrame

public void **testLogFrame**()

Test of logFrame method, of class FrameLogger.

testReset

public void **testReset**()
Test for resetting the logger to its initial state.

testSaveLogFile

public void **testSaveLogFile**()
Test for saving the log file.

Throws

- **java.io.IOException** –

testSetFilename

public void **testSetFilename**()
Test of setFilename method and unique filename generation.

Throws

- **IOException** –

6.3.5 PositionLogger

public class **PositionLogger** extends *AbstractLogger*
Records the fluorophore positions to a file. Currently, the position logger logs the initial positions of emitters to a file, i.e. it will not track moving fluorophores. This feature may be added in the future if desired. The PositionLogger is a singleton.

Author Kyle M. Douglass

Methods

getIds

public ArrayList<Integer> **getIds**()

getInstance

public static *PositionLogger* **getInstance**()

Returns An instance of the singleton.

getX

public ArrayList<Double> **getX**()

getY

public ArrayList<Double> **getY**()

getZ

public ArrayList<Double> **getZ**()

logPosition

public void **logPosition**(int *id*, double *x*, double *y*, double *z*)
Simple logger for positions and their times.

Parameters

- **id** – The emitter's unique ID.
- **x** – x-position of the emitter
- **y** – y-position of the emitter
- **z** – z-position of the emitter

reset

public void **reset**()
Resets the logger to its initial state.

saveLogFile

public void **saveLogFile**()
Saves the state of the logger to a file.

Throws

- **IOException** –

6.3.6 PositionLoggerTest

public class **PositionLoggerTest**
Logs emitter positions from a simulation.

Author Kyle M. Douglass

Fields

tempDir

public TemporaryFolder **tempDir**

Constructors

PositionLoggerTest

public **PositionLoggerTest**()

Methods

setUp

public void **setUp**()

testLogPosition

public void **testLogPosition**()
Test of logStateTransition method, of class StateLogger.

testReset

public void **testReset**()
Test for resetting the logger to its initial state.

testSaveLogFile

public void **testSaveLogFile**()
Test for saving the log file.

Throws

- **java.io.IOException** –

testSetFilename

public void **testSetFilename**()
Test of setFilename method and unique filename generation.

Throws

- **IOException** –

6.3.7 StateLogger

public class **StateLogger** extends *AbstractLogger*
Records the fluorophore states to a file. The StateLogger is a singleton.

Author Kyle M. Douglass

Methods

getElapsedTimes

public ArrayList<Double> **getElapsedTimes**()

getFilename

public String **getFilename**()
Return the current filename for the log file.

Returns filename

getIds

public ArrayList<Integer> **getIds**()

getInitialStates

public ArrayList<Integer> **getInitialStates**()

getInstance

public static *StateLogger* **getInstance**()

Returns An instance of the singleton.

getNextStates

public ArrayList<Integer> **getNextStates**()

logStateTransition

public void **logStateTransition**(int *id*, double *timeElapsed*, int *initialState*, int *nextState*)
Simple logger for state transitions and their times.

Parameters

- **id** – integer ID of the emitter
- **timeElapsed** – The time spent in the current state
- **initialState** – integer ID of the original state
- **nextState** – integer ID for the new fluorophore state

reset

public void **reset**()
Resets the logger to its initial state.

saveLogFile

public void **saveLogFile**()
Saves the state of the logger to a file.

Throws

- **IOException** –

6.3.8 StateLoggerTest

public class **StateLoggerTest**
Logs all state transitions from a simulation.

Author Kyle M. Douglass

Fields

tempDir

public TemporaryFolder **tempDir**

Constructors

StateLoggerTest

public **StateLoggerTest**()

Methods

setUp

public void **setUp**()

testLogStateTransition

public void **testLogStateTransition**()
Test of logStateTransition method, of class StateLogger.

testReset

public void **testReset**()
Test for resetting the logger to its initial state.

testSaveLogFile

public void **testSaveLogFile**()
Test for saving the log file.

Throws

- **java.io.IOException** –

testSetFilename

public void **testSetFilename**()
Test of setFilename method and unique filename generation.

Throws

- **IOException** –

6.4 ch.epfl.leb.sass.models

6.4.1 Microscope

public class **Microscope**
Integrates all the components into one microscope.

Constructors

Microscope

public **Microscope**(*Camera.Builder cameraBuilder*, *Laser.Builder laserBuilder*, *Objective.Builder objectiveBuilder*, *PSFBuilder psfBuilder*, *Stage.Builder stageBuilder*, *FluorophoreCommandBuilder positionBuilder*, *FluorophoreDynamicsBuilder fluorDynamicsBuilder*, *ObstructorCommandBuilder obstructorBuilder*, *BackgroundCommandBuilder backgroundBuilder*)
Initializes the microscope for simulations.

Parameters

- **cameraBuilder** –
- **laserBuilder** –
- **objectiveBuilder** –
- **psfBuilder** –
- **stageBuilder** –
- **positionBuilder** – Positions fluorophore's within the field of view.
- **fluorDynamicsBuilder** –
- **obstructorBuilder** – Creates the obstructors, e.g. fiducials.
- **backgroundBuilder** – Creates the background signal on the image.

Methods

getFovSize

public double **getFovSize**()

Returns size of current FOV in square micrometers

getLaserPower

public double **getLaserPower**()
Return current power of the laser.

Returns laser power

getObjectSpacePixelSize

public double **getObjectSpacePixelSize**()
The size of a pixel after division by the objective magnification.

Returns Length of one pixel side in object space units

getOnEmitterCount

public double **getOnEmitterCount**()
Returns the number of currently active emitters.

Returns number of shining emitters

getResolution

public int[] **getResolution**()
Return the number of camera pixels in x and y.

Returns 2D array with number of pixels in x and y.

setLaserPower

public void **setLaserPower**(double *laserPower*)
Modifies the laser power to desired value.

Parameters

- **laserPower** – new laser power

simulateFrame

public *ImageS* **simulateFrame**()
Generates a new frame and moves the device state forward. First the obstructors are drawn on the frame, then the fluorophores, and finally noise.

Returns simulated frame

6.5 ch.epfl.leb.sass.models.backgrounds

6.5.1 BackgroundCommand

public interface **BackgroundCommand**
Commands for creating a background in an image.

Author Kyle M. Douglass

Methods

generateBackground

public float[][] **generateBackground**()

6.5.2 BackgroundCommandBuilder

public interface **BackgroundCommandBuilder**
Interface BackgroundCommand builders.

Author Kyle M. Douglass

Methods

build

public *BackgroundCommand* **build**()

nX

public *BackgroundCommandBuilder* **nX**(int *nX*)
Sets the number of pixels of the images in the x-direction.

Parameters

- **nX** – Number of pixels in x.

Returns The very same builder object.

nY

public *BackgroundCommandBuilder* **nY**(int *nY*)
Sets the number of pixels of the images in the y-direction.

Parameters

- **nY** – Number of pixels in y.

Returns The very same builder object.

6.6 ch.epfl.leb.sass.models.backgrounds.internal.commands

6.6.1 GenerateBackgroundFromFile

public final class **`GenerateBackgroundFromFile`** implements *BackgroundCommand*
Constant overlay loaded from a tif image.

Author Marcel Stefko

Methods

generateBackground

public float[][] **`generateBackground`**()
Creates the background image.

Returns The background image.

6.6.2 GenerateBackgroundFromFile.Builder

public static class **`Builder`** implements *BackgroundCommandBuilder*

Methods

build

public *GenerateBackgroundFromFile* **`build`**()

file

public *Builder* **`file`**(File *file*)

nX

public *Builder* **`nX`**(int *nX*)

nY

public *Builder* **`nY`**(int *nY*)

6.6.3 GenerateBackgroundFromFileTest

public class **`GenerateBackgroundFromFileTest`**
Tests for generating a constant background from a .tif file.

Author Kyle M. Douglass

Fields

tempDir

public TemporaryFolder **tempDir**

Constructors

GenerateBackgroundFromFileTest

public **GenerateBackgroundFromFileTest** ( )

Methods

setUp

public void **setUp** ( )
Creates a test .tif file as an example background.

testGenerateBackground

public void **testGenerateBackground** ( )
Test of generateBackground method, of class GenerateBackgroundFromFile.

6.6.4 GenerateRandomBackground

public class **GenerateRandomBackground** implements *BackgroundCommand*
Generates random background patterns from a simplex noise generator.

Author Kyle M. Douglass

Methods

generateBackground

public float[][] **generateBackground** ( )
Create the random background signal.

Returns A 2D array of background photons for each pixel.

6.6.5 GenerateRandomBackground.Builder

public static class **Builder** implements *BackgroundCommandBuilder*

Methods

build

public *GenerateRandomBackground* **build**()

featureSize

public *Builder* **featureSize**(double *featureSize*)

max

public *Builder* **max**(float *max*)

min

public *Builder* **min**(float *min*)

nX

public *Builder* **nX**(int *nX*)

nY

public *Builder* **nY**(int *nY*)

seed

public *Builder* **seed**(int *seed*)

6.6.6 GenerateRandomBackgroundTest

public class **GenerateRandomBackgroundTest**

Author Kyle M. Douglass

Constructors

GenerateRandomBackgroundTest

public **GenerateRandomBackgroundTest**()

Methods

testGenerateBackground

public void **testGenerateBackground**()
Test of generateBackground method, of class GenerateRandomBackground.

6.6.7 GenerateUniformBackground

public final class **GenerateUniformBackground** implements *BackgroundCommand*

Author Kyle M. Douglass

Methods

generateBackground

public float[][] **generateBackground**()
Create the background signal.

Returns A 2D array of background photons for each pixel.

6.6.8 GenerateUniformBackground.Builder

public static class **Builder** implements *BackgroundCommandBuilder*
Creates the command to generate a uniform background.

Methods

backgroundSignal

public *Builder* **backgroundSignal**(float *backgroundSignal*)

build

public *GenerateUniformBackground* **build**()
Builds the command.

Returns The command to build a uniform background.

nX

public *Builder* **nX**(int *nX*)

nY

public *Builder* **nY**(int *nY*)

6.6.9 OpenSimplexNoise

public class **OpenSimplexNoise**

Constructors

OpenSimplexNoise

public **OpenSimplexNoise**()

OpenSimplexNoise

public **OpenSimplexNoise**(short[] *perm*)

OpenSimplexNoise

public **OpenSimplexNoise**(long *seed*)

Methods

eval

public double **eval**(double *x*, double *y*)

eval

public double **eval**(double *x*, double *y*, double *z*)

eval

public double **eval**(double *x*, double *y*, double *z*, double *w*)

6.7 ch.epfl.leb.sass.models.components

6.7.1 Camera

public final class **Camera**
Represents the parameters of the camera.

Methods

getAduPerElectron

public double **getAduPerElectron**()

getBaseline

public int **getBaseline**()

getDarkCurrent

public double **getDarkCurrent**()

getEmGain

public int **getEmGain**()

getNX

public int **getNX**()

Returns The number of pixels in x.

getNY

public int **getNY**()

Returns The number of pixels in y.

getPixelSize

public double **getPixelSize**()

getQuantumEfficiency

public double **getQuantumEfficiency**()

getReadoutNoise

public double **getReadoutNoise**()

getThermalNoise

public double **getThermalNoise**()

6.7.2 Camera.Builder

public static class **Builder**

Methods

aduPerElectron

public *Builder* **aduPerElectron**(double *aduPerElectron*)

baseline

public *Builder* **baseline**(int *baseline*)

build

public *Camera* **build**()

darkCurrent

public *Builder* **darkCurrent**(double *darkCurrent*)

emGain

public *Builder* **emGain**(int *emGain*)

nX

public *Builder* **nX**(int *nX*)

nY

public *Builder* **nY**(int *nY*)

pixelSize

public *Builder* **pixelSize**(double *pixelSize*)

quantumEfficiency

public *Builder* **quantumEfficiency**(double *quantumEfficiency*)

readoutNoise

public *Builder* **readoutNoise**(double *readoutNoise*)

thermalNoise

public *Builder* **thermalNoise**(double *thermalNoise*)

6.7.3 CameraTest

public class **CameraTest**
Unit tests for the Camera class.

Author Kyle M. Douglass

Constructors

CameraTest

public **CameraTest**()

Methods

testGetAduPerElectron

public void **testGetAduPerElectron**()
Test of getAduPerElectron method, of class Camera.

testGetBaseline

public void **testGetBaseline**()
Test of getBaseline method, of class Camera.

testGetDarkCurrent

public void **testGetDarkCurrent**()
Test of getDarkCurrent method, of class Camera.

testGetEmGain

public void **testGetEmGain**()
Test of getEmGain method, of class Camera.

testGetNX

public void **testGetNX**()
Test of getNX method, of class Camera.

testGetNY

public void **testGetNY**()
Test of getNY method, of class Camera.

testGetPixelSize

public void **testGetPixelSize**()
Test of getPixelSize method, of class Camera.

testGetQuantumEfficiency

public void **testGetQuantumEfficiency**()
Test of getQuantumEfficiency method, of class Camera.

testGetReadoutNoise

public void **testGetReadoutNoise**()
Test of getReadoutNoise method, of class Camera.

testGetThermalNoise

public void **testGetThermalNoise**()
Test of getThermalNoise method, of class Camera.

6.7.4 Laser

public class **Laser**
A source of light for illuminating the sample.

Methods

getPower

public double **getPower**()
Returns the current power.

Returns current laser power

setPower

public void **setPower**(double *newPower*)
Sets the light source's power. If the value is not within the limits, set it to the the closest allowed value.

Parameters

- **newPower** – The power of the light source.

6.7.5 Laser.Builder

public static class **Builder**

Methods

build

public *Laser* **build**()

currentPower

public *Builder* **currentPower**(double *currentPower*)

maxPower

public *Builder* **maxPower**(double *maxPower*)

minPower

public *Builder* **minPower**(double *minPower*)

6.7.6 LaserTest

public class **LaserTest**

Author kmdouglass

Constructors

LaserTest

public **LaserTest**()

Methods

setUp

public void **setUp**()

testGetPower

public void **testGetPower**()
Test of getPower method, of class Laser.

testSetPower

public void **testSetPower**()
Test of setPower method, of class Laser.

6.7.7 Objective

public final class **Objective**
Properties related to the microscope objective.

Author Kyle M. Douglass

Methods

airyFWHM

public double **airyFWHM**(double *wavelength*)
Computes the full width at half maximum of the Airy disk. Units are the same as those of wavelength.

Parameters

- **wavelength** –

Returns Full width at half maximum size of the Airy disk.

airyRadius

public double **airyRadius**(double *wavelength*)
Computes the radius of the Airy disk. Units are the same as those of wavelength.

Parameters

- **wavelength** –

Returns Distance from center of Airy disk to first minimum.

getMag

public double **getMag**()

Returns The objective's magnification.

getNA

public double **getNA**()

Returns The objective' numerical aperture

6.7.8 Objective.Builder

public static class **Builder**

Methods

NA

public *Builder* **NA**(double *NA*)

build

public *Objective* **build**()

mag

public *Builder* **mag**(double *mag*)

6.7.9 ObjectiveTest

public class **ObjectiveTest**

Author Kyle M. Douglass

Constructors

ObjectiveTest

public **ObjectiveTest**()

Methods

testAiryFWHM

public void **testAiryFWHM**()
Test of psfFWHM method, of class Objective.

testAiryRadius

public void **testAiryRadius**()
Test of psfFWHM method, of class Objective.

6.7.10 Stage

public class **Stage**
The sample stage.

Author Kyle M. Douglass

Methods

getX

public double **getX**()

Returns The stage's x-position.

getY

public double **getY**()

Returns The stage's y-position.

getZ

public double **getZ**()

Returns The stage's z-position.

setX

public void **setX**(double *x*)
Set the stage's z-position.

Parameters

- **x** –

setY

public void **setY**(double *y*)
Set the stage's y-position.

Parameters

- **y** –

setZ

public void **setZ**(double *z*)
Set the stage's z-position.

Parameters

- **z** –

6.7.11 Stage.Builder

public static class **Builder**
Builder for creating stage instances.

Methods

build

public *Stage* **build**()

x

public *Builder* **x**(double *x*)

y

public *Builder* **y**(double *y*)

z

public *Builder* **z**(double *z*)

6.7.12 StageTest

public class **StageTest**

Author kmdouglass

Constructors

StageTest

public **StageTest**()

Methods

setUp

public void **setUp**()

testGetX

public void **testGetX**()
Test of getX method, of class Stage.

testGetY

public void **testGetY**()
Test of getY method, of class Stage.

testGetZ

public void **testGetZ**()
Test of getZ method, of class Stage.

testsetX

public void **testsetX**()
Test of setX method, of class Stage.

testsetY

public void **testsetY**()
Test of setY method, of class Stage.

testsetZ

public void **testsetZ**()
Test of setZ method, of class Stage.

6.8 ch.epfl.leb.sass.models.emitters

6.8.1 AbstractEmitterTest

public class **AbstractEmitterTest**

Author douglass

Constructors

AbstractEmitterTest

public **AbstractEmitterTest**()

Methods

testGetPixelsWithinRadiusLessThanOne

public void **testGetPixelsWithinRadiusLessThanOne**()
Test of getPixelsWithinRadius method, of class Camera. Tests that only the pixel containing the point is returned if the radius is less than one.

testGetPixelsWithinRadiusOfOrigin

public void **testGetPixelsWithinRadiusOfOrigin**()
Test of getPixelsWithinRadius method, of class Camera. Tests that all pixels within a certain radius of the origin are correctly returned.

6.9 ch.epfl.leb.sass.models.emitters.internal

6.9.1 AbstractEmitter

public abstract class **AbstractEmitter** extends Point2D.Double
A point source of light and tools to compute its signature on a digital detector. Emitters are general point sources of light that are imaged by an optical system and recorded by a digital sensor. The AbstractEmitter class contains tools for generating the digital images of point sources without any regard for the dynamics of the of the signal (apart from photon shot noise). Classes that extend the AbstractEmitter class are intended to implement the dynamics of the source's signal.

Author Marcel Stefko, Kyle M. Douglass

Fields

builder

protected *PSFBuilder* **builder**
A builder for creating/updating the emitter PSF.

camera

protected final *Camera* **camera**
Camera settings used for calculating PSF

frameLogger

protected final *FrameLogger* **frameLogger**
A copy of the frame logger.

id

protected int **id**
A unique ID assigned to this emitter.

numberOfEmitters

protected static int **numberOfEmitters**
Running total of the number of emitters.

pixel_list

protected ArrayList<*Pixel*> **pixel_list**
List of pixels which are affected by this emitter's light (these pixels need to be updated when the emitter is on).

poisson

protected Poisson **poisson**
Poisson RNG for flickering simulation.

positionLogger

protected final *PositionLogger* **positionLogger**
A copy of the position logger.

psf

protected *PSF* **psf**
The PSF model that's created by the emitter.

stateLogger

protected final *StateLogger* **stateLogger**
A copy of the state logger.

z

public double **z**
The emitter's z-position.

Constructors

AbstractEmitter

public **AbstractEmitter**(*Camera camera*, double *x*, double *y*)
Creates emitter at given position, and calculates its signature on the image (what does it look like when it is turned on).

Parameters

- **camera** – camera properties (needed for PSF calculation)
- **x** – x-position in image [pixels, with sub-pixel precision]
- **y** – y-position in image [pixels, with sub-pixel precision]

AbstractEmitter

public **AbstractEmitter**(double *x*, double *y*, double *z*, *PSFBuilder psfBuilder*)

Creates the emitter at given position, and calculates its image from the PSF and camera.

Parameters

- **x** – x-position in image [pixels, with sub-pixel precision]
- **y** – y-position in image [pixels, with sub-pixel precision]
- **z** – z-position in image [pixels, with sub-pixel precision]
- **psfBuilder** – Builder for creating the emitter's PSF.

Methods

applyTo

public void **applyTo**(float[][] *pixels*)

Simulates the brightness pattern of this emitter for the next frame duration, and renders the emitter onto the image.

Parameters

- **pixels** – image to be drawn on

flicker

protected double **flicker**(double *baseBrightness*)

Applies Poisson statistics to simulate flickering of an emitter.

Parameters

- **baseBrightness** – mean of Poisson distribution to draw from

Returns actual brightness of this emitter for this frame

generate_signature_for_pixel

protected double **generate_signature_for_pixel**(int *x*, int *y*, double *camera_fwhm_digital*)

Returns the signature that this emitter leaves on a given pixel (what fraction of this emitter's photons hits this particular pixel).

Parameters

- **x** – pixel x-position
- **y** – pixel y-position
- **camera_fwhm_digital** – camera fwhm value

Throws

- **MathException** –

Returns signature value for this pixel

getId

public int **getId**()
Returns the emitter's ID.

Returns The unique integer identifying the emitter.

getPSF

public *PSF* **getPSF**()
Returns the emitter's PSF model.

Returns The PSF model used to create the image of this emitter.

getPixelList

public ArrayList<*Pixel*> **getPixelList**()
Returns list of pixels which need to be drawn on the image to accurately render the emitter.

Returns list of Pixels

getPixelsWithinRadius

public static final ArrayList<*Pixel*> **getPixelsWithinRadius**(Point2D *point*, double *radius*)
Returns a list of pixels within a certain radius from a point. This method locates all the pixels within a circular area surrounding a given two-dimensional point whose center lies at (x, y). The coordinate of a pixel is assumed to lie at the pixel's center, and a pixel is within a given radius of another if the pixel's center lies within this circle.

Parameters

- **point** –
- **radius** – radius value [pixels]

Returns list of Pixels with pre-calculated signatures

get_pixels_within_radius

protected final ArrayList<*Pixel*> **get_pixels_within_radius**(double *radius*, double *camera_fwhm_digital*)
Returns a list of pixels within a certain radius from this emitter (so that their signature is precalculated). Pixels outside this radius are considered to have negligible signature.

Parameters

- **radius** – radius value [pixels]
- **camera_fwhm_digital** – camera fwhm value

Returns list of Pixels with precalculated signatures

setPSF

public void **setPSF** (*PSF psf*)
Change the emitter's PSF model.

Parameters

- **psf** – The PSF model used to create the image of this emitter.

simulateBrightness

protected abstract double **simulateBrightness** ()
Simulates the state evolution of the emitter for the next frame, and returns the integrated brightness of this emitter for this frame.

Returns brightness of emitter in this frame [photons emitted]

6.9.2 Pixel

public class **Pixel**
Representation of a single pixel signature caused by a single emitter.

Author Marcel Stefko

Fields

x

public final int **x**
X-position of pixel in image.

y

public final int **y**
Y-position of pixel in image.

Constructors

Pixel

public **Pixel** (int *x*, int *y*, double *signature*)
Initialize new pixel with position and signature.

Parameters

- **x** – x-position [px]
- **y** – y-position [px]
- **signature** – relative brightness of this pixel due to emitter [-]

Methods

distance_to

public double **distance_to**(*Pixel p*)
Calculates euclidean distance to another pixel

Parameters

- **p** – another Pixel

Returns euclidean distance between these pixels [px]

distance_to_sq

public int **distance_to_sq**(*Pixel p*)
Calculates squared distance to another pixel

Parameters

- **p** – another Pixel

Returns squared distance between these pixels [px2]

getSignature

public double **getSignature**()
Returns this pixel's signature

Returns relative brightness of this pixel due to an emitter [-]

setSignature

public void **setSignature**(double *signature*)
Set's the pixel's signature.

6.10 ch.epfl.leb.sass.models.fluorophores

6.10.1 Fluorophore

public interface **Fluorophore**
A single fluorophore including its position and photophysical properties.

Author Kyle M. Douglass

Methods

isBleached

public boolean **isBleached**()
Has the fluorophore been bleached? If so, it can never return to a fluorescence-emitting state.

Returns A true/false value describing whether the fluorophore is bleached.

isOn

public boolean **isOn**()
Describes whether the fluorophore is emitting light or is in a dark state.

Returns A true/false value describing whether the fluorophore is emitting.

recalculateLifetimes

public void **recalculateLifetimes**(double *laserPower*)
This method recalculates the lifetimes of the fluorophore's state system based on the laser power.

Parameters

- **laserPower** – The new power of the laser.

6.11 ch.epfl.leb.sass.models.fluorophores.internal

6.11.1 DefaultFluorophore

public class **DefaultFluorophore** extends *AbstractEmitter* implements *Fluorophore*
A general fluorescent molecule which emits light.

Author Marcel Stefko

Constructors

DefaultFluorophore

public **DefaultFluorophore**(*Camera camera*, double *signal*, *StateSystem state_system*, int *start_state*, double *x*, double *y*)
Initialize fluorophore and calculate its pattern on camera

Parameters

- **camera** – Camera used for calculating diffraction pattern
- **signal** – No of photons per frame.
- **state_system** – Internal state system for this fluorophore
- **start_state** – Initial state number
- **x** – x-position in pixels
- **y** – y-position in pixels

DefaultFluorophore

public **DefaultFluorophore**(*PSFBuilder psfBuilder*, double *signal*, *StateSystem state_system*, int *start_state*, double *x*, double *y*, double *z*)

Initialize fluorophore and calculate its pattern on camera

Parameters

- **psfBuilder** – The Builder for calculating microscope PSFs.
- **signal** – Number of photons per frame.
- **state_system** – Internal state system for this fluorophore
- **start_state** – Initial state number
- **x** – x-position in pixels
- **y** – y-position in pixels
- **z** – z-position in pixels

Methods

isBleached

public boolean **isBleached**()

Informs if this emitter switched into the irreversible bleached state.

Returns boolean, true if emitter is bleached

isOn

public boolean **isOn**()

Returns the current state of the emitter (on or off), but does not inform if this emitter is also bleached!

Returns true-emitter is on, false-emitter is off

nextExponential

protected final double **nextExponential**(double *mean*)

Sample an random number from an exponential distribution

Parameters

- **mean** – mean of the distribution

Returns random number from this distribution

recalculateLifetimes

public void **recalculateLifetimes**(double *laserPower*)

Recalculates the lifetimes of this emitter based on current laser power.

Parameters

- **laserPower** – current laser power

simulateBrightness

protected double **simulateBrightness**()

6.11.2 DefaultFluorophoreTest

public class **DefaultFluorophoreTest**

Author Kyle M. Douglass

Constructors

DefaultFluorophoreTest

public **DefaultFluorophoreTest**()

Methods

setUp

public void **setUp**()

testFluorophoreIdAssignment

public void **testFluorophoreIdAssignment**()
Test that fluorophores are assigned their proper IDs in successive order.

6.11.3 StateSystem

public class **StateSystem**
Class which describes a Markovian fluorophore state model. This class provides transition rates and mean lifetimes for Markovian models based on current laser illumination intensity.

Author stefko

Fields

current_laser_power

protected double **current_laser_power**
Laser power value for which the currently stored lifetime values are calculated.

Constructors

StateSystem

public **StateSystem**(int *N_states*, double[][][] *M_scaling*)
Initialize the state system.

Parameters

- **N_states** – number of states
- **M_scaling** – double[][][] matrix of dimensions N x N x A. A can be different for each position in the matrix. This matrix can be interpreted as follows: double[] P = M_scaling[i][j]; k_ij(I) = P[0] + P[1]*I + P[2]*I^2 + ... P[n]*I^n; k_ij(I) is transition rate between i-th and j-th state under laser illumination intensity I. The first row of this matrix is considered the active state, the last row is considered the bleached state.

Methods

getMeanTransitionLifetime

public final double **getMeanTransitionLifetime**(int *from*, int *to*)

Parameters

- **from** – index of initial state
- **to** – index of final state

Returns mean transition lifetime from one state to another

getNStates

public int **getNStates**()

Returns number of states of this model

getTransitionRate

public final double **getTransitionRate**(int *from*, int *to*)

Parameters

- **from** – index of initial state
- **to** – index of final state

Returns transition rate from one state to another

isBleachedState

public boolean **isBleachedState**(int *state*)
Returns true if the state is the bleached state (the last state of the model)

Parameters

- **state** – id of current state

Returns state == (N_states - 1)

isOnState

public boolean **isOnState**(int *state*)
Returns true if the state is the active state (the 0-th state)

Parameters

- **state** – id of current state

Returns (state==0)

recalculate_lifetimes

public final void **recalculate_lifetimes**(double *laser_power*)
Recalculates each element of the transition matrix, based on the scaling matrix provided at initialization. double[] P = M_scaling[i][j]; k_ij(I) = P[0] + P[1]*I + P[2]*I^2 + ... P[n]*I^n; k_ij(I) is transition rate between i-th and j-th state under laser illumination intensity I.

Parameters

- **laser_power** – illumination intensity I to recalculate for

6.12 ch.epfl.leb.sass.models.fluorophores.internal.commands

6.12.1 FluorophoreCommand

public interface **FluorophoreCommand**
Executes a command for generating fluorophores.

Author Kyle M. Douglass

Methods

generateFluorophores

public List<*DefaultFluorophore*> **generateFluorophores**()

6.12.2 FluorophoreCommandBuilder

public interface **FluorophoreCommandBuilder**
Interface for populating the field with fluorophores.

Author Kyle M. Douglass

Methods

build

public *FluorophoreCommand* **build**()

camera

public *FluorophoreCommandBuilder* **camera** (*Camera camera*)

fluorDynamics

public *FluorophoreCommandBuilder* **fluorDynamics** (*FluorophoreDynamics fluorDynamics*)

psfBuilder

public *FluorophoreCommandBuilder* **psfBuilder** (*PSFBuilder psfBuilder*)

6.12.3 FluorophoreReceiver

public class **FluorophoreReceiver**

Populates a field of view with fluorophores. The FluorophoreGenerator contains a number of methods for creating actual fluorophore instances and in different arrangements, such as placing them on a grid, randomly distributing them in the FOV, and placing them according to input from a text file.

Author Marcel Stefko, Kyle M. Douglass

Methods

generateFluorophoresFromCSV

public static ArrayList<*DefaultFluorophore*> **generateFluorophoresFromCSV** (File *file*, *Camera camera*, *PSFBuilder psfBuilder*, *FluorophoreDynamics fluorDynamics*, boolean *rescale*)

Parse a CSV file and generate fluorophores from it.

Parameters

- **file** – The CSV file. If this is null, then a dialog is opened.
- **camera** – The camera for determining the size of the field of view.
- **psfBuilder** – Builder for calculating microscope PSFs.
- **fluorDynamics** – The fluorophore dynamics properties.
- **rescale** – if true, positions are rescaled to fit into frame, otherwise positions outside of frame are cropped

Throws

- **IOException** –
- **FileNotFoundException** –

Returns list of fluorophores.

generateFluorophoresGrid2D

public static ArrayList<*DefaultFluorophore*> **generateFluorophoresGrid2D**(int *spacing*, *Camera camera*, *PSFBuilder psfBuilder*, *FluorophoreDynamics fluorDynamics*)

Generate a rectangular grid of fluorophores.

Parameters

- **spacing** – The distance along the grid between nearest neighbors.
- **camera** – The camera for determining the size of the field of view.
- **psfBuilder** – Builder for calculating microscope PSFs.
- **fluorDynamics** – The fluorophore dynamics properties.

Returns The list of fluorophores.

generateFluorophoresGrid3D

public static ArrayList<*DefaultFluorophore*> **generateFluorophoresGrid3D**(int *spacing*, double *zLow*, double *zHigh*, *Camera camera*, *PSFBuilder psfBuilder*, *FluorophoreDynamics fluorDynamics*)

Create fluorophores on a 2D grid and step-wise in the axial direction.

Parameters

- **spacing** – The distance along the grid between nearest neighbors.
- **zLow** – The lower bound on the range in z in units of pixels.
- **zHigh** – The upper bound on the range in z in units of pixels.
- **camera** – The camera for determining the size of the field of view.
- **psfBuilder** – Builder for calculating microscope PSFs.
- **fluorDynamics** – The fluorophore dynamics properties.

Returns The list of fluorophores.

generateFluorophoresRandom2D

public static ArrayList<*DefaultFluorophore*> **generateFluorophoresRandom2D**(int *numFluors*, *Camera camera*, *PSFBuilder psfBuilder*, *FluorophoreDynamics fluorDynamics*)

Randomly populate the field of view with fluorophores.

Parameters

- **numFluors** – The number of fluorophores to add to the field of view.
- **camera** – The camera for determining the size of the field of view.

- **psfBuilder** – Builder for calculating microscope PSFs.
- **fluorDynamics** – The fluorophore dynamics properties.

Returns The list of fluorophores.

generateFluorophoresRandom3D

public static ArrayList<*DefaultFluorophore*> **generateFluorophoresRandom3D**(int *numFluors*, double *zLow*, double *zHigh*, *Camera* *camera*, *PSFBuilder* *psfBuilder*, *FluorophoreDynamics* *fluorDynamics*)

Randomly populate the field of view with fluorophores in three dimensions.

Parameters

- **numFluors** – The number of fluorophores to add to the field of view.
- **zLow** – The lower bound on the range in z in units of pixels
- **zHigh** – The upper bound on the range in z in units of pixels
- **camera** – The camera for determining the size of the field of view.
- **psfBuilder** – Builder for calculating microscope PSFs.
- **fluorDynamics** – The fluorophore dynamics properties.

Returns The list of fluorophores.

6.12.4 GenerateFluorophoresFromCSV

public final class **GenerateFluorophoresFromCSV** implements *FluorophoreCommand*

This serves as the Invoker of a DefaultFluorophore command.

Author Kyle M.Douglass

Methods

generateFluorophores

public List<*DefaultFluorophore*> **generateFluorophores**()

Executes the command that generates the fluorophores.

Returns The list of Fluorophores.

6.12.5 GenerateFluorophoresFromCSV.Builder

public static class **Builder** implements *FluorophoreCommandBuilder*

A builder for creating this command for fluorophore generation.

Methods

build

public *FluorophoreCommand* **build**()

camera

public *Builder* **camera**(*Camera camera*)

file

public *Builder* **file**(File *file*)

fluorDynamics

public *Builder* **fluorDynamics**(*FluorophoreDynamics fluorDynamics*)

psfBuilder

public *Builder* **psfBuilder**(*PSFBuilder psfBuilder*)

rescale

public *Builder* **rescale**(boolean *rescale*)

6.12.6 GenerateFluorophoresGrid2D

public final class **GenerateFluorophoresGrid2D** implements *FluorophoreCommand*
This serves as the Invoker of a DefaultFluorophore command.

Author Kyle M.Douglass

Methods

generateFluorophores

public List<*DefaultFluorophore*> **generateFluorophores**()
Executes the command that generates the fluorophores.

Returns The list of fluorophores.

6.12.7 GenerateFluorophoresGrid2D.Builder

public static class **Builder** implements *FluorophoreCommandBuilder*
A builder for creating this command for fluorophore generation.

Methods

build

public *FluorophoreCommand* **build**()

camera

public *Builder* **camera**(*Camera camera*)

fluorDynamics

public *Builder* **fluorDynamics**(*FluorophoreDynamics fluorDynamics*)

psfBuilder

public *Builder* **psfBuilder**(*PSFBuilder psfBuilder*)

spacing

public *Builder* **spacing**(int *spacing*)

6.12.8 GenerateFluorophoresGrid3D

public final class **GenerateFluorophoresGrid3D** implements *FluorophoreCommand*
This serves as the Invoker of a DefaultFluorophore command.

Author Kyle M.Douglass

Methods

generateFluorophores

public List<*DefaultFluorophore*> **generateFluorophores**()
Executes the command that generates the fluorophores.

Returns The list of Fluorophores.

6.12.9 GenerateFluorophoresGrid3D.Builder

public static class **Builder** implements *FluorophoreCommandBuilder*
A builder for creating this command for fluorophore generation.

Methods

build

public *FluorophoreCommand* **build**()

camera

public *Builder* **camera**(*Camera camera*)

fluorDynamics

public *Builder* **fluorDynamics**(*FluorophoreDynamics fluorDynamics*)

psfBuilder

public *Builder* **psfBuilder**(*PSFBuilder psfBuilder*)

spacing

public *Builder* **spacing**(int *spacing*)

zHigh

public *Builder* **zHigh**(double *zHigh*)

zLow

public *Builder* **zLow**(double *zLow*)

6.12.10 GenerateFluorophoresRandom2D

public final class **GenerateFluorophoresRandom2D** implements *FluorophoreCommand*

This serves as the Invoker of a DefaultFluorophore command.

Author Kyle M.Douglass

Methods

generateFluorophores

public List<*DefaultFluorophore*> **generateFluorophores**()

Executes the command that generates the fluorophores.

Returns The list of fluorophores.

6.12.11 GenerateFluorophoresRandom2D.Builder

public static class **Builder** implements *FluorophoreCommandBuilder*
A builder for creating this command for fluorophore generation.

Methods

build

public *FluorophoreCommand* **build**()

camera

public *Builder* **camera**(*Camera camera*)

fluorDynamics

public *Builder* **fluorDynamics**(*FluorophoreDynamics fluorDynamics*)

numFluors

public *Builder* **numFluors**(int *numFluors*)

psfBuilder

public *Builder* **psfBuilder**(*PSFBuilder psfBuilder*)

6.12.12 GenerateFluorophoresRandom3D

public final class **GenerateFluorophoresRandom3D** implements *FluorophoreCommand*
This serves as the Invoker of a DefaultFluorophore command.

Author Kyle M.Douglass

Methods

generateFluorophores

public List<*DefaultFluorophore*> **generateFluorophores**()
Executes the command that generates the fluorophores.

Returns The list of Fluorophores.

6.12.13 GenerateFluorophoresRandom3D.Builder

public static class **Builder** implements *FluorophoreCommandBuilder*
A builder for creating this command for fluorophore generation.

Methods

build

public *FluorophoreCommand* **build**()

camera

public *Builder* **camera**(*Camera camera*)

fluorDynamics

public *Builder* **fluorDynamics**(*FluorophoreDynamics fluorDynamics*)

numFluors

public *Builder* **numFluors**(int *numFluors*)

psfBuilder

public *Builder* **psfBuilder**(*PSFBuilder psfBuilder*)

zHigh

public *Builder* **zHigh**(double *zHigh*)

zLow

public *Builder* **zLow**(double *zLow*)

6.13 ch.epfl.leb.sass.models.fluorophores.internal.dynamics

6.13.1 FluorophoreDynamics

public abstract class **FluorophoreDynamics**
A fluorophore state system.

Fields

stateSystem

protected final *StateSystem* **stateSystem**
The state system describing the fluorescence dynamics.

Constructors

FluorophoreDynamics

protected **FluorophoreDynamics**(double *signal*, double *wavelength*, *StateSystem* *stateSystem*, int *startingState*, double[][][] *Mk*)
Initializes the state system with the transition rates and starting state.

Parameters

- **stateSystem** –
- **startingState** –
- **Mk** –

Methods

getMk

public double[][][] **getMk**()

getSignal

public double **getSignal**()

getStartingState

public int **getStartingState**()

getStateSystem

public *StateSystem* **getStateSystem**()

getWavelength

public double **getWavelength**()

6.13.2 FluorophoreDynamicsBuilder

public interface **FluorophoreDynamicsBuilder**
Interface for creating fluorophore dynamics.

Methods

build

public *FluorophoreDynamics* **build**()

6.13.3 PalmDynamics

public class **PalmDynamics** extends *FluorophoreDynamics*
A dynamical system for modeling PALM-like fluorescence dynamics.

Author Marcel Stefko, Kyle M. Douglass

Fields

STARTINGSTATE

public static final int **STARTINGSTATE**
Fluorophores start in the dark state.

6.13.4 PalmDynamics.Builder

public static class **Builder** implements *FluorophoreDynamicsBuilder*
Builder for creating PALM dynamical systems.

Methods

build

public *PalmDynamics* **build**()
Initialize a PALM-like dynamical system for fluorescence dynamics.

Returns The PALM dynamical system.

kA

public *Builder* **kA**(double *kA*)
The activation rate

Parameters

- **kA** –

kB

public *Builder* **kB**(double *kB*)
The bleaching rate

kD1

public *Builder* **kD1**(double *kD1*)
The rate of entering the first dark state

kD2

public *Builder* **kD2** (double *kD2*)
The rate of entering the second dark state

kR1

public *Builder* **kR1** (double *kR1*)
The return rate from the first dark state

kR2

public *Builder* **kR2** (double *kR2*)
The return rate from the second dark state

signal

public *Builder* **signal** (double *signal*)
The average number of photons per fluorophore per frame

Parameters

- **signal** –

Returns PalmDynamics builder

wavelength

public *Builder* **wavelength** (double *wavelength*)
The center wavelength of the fluorescence emission

Parameters

- **wavelength** –

Returns PalmDynamics builder

6.13.5 SimpleDynamics

public class **SimpleDynamics** extends *FluorophoreDynamics*
Dynamics for a simple three-state system (emitting, non-emitting, and bleached).

Author Marcel Stefko, Kyle M. Douglass

Fields

STARTINGSTATE

public static final int **STARTINGSTATE**
Fluorophores start in the dark state.

6.13.6 SimpleDynamics.Builder

public static class **Builder** implements *FluorophoreDynamicsBuilder*
Builder for creating Simple dynamical systems.

Methods

build

public *SimpleDynamics* **build**()
Creates a Simple dynamical system.

signal

public *Builder* **signal**(double *signal*)
The average number of photons per fluorophore per frame

Parameters

- **signal** –

Returns SimpleDynamics builder

tBl

public *Builder* **tBl**(double *tBl*)
The average bleaching time

Parameters

- **tBl** –

Returns SimpleDynamics builder

tOff

public *Builder* **tOff**(double *tOff*)
The average off time

Parameters

- **tOff** –

Returns SimpleDynamics builder

tOn

public *Builder* **tOn**(double *tOn*)
The average on time

Parameters

- **tOn** –

Returns SimpleDynamics builder

wavelength

public *Builder* **wavelength**(double *wavelength*)
The center wavelength of the fluorescence emission

Parameters

- **wavelength** –

Returns SimpleDynamics builder

6.13.7 StormDynamics

public class **StormDynamics** extends *FluorophoreDynamics*
A dynamical system for modeling STORM-like fluorescence dynamics.

Author Marcel Stefko, Kyle M. Douglass

Fields

STARTINGSTATE

public static final int **STARTINGSTATE**
Fluorophores start in the dark state.

6.13.8 StormDynamics.Builder

public static class **Builder** implements *FluorophoreDynamicsBuilder*

Methods

build

public *StormDynamics* **build**()

kBl

public *Builder* **kBl**(double *kBl*)
The bleaching rate

Returns StormDynamics Builder

kDark

public *Builder* **kDark**(double *kDark*)
The transition to the dark state

Parameters

- **kDark** –

Returns StormDynamics builder

kDarkRecovery

public *Builder* **kDarkRecovery**(double *kDarkRecovery*)
The recovery from the dark state

Parameters

- **kDarkRecovery** –

Returns StormDynamics builder

kDarkRecoveryConstant

public *Builder* **kDarkRecoveryConstant**(double *kDarkRecoveryConstant*)
The constant recovery rate from the dark state

Parameters

- **kDarkRecoveryConstant** –

Returns StormDynamics builder

kTriplet

public *Builder* **kTriplet**(double *kTriplet*)
The transition to the triplet state

Parameters

- **kTriplet** –

Returns StormDynamics builder

kTripletRecovery

public *Builder* **kTripletRecovery**(double *kTripletRecovery*)
The recovery rate from the triplet state

Parameters

- **kTripletRecovery** –

Returns StormDynamics builder

signal

public *Builder* **signal**(double *signal*)
The average number of photons per fluorophore per frame

Parameters

- **signal** –

Returns StormDynamics builder

wavelength

public *Builder* **wavelength** (double *wavelength*)
The center wavelength of the fluorescence emission

Parameters

- **wavelength** –

Returns StormDynamics builder

6.14 ch.epfl.leb.sass.models.legacy

6.14.1 Camera

public class **Camera**
Represents the parameters of the camera.

Author Marcel Stefko, Kyle M. Douglass

Fields

ADU_per_electron

public final double **ADU_per_electron**
Conversion factor between camera's analog-to-digital units (ADU) and electrons. [-]

EM_gain

public final int **EM_gain**
Electron multiplication (EM) gain of the camera. This may be set to zero for sensors without EM gain, such as CMOS sensors.

NA

public final double **NA**
numerical aperture [-]

acq_speed

public final int **acq_speed**
frame rate [frames/second]

baseline

public final int **baseline**
Camera pixel baseline (zero signal mean) [ADU]

dark_current

public final double **dark_current**
dark current [electrons/second/pixel]

fwhm_digital

public final double **fwhm_digital**
digital representation of the FWHM

magnification

public final double **magnification**
magnification of camera [-]

pixel_size

public final double **pixel_size**
physical size of pixel [m]

quantum_efficiency

public final double **quantum_efficiency**
quantum efficiency [0.0-1.0]

readout_noise

public final double **readout_noise**
readout noise of camera [RMS]

res_x

public final int **res_x**
horizontal image size [pixels]

res_y

public final int **res_y**
vertical image size [pixels]

stagePosition

public double **stagePosition**
Displacement of the coverslip surface from the objective's focal plane. This value assumes that the microscope

objective is inverted (facing up) and that the axial direction is positive going upwards. A negative value therefore implies that the coverslip has been moved down towards the objective bringing objects located above the coverslip into focus.

thermal_noise

public final double **thermal_noise**
noise in frame caused by dark current [electrons/frame/pixel]

wavelength

public final double **wavelength**
light wavelength [m]

Constructors

Camera

public **Camera**(int *res_x*, int *res_y*, int *acq_speed*, double *readout_noise*, double *dark_current*, double *quantum_efficiency*, double *ADU_per_electron*, int *EM_gain*, int *baseline*, double *pixel_size*, double *NA*, double *wavelength*, double *magnification*)
Initialize camera with parameters.

Parameters

- **res_x** – horizontal resolution [pixels]
- **res_y** – vertical resolution [pixels]
- **acq_speed** – frame rate [frames/second]
- **readout_noise** – readout noise of camera [RMS]
- **dark_current** – dark current [electrons/second/pixel]
- **quantum_efficiency** – quantum efficiency [0.0-1.0]
- **ADU_per_electron** – conversion between camera units and electrons [-]
- **EM_gain** – electron multiplication gain [-]
- **baseline** – zero-signal average of pixel values [ADU]
- **pixel_size** – physical size of pixel [m]
- **NA** – numerical aperture [-]
- **wavelength** – light wavelength [m]
- **magnification** – magnification of camera [-]

Methods

getRes_X

public int **getRes_X**()

getRes_Y

public int **getRes_Y**()

6.14.2 Device

public class **Device**
Encapsulator class which contains all device objects (camera, laser...)

Author Marcel Stefko, Kyle M. Douglass

Constructors

Device

public **Device**()
Initialize device with default parameters.

Device

public **Device**(*Camera cam*, *FluorophoreProperties fluo*, *Laser laser*, ArrayList<*DefaultFluorophore*> *emitters*, ArrayList<*Obstructor*> *obstructors*)
Initializes the device with given parameters.

Parameters

- **cam** – camera properties
- **fluo** – fluorophore properties
- **laser** – laser settings
- **emitters** – list of fluorophores
- **obstructors** – list of obstructors

Methods

getFOVsize_um

public double **getFOVsize_um**()

Returns size of current FOV in square micrometers

getLaserPower

public double **getLaserPower**()
Return current power of the laser.

Returns laser power

getOnEmitterCount

public double **getOnEmitterCount**()
Returns the number of currently active emitters.

Returns number of shining emitters

getPixelSizeUm

public double **getPixelSizeUm**()

Returns length of one pixel side in micrometers

getResolution

public int[] **getResolution**()
Return the camera resolution

Returns [res_x, res_y] int array

setLaserPower

public void **setLaserPower**(double *laser_power*)
Modifies the laser power to desired value.

Parameters

- **laser_power** – new laser power [W]

simulateFrame

public *ImageS* **simulateFrame**()
Generates a new frame based on the current device state, and moves device state forward. First the obstructions are drawn on the frame, then the fluorophores, and afterwards noise is added.

Returns simulated frame

6.14.3 Fluorophore3D

public class **Fluorophore3D** extends *DefaultFluorophore*
3D version of the fluorophore. It adds a third coordinate, and the fluorophore's PSF changes depending on its z-position.

Author Marcel Stefko

Fields

z

protected final double **z**

Constructors

Fluorophore3D

public **Fluorophore3D**(*Camera camera*, double *signal*, *StateSystem state_system*, int *start_state*, double *x*, double *y*, double *z*)

Methods

generate_signature_for_pixel

protected double **generate_signature_for_pixel**(int *x*, int *y*, double *camera_fwhm_digital*)

This function should implement the new PSF shape for the 3D fluorophore

Parameters

- **x** – x-position of camera pixel
- **y** – y-position of camera pixel
- **camera_fwhm_digital** – fwhm of gaussian PSF

Throws

- **MathException** –

Returns normalized value of pixel brightness (ie how bright is this particular pixel due to emission from the relevant fluorophore)

6.14.4 FluorophoreGenerator

public class **FluorophoreGenerator**

Populates a field of view with fluorophores. The FluorophoreGenerator contains a number of methods for creating actual fluorophore instances and in different arrangements, such as placing them on a grid, randomly distributing them in the FOV, and placing them according to input from a text file.

Author Marcel Stefko, Kyle M. Douglass

Methods

generate3DFluorophoresGrid

public static ArrayList<*Fluorophore3D*> **generate3DFluorophoresGrid**(int *spacing*, *Camera cam*, *FluorophoreProperties fluo*)

Parameters

- **spacing** –
- **cam** –
- **fluo** –

generateFluorophoresFromCSV

public static ArrayList<*DefaultFluorophore*> **generateFluorophoresFromCSV**(File *file*, *Camera camera*, *PSFBuilder psfBuilder*, *FluorophoreProperties fluorProp*, boolean *rescale*)

Parse a CSV file and generate fluorophores from it.

Parameters

- **file** – The CSV file. If this is null, then a dialog is opened.
- **camera** – The camera for determining the size of the field of view.
- **psfBuilder** – Builder for calculating microscope PSFs.
- **fluorProp** – The fluorophore dynamics properties.
- **rescale** – if true, positions are rescaled to fit into frame, otherwise positions outside of frame are cropped

Throws

- **IOException** –
- **FileNotFoundException** –

Returns list of fluorophores.

generateFluorophoresGrid

public static ArrayList<*DefaultFluorophore*> **generateFluorophoresGrid**(int *spacing*, *Camera cam*, *FluorophoreProperties fluo*)

Generate a rectangular grid of fluorophores

Parameters

- **spacing** – distance between nearest neighbors
- **cam** – Camera
- **fluo** – type of fluorophore

Returns The list of fluorophores.

generateFluorophoresGrid2D

public static ArrayList<*DefaultFluorophore*> **generateFluorophoresGrid2D**(int *spacing*, *Camera camera*, *PSFBuilder psfBuilder*, *FluorophoreProperties fluorProp*)

Generate a rectangular grid of fluorophores.

Parameters

- **spacing** – The distance along the grid between nearest neighbors.

- **camera** – The camera for determining the size of the field of view.
- **psfBuilder** – Builder for calculating microscope PSFs.
- **fluorProp** – The fluorophore dynamics properties.

Returns The list of fluorophores.

generateFluorophoresGrid3D

public static ArrayList<*DefaultFluorophore*> **generateFluorophoresGrid3D**(int *spacing*, double *zLow*, double *zHigh*, *Camera* *camera*, *PSFBuilder* *psfBuilder*, *FluorophoreProperties* *fluorProp*)

Create fluorophores on a 2D grid and step-wise in the axial direction.

Parameters

- **spacing** – The distance along the grid between nearest neighbors.
- **zLow** – The lower bound on the range in z in units of pixels.
- **zHigh** – The upper bound on the range in z in units of pixels.
- **camera** – The camera for determining the size of the field of view.
- **psfBuilder** – Builder for calculating microscope PSFs.
- **fluorProp** – The fluorophore dynamics properties.

Returns The list of fluorophores.

generateFluorophoresRandom

public static ArrayList<*DefaultFluorophore*> **generateFluorophoresRandom**(int *n_fluos*, *Camera* *cam*, *FluorophoreProperties* *fluo*)

Randomly populate the field of view with fluorophores.

Parameters

- **n_fluos** – number of emitters to be generated
- **cam** – camera properties
- **fluo** – fluorophore properties

generateFluorophoresRandom2D

public static ArrayList<*DefaultFluorophore*> **generateFluorophoresRandom2D**(int *numFluors*, *Camera* *camera*, *PSFBuilder* *psfBuilder*, *FluorophoreProperties* *fluorProp*)

Randomly populate the field of view with fluorophores.

Parameters

- **numFluors** – The number of fluorophores to add to the field of view.
- **camera** – The camera for determining the size of the field of view.
- **psfBuilder** – Builder for calculating microscope PSFs.
- **fluorProp** – The fluorophore dynamics properties.

Returns The list of fluorophores.

generateFluorophoresRandom3D

public static ArrayList<*DefaultFluorophore*> **generateFluorophoresRandom3D**(int *numFluors*, double *zLow*, double *zHigh*, *Camera* *camera*, *PSFBuilder* *psfBuilder*, *FluorophoreProperties* *fluorProp*)

Randomly populate the field of view with fluorophores in three dimensions.

Parameters

- **numFluors** – The number of fluorophores to add to the field of view.
- **zLow** – The lower bound on the range in z in units of pixels
- **zHigh** – The upper bound on the range in z in units of pixels
- **camera** – The camera for determining the size of the field of view.
- **psfBuilder** – Builder for calculating microscope PSFs.
- **fluorProp** – The fluorophore dynamics properties.

Returns The list of fluorophores.

parseFluorophoresFromCsv

public static ArrayList<*DefaultFluorophore*> **parseFluorophoresFromCsv**(File *file*, *Camera* *camera*, *FluorophoreProperties* *fluo*, boolean *rescale*)

Parses fluorophore positions from csv file. All lines which don't start with "#" have to contain at least 2 doubles, which are interpreted as x and y positions in pixels.

Parameters

- **file** – csv file, if null, a dialog is opened
- **camera** – camera settings
- **fluo** – fluorophore settings
- **rescale** – if true, positions are rescaled to fit into frame, otherwise positions outside of frame are cropped

Throws

- **IOException** –
- **FileNotFoundException** –

Returns list of fluorophores

parseMovingFluorophoresFromCsv

public static ArrayList<*MovingFluorophore*> **parseMovingFluorophoresFromCsv**(File *file*, *Camera camera*, *SimpleProperties fluo*)

Parses moving fluorophores and their trajectories from a csv file. CSV file column format: emitter_no[-],frame_no[-],x[px],y[px] Frame and emitter numbers must be strictly increasing, but don't have to be consecutive (gaps in frame numbers are interpolated).

Parameters

- **file** – csv file, if null, a dialog is opened
- **camera** – camera settings
- **fluo** – moving fluorophore settings

Throws

- **IOException** –
- **FileNotFoundException** –

Returns list of fluorophores

6.14.5 FluorophoreGeneratorTest

public class **FluorophoreGeneratorTest**

Author Kyle M. Douglass

Constructors

FluorophoreGeneratorTest

public **FluorophoreGeneratorTest**()

Methods

testGenerateFluorophoresGrid3D

public void **testGenerateFluorophoresGrid3D**()

Test of generateFluorophoresGrid3D method, of class FluorophoreGenerator.

6.14.6 FluorophoreProperties

public abstract class **FluorophoreProperties**

Abstract class that creating fluorophores with specific blinking dynamics.

Author Marcel Stefko, Kyle M. Douglass

Fields

background

public final double **background**

signal

public final double **signal**

wavelength

public final double **wavelength**

Constructors

FluorophoreProperties

public **FluorophoreProperties** (double *signal*, double *background*)

Parameters

- **signal** –
- **background** –

FluorophoreProperties

public **FluorophoreProperties** (double *signal*, double *background*, double *wavelength*)

Methods

createFluorophore

public abstract *DefaultFluorophore* **createFluorophore** (*Camera* *camera*, double *x*, double *y*)

Create fluorophore with these given properties

Parameters

- **camera** – Camera to calculate pattern
- **x** – x-position in pixels
- **y** – y-position in pixels

Returns generated fluorophore

createFluorophore3D

public abstract *Fluorophore3D* **createFluorophore3D** (*Camera* *camera*, double *x*, double *y*, double *z*)

getWavelength

public double **getWavelength**()

newFluorophore

public abstract *DefaultFluorophore* **newFluorophore**(*PSFBuilder psfBuilder*, double *x*, double *y*, double *z*)

Create fluorophore with the given properties.

Parameters

- **psfBuilder** – A PSFBuilder for constructing the microscope PSF.
- **x** – DefaultFluorophore x-position in pixels.
- **y** – DefaultFluorophore y-position in pixels.
- **z** – DefaultFluorophore z-position in pixels.

Returns A new fluorophore object.

6.14.7 GoldBeads

public class **GoldBeads** implements *Obstructor*

A number of constantly-shining gold beads interspersed in the frame.

Author Marcel Stefko

Constructors

GoldBeads

public **GoldBeads**(int *beadCount*, *Camera camera*, double *brightness*)

Randomly places gold beads into the camera field of view.

Parameters

- **beadCount** – number of gold beads
- **camera** – camera properties
- **brightness** – how bright the beads are [photons/frame]

Methods

applyTo

public void **applyTo**(float[][] *pixels*)

6.14.8 Laser

public class **Laser**
Class representing the laser shining on the sample.

Author Marcel Stefko

Constructors

Laser

public **Laser**(double *start_power*, double *max_power*, double *min_power*)
Initialize laser with given parameters.

Parameters

- **start_power** – initial power of laser [W]
- **max_power** – maximal laser power [W]
- **min_power** – minimal laser power [W]

Methods

getPower

public double **getPower**()
Returns current laser power.

Returns current laser power [W]

setPower

public void **setPower**(double *new_power*)
Sets new laser power if it is within limits, or the closest allowed value if it is outside limits.

Parameters

- **new_power** – desired laser power [W]

6.14.9 MovingFluorophore

public class **MovingFluorophore** extends *DefaultFluorophore*
DefaultFluorophore that can move between frames

Author Marcel Stefko

Constructors

MovingFluorophore

public **MovingFluorophore**(*Camera* *camera*, double *signal*, *StateSystem* *state_system*, int *start_state*, double *x*, double *y*, ArrayList<Point2D.Double> *trajectory*)
DefaultFluorophore which moves along a certain trajectory and then stops

Parameters

- **camera** –
- **x** – initial x position
- **y** – initial y position
- **trajectory** – trajectory of fluorophore in absolute numbers

Methods

applyTo

public void **applyTo**(float[][] *pixels*)

6.14.10 PalmProperties

public class **PalmProperties** extends *FluorophoreProperties*

Author stefko

Constructors

PalmProperties

public **PalmProperties**(double *signal*, double *background*, double *k_a*, double *k_b*, double *k_d1*, double *k_d2*, double *k_r1*, double *k_r2*)

Parameters

- **signal** –
- **background** –
- **k_a** –
- **k_b** –
- **k_d1** –
- **k_d2** –
- **k_r1** –
- **k_r2** –

PalmProperties

public **PalmProperties**(double *signal*, double *wavelength*, double *background*, double *k_a*, double *k_b*, double *k_d1*, double *k_d2*, double *k_r1*, double *k_r2*)

Methods

createFluorophore

public *DefaultFluorophore* **createFluorophore** (*Camera camera*, double *x*, double *y*)

createFluorophore3D

public *Fluorophore3D* **createFluorophore3D** (*Camera camera*, double *x*, double *y*, double *z*)

newFluorophore

public *DefaultFluorophore* **newFluorophore** (*PSFBuilder psfBuilder*, double *x*, double *y*, double *z*)

6.14.11 STORMsim

public class **STORMsim** extends *AbstractSimulator*

Implementation of the ImageGenerator interface with methods required by AbstractSimulator.

Author Marcel Stefko

Constructors

STORMsim

public **STORMsim** (*Device device*)

Initialize the generator, either from GUI dialog or use default params.

Parameters

- **device** –

Methods

getControlSignal

public double **getControlSignal** ()

getCustomParameters

public HashMap<String, Double> **getCustomParameters** ()

getFOVSize

public double **getFOVSize** ()

getNextImage

public *ImageS* **getNextImage**()

getObjectSpacePixelSize

public double **getObjectSpacePixelSize**()

Returns length of one pixel side in micrometers

getShortTrueSignalDescription

public String **getShortTrueSignalDescription**()

getTrueSignal

public double **getTrueSignal**(int *image_no*)

incrementTimeStep

public void **incrementTimeStep**()
Advance the simulation by one time step (i.e. one frame), but do not create an image.

setControlSignal

public void **setControlSignal**(double *value*)

setCustomParameters

public void **setCustomParameters**(HashMap<String, Double> *map*)

6.14.12 SimpleProperties

public class **SimpleProperties** extends *FluorophoreProperties*
SimpleProperties properties (signal, background values and time constants).

Author Marcel Stefko

Constructors

SimpleProperties

public **SimpleProperties**(double *signal_per_frame*, double *background_per_frame*, double *Ton*, double *Toff*, double *Tbl*)
Initialize fluorophore with given properties

Parameters

- **signal_per_frame** – photons emitted if fluorophore is fully on
- **background_per_frame** – constant background of the fluorophore
- **Ton** – mean on-time with unit laser power [frames]
- **Toff** – mean off-time with unit laser power [frames]
- **Tbl** – mean bleaching time with unit laser power [frames]

SimpleProperties

public **SimpleProperties**(double *signal_per_frame*, double *wavelength*, double *background_per_frame*, double *Ton*, double *Toff*, double *Tbl*)

Initialize fluorophore with given properties

Parameters

- **signal_per_frame** – photons emitted if fluorophore is fully on
- **wavelength** – The emission wavelength of the fluorophore
- **background_per_frame** – constant background of the fluorophore
- **Ton** – mean on-time with unit laser power [frames]
- **Toff** – mean off-time with unit laser power [frames]
- **Tbl** – mean bleaching time with unit laser power [frames]

Methods

createFluorophore

public *DefaultFluorophore* **createFluorophore**(*Camera camera*, double *x*, double *y*)

createFluorophore3D

public *Fluorophore3D* **createFluorophore3D**(*Camera camera*, double *x*, double *y*, double *z*)

createMovingFluorophore

public *MovingFluorophore* **createMovingFluorophore**(*Camera camera*, double *x*, double *y*, ArrayList<Point2D.Double> *trajectory*)

Creates a moving variant of simple fluorophore

Parameters

- **camera** –
- **x** –
- **y** –
- **trajectory** –

newFluorophore

public *DefaultFluorophore* **newFluorophore** (*PSFBuilder psfBuilder*, double *x*, double *y*, double *z*)

6.14.13 dStormProperties

public class **dStormProperties** extends *FluorophoreProperties*

Author stefko

Constructors

dStormProperties

public **dStormProperties** (double *signal*, double *background*, double *k_bl*, double *k_triplet*, double *k_triplet_recovery*, double *k_dark*, double *k_dark_recovery*, double *k_dark_recovery_constant*)

Parameters

- **signal** –
- **background** –
- **k_bl** –
- **k_triplet** –
- **k_triplet_recovery** –
- **k_dark** –
- **k_dark_recovery** –
- **k_dark_recovery_constant** –

dStormProperties

public **dStormProperties** (double *signal*, double *wavelength*, double *background*, double *k_bl*, double *k_triplet*, double *k_triplet_recovery*, double *k_dark*, double *k_dark_recovery*, double *k_dark_recovery_constant*)

Methods

createFluorophore

public *DefaultFluorophore* **createFluorophore** (*Camera camera*, double *x*, double *y*)

createFluorophore3D

public *Fluorophore3D* **createFluorophore3D** (*Camera camera*, double *x*, double *y*, double *z*)

newFluorophore

public *DefaultFluorophore* **newFluorophore** (*PSFBuilder psfBuilder*, double *x*, double *y*, double *z*)

6.15 ch.epfl.leb.sass.models.obstructors

6.15.1 Obstructor

public interface **Obstructor**

This object is a constant obstruction of the field of view (for example gold bead, foreign object in field of view, dirt, etc.)

Author Marcel Stefko

Methods

applyTo

public void **applyTo** (float[][] *pixels*)

Draws the obstruction onto the given float array representing an image.

Parameters

- **pixels** – image to be drawn on

6.16 ch.epfl.leb.sass.models.obstructors.internal

6.16.1 ConstantBackground

public class **ConstantBackground** implements *Obstructor*

Constant overlay loaded from an image

Author Marcel Stefko

Constructors

ConstantBackground

public **ConstantBackground** (*Camera camera*)

Load the background image by a file selection dialog

Parameters

- **camera** – camera settings

ConstantBackground

public **ConstantBackground** (*Camera camera*, File *file*)

Load background image from specified file

Parameters

- **camera** – camera settings
- **file** – tiff file to be loaded

Methods

applyTo

public void **applyTo**(float[][] *pixels*)

6.16.2 Fiducial

public class **Fiducial** extends *AbstractEmitter* implements *Obstructor*

Constructors

Fiducial

public **Fiducial**(*Camera camera*, double *brightness*, double *x*, double *y*)

Parameters

- **camera** –
- **brightness** –
- **x** –
- **y** –

Fiducial

public **Fiducial**(*PSFBuilder psfBuilder*, double *brightness*, double *x*, double *y*, double *z*)

Methods

simulateBrightness

protected double **simulateBrightness**()

6.16.3 GoldBead

public class **GoldBead** extends *AbstractEmitter*

Constructors

GoldBead

public **GoldBead** (*Camera camera*, double *brightness*, double *x*, double *y*)

Parameters

- **camera** –
- **brightness** –
- **x** –
- **y** –

GoldBead

public **GoldBead** (*PSFBuilder psfBuilder*, double *brightness*, double *x*, double *y*, double *z*)

Methods

simulateBrightness

protected double **simulateBrightness** ()

6.17 ch.epfl.leb.sass.models.obstructors.internal.commands

6.17.1 GenerateFiducialsRandom2D

public final class **GenerateFiducialsRandom2D** implements *ObstructorCommand*

Author Kyle M. Douglass

Methods

generateObstructors

public List<*Obstructor*> **generateObstructors** ()
Executes the command that generates the fluorophores.

Returns The list of fluorophores.

6.17.2 GenerateFiducialsRandom2D.Builder

public static class **Builder** implements *ObstructorCommandBuilder*
A builder for creating this command for obstructor generation.

Methods

brightness

public *Builder* **brightness**(double *brightness*)

build

public *ObstructorCommand* **build**()

camera

public *Builder* **camera**(*Camera camera*)

numFiducials

public *Builder* **numFiducials**(int *numFiducials*)

psfBuilder

public *Builder* **psfBuilder**(*PSFBuilder psfBuilder*)

stage

public *Builder* **stage**(*Stage stage*)

6.17.3 ObstructorCommand

public interface **ObstructorCommand**

Author Kyle M. Douglass

Methods

generateObstructors

public List<*Obstructor*> **generateObstructors**()

6.17.4 ObstructorCommandBuilder

public interface **ObstructorCommandBuilder**

Interface for populating the field with obstructors, i.e. gold beads.

Author Kyle M. Douglass

Methods

brightness

public *ObstructorCommandBuilder* **brightness**(double *brightness*)

build

public *ObstructorCommand* **build**()

camera

public *ObstructorCommandBuilder* **camera**(*Camera* *camera*)

psfBuilder

public *ObstructorCommandBuilder* **psfBuilder**(*PSFBuilder* *psfBuilder*)

stage

public *ObstructorCommandBuilder* **stage**(*Stage* *stage*)

6.17.5 ObstructorReceiver

public class **ObstructorReceiver**
Creates obstructors after receiving commands.

Author Kyle M. Douglass

Methods

generateGoldBeadsRandom2D

public static ArrayList<*Obstructor*> **generateGoldBeadsRandom2D**(int *numBeads*, double *brightness*, *Camera* *camera*, *Stage* *stage*, *PSFBuilder* *psfBuilder*)

6.18 ch.epfl.leb.sass.models.psfs

6.18.1 PSF

public interface **PSF**
Interface that defines the behavior of a microscope point spread function.

Author Kyle M. Douglass

Methods

generatePixelSignature

public double **generatePixelSignature**(int *pixelX*, int *pixelY*)
Computes the expected value for the PSF integrated over a pixel.

Parameters

- **pixelX** – The pixel's x-position.
- **pixelY** – The pixel's y-position.

Throws

- **org.apache.commons.math.MathException** –

Returns The relative probability of a photon hitting this pixel.

generateSignature

public void **generateSignature**(ArrayList<*Pixel*> *pixels*)
Computes the digitized PSF across all pixels within the emitter's vicinity.

Parameters

- **pixels** – The list of pixels spanned by the emitter's image.

getRadius

public double **getRadius**()
Returns the radius of the circle that fully encloses the PSF. This value is used to determine how many pixels within the vicinity of the emitter contribute to the PSF. It is necessary because many PSF models extend to infinity in one or more directions.

Returns The radius of the PSF in pixels.

6.18.2 PSFBuilder

public interface **PSFBuilder**
Defines the Builder interface for constructing PSFs. Passing Builders instances, rather than PSF instances, to the simulation allows the PSF to be constructed at different times during the simulation. For example, one might set basic parameters like the wavelength in the beginning of the simulation and set the emitter's z-position immediately before a frame is computed. This means the simulation can dynamically create new PSF instances in response to changing simulation parameters.

Author Kyle M. Douglass

Methods

FWHM

public *PSFBuilder* **FWHM**(double *FWHM*)
The Gaussian approximation's FWHM for this PSF.

NA

public *PSFBuilder* **NA**(double *NA*)
The numerical aperture of the objective.

build

public *PSF* **build**()
Builds a new instance of the PSF model.

Returns The PSF model.

eX

public *PSFBuilder* **eX**(double *eX*)
Sets the emitter's x-position.

Parameters

- **eX** – The emitter's x-position. [pixels]

eY

public *PSFBuilder* **eY**(double *eY*)
Sets the emitter's y-position.

Parameters

- **eY** – The emitter's y-position. [pixels]

eZ

public *PSFBuilder* **eZ**(double *eZ*)
Sets the emitter's z-position.

Parameters

- **eZ** – The emitter's z-position. [pixels]

resLateral

public *PSFBuilder* **resLateral**(double *resLateral*)
Object space pixel size

stageDisplacement

public *PSFBuilder* **stageDisplacement**(double *stageDisplacement*)
Sets the stage displacement for axially-dependent PSFs.

wavelength

public *PSFBuilder* **wavelength** (double *wavelength*)
Wavelength of the light.

6.19 ch.epfl.leb.sass.models.psfs.internal

6.19.1 Gaussian2D

public final class **Gaussian2D** implements *PSF*
Generates a digital representation of a two-dimensional Gaussian PSF.

Author Kyle M. Douglass

Methods

generatePixelSignature

public double **generatePixelSignature** (int *pixelX*, int *pixelY*)
Computes the relative probability of receiving a photon at the pixel. (emitterX, emitterY). The z-position of the emitter is ignored.

Parameters

- **pixelX** – The pixel's x-position.
- **pixelY** – The pixel's y-position.

Throws

- **org.apache.commons.math.MathException** –

Returns The probability of a photon hitting this pixel.

generateSignature

public void **generateSignature** (ArrayList<*Pixel*> *pixels*)
Generates the digital signature of the emitter on its nearby pixels.

Parameters

- **pixels** – The list of pixels spanned by the emitter's image.

getFWHM

public double **getFWHM** ()

getRadius

public double **getRadius** ()
Computes the half-width of the PSF for determining which pixels contribute to the emitter signal. For a 2D Gaussian, the effective width used here is three times the standard deviation.

Returns The width of the PSF.

setFWHM

public void **setFWHM**(double *fwhm*)

6.19.2 Gaussian2D.Builder

public static class **Builder** implements *PSFBuilder*
The builder for constructing Gaussian2D instances.

Methods

FWHM

public *Builder* **FWHM**(double *fwhm*)

NA

public *Builder* **NA**(double *NA*)

build

public *Gaussian2D* **build**()

eX

public *Builder* **eX**(double *eX*)

eY

public *Builder* **eY**(double *eY*)

eZ

public *Builder* **eZ**(double *eZ*)

resLateral

public *Builder* **resLateral**(double *resLateral*)

stageDisplacement

public *Builder* **stageDisplacement**(double *stageDisplacement*)

wavelength

public *Builder* **wavelength** (double *wavelength*)

6.19.3 Gaussian2DTest

public class **Gaussian2DTest**

Author Kyle M. Douglass

Methods

setUp

public void **setUp** ()

testGeneratePixelSignature

public void **testGeneratePixelSignature** ()
Test of generatePixelSignature method, of class Gaussian2D.

testGetRadius

public void **testGetRadius** ()
Test of getRadius method, of class Gaussian2D.

testGetSignature

public void **testGetSignature** ()
Test of getSignature method, of class Gaussian2D.

6.19.4 Gaussian3D

public final class **Gaussian3D** implements *PSF*
Generates a digital representation of a three-dimensional Gaussian PSF. In this simple but unphysical model, the variance of the Gaussian PSF from an emitter at a distance z from the focal plane scales linearly with the amount of defocus.

Author Kyle M. Douglass

Methods

generatePixelSignature

public double **generatePixelSignature** (int *pixelX*, int *pixelY*)
Computes the relative probability of receiving a photon at the pixel.

Parameters

- **pixelX** – The pixel's x-position.
- **pixelY** – The pixel's y-position.

Throws

- **org.apache.commons.math.MathException** –

Returns The probability of a photon hitting this pixel.

generateSignature

public void **generateSignature**(ArrayList<*Pixel*> *pixels*)

Generates the digital signature of the emitter on its nearby pixels.

Parameters

- **pixels** – The list of pixels spanned by the emitter's image.

getFWHM

public double **getFWHM**()

getNumericalAperture

public double **getNumericalAperture**()

getRadius

public double **getRadius**()

Computes the half-width of the PSF for determining which pixels contribute to the emitter signal. The effective width used here is five times the standard deviation when the emitter is exactly in focus. The larger factor of five accounts for the larger lateral PSF size when it is out of focus.

Returns The width of the PSF.

setFWHM

public void **setFWHM**(double *fwhm*)

setNumericalAperture

public void **setNumericalAperture**(double *numericalAperture*)

6.19.5 Gaussian3D.Builder

public static class **Builder** implements *PSFBuilder*

The builder for constructing Gaussian2D instances.

Methods

FWHM

public *Builder* **FWHM**(double *fwhm*)

NA

public *Builder* **NA**(double *NA*)

build

public *Gaussian3D* **build**()

eX

public *Builder* **eX**(double *eX*)

eY

public *Builder* **eY**(double *eY*)

eZ

public *Builder* **eZ**(double *eZ*)

resLateral

public *Builder* **resLateral**(double *resLateral*)

stageDisplacement

public *Builder* **stageDisplacement**(double *stageDisplacement*)

wavelength

public *Builder* **wavelength**(double *wavelength*)

6.19.6 Gaussian3DTest

public class **Gaussian3DTest**

Author douglass

Methods

setUp

public void **setUp**()

testGeneratePixelSignatureInFocus

public void **testGeneratePixelSignatureInFocus**()
Test of generatePixelSignature method, of class Gaussian3D.

testGeneratePixelSignatureOutOfFocus

public void **testGeneratePixelSignatureOutOfFocus**()
Test of generatePixelSignature method, of class Gaussian3D.

testGetRadius

public void **testGetRadius**()
Test of getRadius method, of class Gaussian2D.

testGetSignatureInFocus

public void **testGetSignatureInFocus**()
Test of getSignature method, of class Gaussian2D.

6.19.7 GibsonLanniPSF

public final class **GibsonLanniPSF** implements *PSF*
Computes an emitter PSF based on the Gibson-Lanni model. This algorithm was first described in Li, J., Xue, F., and Blu, T. (2017). Fast and accurate three-dimensional point spread function computation for fluorescence microscopy. JOSA A, 34(6), 1029-1034. The code is adapted from MicroscPSF-ImageJ by Jizhou Li: https://github.com/hijizhou/MicroscPSF-ImageJ

Author Kyle M. Douglass

Methods

generatePixelSignature

public double **generatePixelSignature**(int *pixelX*, int *pixelY*)
Computes the relative probability of receiving a photon at pixel (pixelX, pixelY) from an emitter at (emitterX, emitterY, emitterZ).

Parameters

- **pixelX** – The pixel's x-position.
- **pixelY** – The pixel's y-position.

Returns The probability of a photon hitting this pixel.

generateSignature

public void **generateSignature**(ArrayList<*Pixel*> *pixels*)

Generates the digital signature (the PSF) of the emitter on its nearby pixels.

Parameters

- **pixels** – The list of pixels spanned by the emitter's image.

getRadius

public double **getRadius**()

Computes the half-width of the PSF for determining which pixels contribute to the emitter signal. This number is based on the greatest horizontal or vertical extent of the grid that the PSF is computed on. If maxRadius is smaller than that determined by the PSF's computational grid, then maxRadius is returned.

Returns The width of the PSF.

6.19.8 GibsonLanniPSF.Builder

public static class **Builder** implements *PSFBuilder*

Constructors

Builder

public **Builder**()

Methods

FWHM

public *Builder* **FWHM**(double *FWHM*)

NA

public *Builder* **NA**(double *NA*)

build

public *GibsonLanniPSF* **build**()

eX

public *Builder* **eX**(double *eX*)

eY

public *Builder* **eY** (double *eY*)

eZ

public *Builder* **eZ** (double *eZ*)

maxRadius

public *Builder* **maxRadius** (double *maxRadius*)

ng

public *Builder* **ng** (double *ng*)

ng0

public *Builder* **ng0** (double *ng0*)

ni

public *Builder* **ni** (double *ni*)

ni0

public *Builder* **ni0** (double *ni0*)

ns

public *Builder* **ns** (double *ns*)

numBasis

public *Builder* **numBasis** (int *numBasis*)

numSamples

public *Builder* **numSamples** (int *numSamples*)

oversampling

public *Builder* **oversampling** (int *oversampling*)

resLateral

public *Builder* **resLateral** (double *resLateral*)

resPSF

public *Builder* **resPSF** (double *resPSF*)

resPSFAxial

public *Builder* **resPSFAxial** (double *resPSFAxial*)

sizeX

public *Builder* **sizeX** (int *sizeX*)

sizeY

public *Builder* **sizeY** (int *sizeY*)

solver

public *Builder* **solver** (String *solver*)

stageDisplacement

public *Builder* **stageDisplacement** (double *stageDisplacement*)

tg

public *Builder* **tg** (double *tg*)

tg0

public *Builder* **tg0** (double *tg0*)

ti0

public *Builder* **ti0** (double *ti0*)

wavelength

public *Builder* **wavelength** (double *wavelength*)

6.19.9 GibsonLanniPSFTest

public class **GibsonLanniPSFTest**
Tests for the GibsonLanniPSF class.

Author Kyle M. Douglass

Constructors

GibsonLanniPSFTest

public **GibsonLanniPSFTest**()

Methods

setUp

public void **setUp**()

testGeneratePixelSignature

public void **testGeneratePixelSignature**()
Test of generatePixelSignature method, of class GibsonLanniPSF.

testGenerateSignature

public void **testGenerateSignature**()
Test of generateSignature method, of class GibsonLanniPSF.

testGetRadius

public void **testGetRadius**()
Test of getRadius method, of class GibsonLanniPSF.

testGetRadiusSmallMaxRadius

public void **testGetRadiusSmallMaxRadius**()
Test of getRadius method, of class GibsonLanniPSF, with maxRadius small.

6.19.10 ProfileGibsonLanniPSF

public class **ProfileGibsonLanniPSF**
Demonstrates how to create a Gibson-Lanni PSF.

Author Kyle M. Douglass

Methods

main

public static void **main** (String[] *args*)

6.20 ch.epfl.leb.sass.server

6.20.1 ImageGenerationException

public class **ImageGenerationException** extends org.apache.thrift.TException implements org.apache.thrift.TBase<*ImageGene*

Fields

metaDataMap

public static final java.util.Map<*_Fields*, org.apache.thrift.meta_data.FieldMetaData> **metaDataMap**

Constructors

ImageGenerationException

public **ImageGenerationException** ()

ImageGenerationException

public **ImageGenerationException** (*ImageGenerationException other*)
Performs a deep copy on *other*.

Methods

clear

public void **clear** ()

compareTo

public int **compareTo** (*ImageGenerationException other*)

deepCopy

public *ImageGenerationException* **deepCopy** ()

equals

public boolean **equals** (java.lang.Object *that*)

equals

public boolean **equals** (*ImageGenerationException* *that*)

fieldForId

public *_Fields* **fieldForId** (int *fieldId*)

getFieldValue

public java.lang.Object **getFieldValue** (*_Fields* *field*)

hashCode

public int **hashCode** ()

isSet

public boolean **isSet** (*_Fields* *field*)
Returns true if field corresponding to fieldID is set (has been assigned a value) and false otherwise

read

public void **read** (org.apache.thrift.protocol.TProtocol *iprot*)

setFieldValue

public void **setFieldValue** (*_Fields* *field*, java.lang.Object *value*)

toString

public java.lang.String **toString** ()

validate

public void **validate** ()

write

public void **write** (org.apache.thrift.protocol.TProtocol *oprot*)

6.20.2 ImageGenerationException._Fields

public enum **`_Fields`** implements org.apache.thrift.TFieldIdEnum
The set of fields this struct contains, along with convenience methods for finding and manipulating them.

Enum Constants

6.20.3 RPCServer

public class **`RPCServer`**
An RPC server for remote control of the simulation over a network socket.

Author Kyle M. Douglass

Fields

handler

public static *RemoteSimulationServiceHandler* **`handler`**

processor

public static *RemoteSimulationService.Processor* **`processor`**

Constructors

RPCServer

public **`RPCServer`**(*Model model*, int *port*)
Creates a new RPCServer and initializes–but does not start–it.

Parameters

- **`model`** – A model of a microscope to simulate.
- **`port`** – The port number for server communications.

RPCServer

public **`RPCServer`**(*Microscope microscope*, int *port*)
Creates a new RPCServer and initializes–but does not start–it.

Parameters

- **`microscope`** – An instance of a microscope to simulate.
- **`port`** – The port number for server communications.

Methods

isServing

public boolean **isServing**()
Checks the status of the server.

Returns Is the server running? (true or false)

main

public static void **main**(String[] *args*)
Main function used for testing the RPC server.

Parameters

- **args** –

serve

public void **serve**()
Starts the server.

simple

public static void **simple**(*RemoteSimulationService.Processor processor*)

stop

public void **stop**()
Stops the server.

6.20.4 RemoteSimulationService

public class **RemoteSimulationService**

6.20.5 RemoteSimulationService.AsyncClient

public static class **AsyncClient** extends org.apache.thrift.async.TAsyncClient implements *AsyncIface*

Constructors

AsyncClient

public **AsyncClient**(org.apache.thrift.protocol.TProtocolFactory *protocolFactory*,
org.apache.thrift.async.TAsyncClientManager *clientManager*,
org.apache.thrift.transport.TNonblockingTransport *transport*)

Methods

getNextImage

public void **getNextImage**(org.apache.thrift.async.AsyncMethodCallback<java.nio.ByteBuffer> *resultHandler*)

getServerStatus

public void **getServerStatus**(org.apache.thrift.async.AsyncMethodCallback<java.lang.String> *resultHandler*)

getSimulationState

public void **getSimulationState**(org.apache.thrift.async.AsyncMethodCallback<java.lang.String> *resultHandler*)

setActivationLaserPower

public void **setActivationLaserPower**(double *power*, org.apache.thrift.async.AsyncMethodCallback<Void> *resultHandler*)

6.20.6 RemoteSimulationService.AsyncClient.Factory

public static class **Factory** implements org.apache.thrift.async.TAsyncClientFactory<*AsyncClient*>

Constructors

Factory

public **Factory**(org.apache.thrift.async.TAsyncClientManager *clientManager*, org.apache.thrift.protocol.TProtocolFactory *protocolFactory*)

Methods

getAsyncClient

public *AsyncClient* **getAsyncClient**(org.apache.thrift.transport.TNonblockingTransport *transport*)

6.20.7 RemoteSimulationService.AsyncClient.getNextImage_call

public static class **getNextImage_call** extends org.apache.thrift.async.TAsyncMethodCall<java.nio.ByteBuffer>

Constructors

getNextImage_call

public **getNextImage_call**(org.apache.thrift.async.AsyncMethodCallback<java.nio.ByteBuffer> *resultHandler*, org.apache.thrift.async.TAsyncClient *client*, org.apache.thrift.protocol.TProtocolFactory *protocolFactory*, org.apache.thrift.transport.TNonblockingTransport *transport*)

Methods

getResult

public java.nio.ByteBuffer **getResult**()

write_args

public void **write_args**(org.apache.thrift.protocol.TProtocol *prot*)

6.20.8 RemoteSimulationService.AsyncClient.getServerStatus_call

public static class **getServerStatus_call** extends org.apache.thrift.async.TAsyncMethodCall<java.lang.String>

Constructors

getServerStatus_call

public **getServerStatus_call**(org.apache.thrift.async.AsyncMethodCallback<java.lang.String> *resultHandler*, org.apache.thrift.async.TAsyncClient *client*, org.apache.thrift.protocol.TProtocolFactory *protocolFactory*, org.apache.thrift.transport.TNonblockingTransport *transport*)

Methods

getResult

public java.lang.String **getResult**()

write_args

public void **write_args**(org.apache.thrift.protocol.TProtocol *prot*)

6.20.9 RemoteSimulationService.AsyncClient.getSimulationState_call

public static class **getSimulationState_call** extends org.apache.thrift.async.TAsyncMethodCall<java.lang.String>

Constructors

getSimulationState_call

public **getSimulationState_call**(org.apache.thrift.async.AsyncMethodCallback<java.lang.String> *resultHandler*, org.apache.thrift.async.TAsyncClient *client*, org.apache.thrift.protocol.TProtocolFactory *protocolFactory*, org.apache.thrift.transport.TNonblockingTransport *transport*)

Methods

getResult

public java.lang.String **getResult**()

write_args

public void **write_args**(org.apache.thrift.protocol.TProtocol *prot*)

6.20.10 RemoteSimulationService.AsyncClient.setActivationLaserPower_call

public static class **setActivationLaserPower_call** extends org.apache.thrift.async.TAsyncMethodCall<Void>

Constructors

setActivationLaserPower_call

public **setActivationLaserPower_call**(double *power*, org.apache.thrift.async.AsyncMethodCallback<Void> *resultHandler*, org.apache.thrift.async.TAsyncClient *client*, org.apache.thrift.protocol.TProtocolFactory *protocolFactory*, org.apache.thrift.transport.TNonblockingTransport *transport*)

Methods

getResult

public Void **getResult**()

write_args

public void **write_args**(org.apache.thrift.protocol.TProtocol *prot*)

6.20.11 RemoteSimulationService.AsyncIface

public interface **AsyncIface**

Methods

getNextImage

public void **getNextImage**(org.apache.thrift.async.AsyncMethodCallback<java.nio.ByteBuffer> *resultHandler*)

getServerStatus

public void **getServerStatus**(org.apache.thrift.async.AsyncMethodCallback<java.lang.String> *resultHandler*)

getSimulationState

public void **getSimulationState**(org.apache.thrift.async.AsyncMethodCallback<java.lang.String> *resultHandler*)

setActivationLaserPower

public void **setActivationLaserPower**(double *power*, org.apache.thrift.async.AsyncMethodCallback<Void> *resultHandler*)

6.20.12 RemoteSimulationService.AsyncProcessor

public static class **AsyncProcessor**<I extends AsyncIface> extends org.apache.thrift.TBaseAsyncProcessor<I>

Constructors

AsyncProcessor

public **AsyncProcessor**(I *iface*)

AsyncProcessor

protected **AsyncProcessor**(I *iface*, java.util.Map<java.lang.String, org.apache.thrift.AsyncProcessFunction<I, ? extends org.apache.thrift.TBase, ?>> *processMap*)

6.20.13 RemoteSimulationService.AsyncProcessor.getNextImage

public static class **getNextImage**<I extends AsyncIface> extends org.apache.thrift.AsyncProcessFunction<I, *getNextImage_args*, jav

Constructors

getNextImage

public **getNextImage**()

Methods

getEmptyArgsInstance

public *getNextImage_args* **getEmptyArgsInstance**()

getResultHandler

public org.apache.thrift.async.AsyncMethodCallback<java.nio.ByteBuffer> **getResultHandler**(org.apache.thrift.server.AbstractN
fb, int
seqid)

isOneway

protected boolean **isOneway**()

start

public void **start**(I *iface*, *getNextImage_args* *args*, org.apache.thrift.async.AsyncMethodCallback<java.nio.ByteBuffer>
resultHandler)

6.20.14 RemoteSimulationService.AsyncProcessor.getServerStatus

public static class **getServerStatus**<I extends AsyncIface> extends org.apache.thrift.AsyncProcessFunction<I, *getServerStatus_a*

Constructors

getServerStatus

public **getServerStatus**()

Methods

getEmptyArgsInstance

public *getServerStatus_args* **getEmptyArgsInstance**()

getResultHandler

public org.apache.thrift.async.AsyncMethodCallback<java.lang.String> **getResultHandler**(org.apache.thrift.server.AbstractNonb
fb, int *seqid*)

isOneway

protected boolean **isOneway**()

start

public void **start** (I *iface*, *getServerStatus_args* *args*, org.apache.thrift.async.AsyncMethodCallback<java.lang.String> *resultHandler*)

6.20.15 RemoteSimulationService.AsyncProcessor.getSimulationState

public static class **getSimulationState**<I extends AsyncIface> extends org.apache.thrift.AsyncProcessFunction<I, *getSimulation*

Constructors

getSimulationState

public **getSimulationState** ()

Methods

getEmptyArgsInstance

public *getSimulationState_args* **getEmptyArgsInstance** ()

getResultHandler

public org.apache.thrift.async.AsyncMethodCallback<java.lang.String> **getResultHandler** (org.apache.thrift.server.AbstractNonb *fb*, int *seqid*)

isOneway

protected boolean **isOneway** ()

start

public void **start** (I *iface*, *getSimulationState_args* *args*, org.apache.thrift.async.AsyncMethodCallback<java.lang.String> *resultHandler*)

6.20.16 RemoteSimulationService.AsyncProcessor.setActivationLaserPower

public static class **setActivationLaserPower**<I extends AsyncIface> extends org.apache.thrift.AsyncProcessFunction<I, *setAct*

Constructors

setActivationLaserPower

public **setActivationLaserPower** ()

Methods

getEmptyArgsInstance

public *setActivationLaserPower_args* **getEmptyArgsInstance**()

getResultHandler

public org.apache.thrift.async.AsyncMethodCallback<Void> **getResultHandler**(org.apache.thrift.server.AbstractNonblockingSer
fb, int *seqid*)

isOneway

protected boolean **isOneway**()

start

public void **start**(I *iface*, *setActivationLaserPower_args* *args*, org.apache.thrift.async.AsyncMethodCallback<Void>
resultHandler)

6.20.17 RemoteSimulationService.Client

public static class **Client** extends org.apache.thrift.TServiceClient implements *Iface*

Constructors

Client

public **Client**(org.apache.thrift.protocol.TProtocol *prot*)

Client

public **Client**(org.apache.thrift.protocol.TProtocol *iprot*, org.apache.thrift.protocol.TProtocol *oprot*)

Methods

getNextImage

public java.nio.ByteBuffer **getNextImage**()

getServerStatus

public java.lang.String **getServerStatus**()

getSimulationState

public java.lang.String **getSimulationState**()

recv_getNextImage

public java.nio.ByteBuffer **recv_getNextImage**()

recv_getServerStatus

public java.lang.String **recv_getServerStatus**()

recv_getSimulationState

public java.lang.String **recv_getSimulationState**()

recv_setActivationLaserPower

public void **recv_setActivationLaserPower**()

send_getNextImage

public void **send_getNextImage**()

send_getServerStatus

public void **send_getServerStatus**()

send_getSimulationState

public void **send_getSimulationState**()

send_setActivationLaserPower

public void **send_setActivationLaserPower**(double *power*)

setActivationLaserPower

public void **setActivationLaserPower**(double *power*)

6.20.18 RemoteSimulationService.Client.Factory

public static class **Factory** implements org.apache.thrift.TServiceClientFactory<*Client*>

Constructors

Factory

public **Factory**()

Methods

getClient

public *Client* **getClient**(org.apache.thrift.protocol.TProtocol *prot*)

getClient

public *Client* **getClient**(org.apache.thrift.protocol.TProtocol *iprot*, org.apache.thrift.protocol.TProtocol *oprot*)

6.20.19 RemoteSimulationService.Iface

public interface **Iface**

Methods

getNextImage

public java.nio.ByteBuffer **getNextImage**()
Increments the simulation by one time step and returns an image.

getServerStatus

public java.lang.String **getServerStatus**()
Returns the simulation server's current status.

getSimulationState

public java.lang.String **getSimulationState**()
Returns information about the current state of each emitter in a JSON string.

setActivationLaserPower

public void **setActivationLaserPower**(double *power*)
Changes the simulation'ss fluorescence activation laser power.

Parameters

- **power** –

6.20.20 RemoteSimulationService.Processor

public static class **Processor**<I extends Iface> extends org.apache.thrift.TBaseProcessor<I> implements org.apache.thrift.TProcesso

Constructors

Processor

public **Processor**(I *iface*)

Processor

protected **Processor**(I *iface*, java.util.Map<java.lang.String, org.apache.thrift.ProcessFunction<I, ? extends org.apache.thrift.TBase>> *processMap*)

6.20.21 RemoteSimulationService.Processor.getNextImage

public static class **getNextImage**<I extends Iface> extends org.apache.thrift.ProcessFunction<I, *getNextImage_args*>

Constructors

getNextImage

public **getNextImage**()

Methods

getEmptyArgsInstance

public *getNextImage_args* **getEmptyArgsInstance**()

getResult

public *getNextImage_result* **getResult**(I *iface*, *getNextImage_args* *args*)

handleRuntimeExceptions

protected boolean **handleRuntimeExceptions**()

isOneway

protected boolean **isOneway**()

6.20.22 RemoteSimulationService.Processor.getServerStatus

public static class **getServerStatus**<I extends Iface> extends org.apache.thrift.ProcessFunction<I, *getServerStatus_args*>

Constructors

getServerStatus

public **getServerStatus**()

Methods

getEmptyArgsInstance

public *getServerStatus_args* **getEmptyArgsInstance**()

getResult

public *getServerStatus_result* **getResult**(I *iface*, *getServerStatus_args* *args*)

handleRuntimeExceptions

protected boolean **handleRuntimeExceptions**()

isOneway

protected boolean **isOneway**()

6.20.23 RemoteSimulationService.Processor.getSimulationState

public static class **getSimulationState**<I extends Iface> extends org.apache.thrift.ProcessFunction<I, *getSimulationState_args*>

Constructors

getSimulationState

public **getSimulationState**()

Methods

getEmptyArgsInstance

public *getSimulationState_args* **getEmptyArgsInstance**()

getResult

public *getSimulationState_result* **getResult** (I *iface*, *getSimulationState_args* *args*)

handleRuntimeExceptions

protected boolean **handleRuntimeExceptions** ()

isOneway

protected boolean **isOneway** ()

6.20.24 RemoteSimulationService.Processor.setActivationLaserPower

public static class **setActivationLaserPower**<I extends Iface> extends org.apache.thrift.ProcessFunction<I, *setActivationLaserI*

Constructors

setActivationLaserPower

public **setActivationLaserPower** ()

Methods

getEmptyArgsInstance

public *setActivationLaserPower_args* **getEmptyArgsInstance** ()

getResult

public *setActivationLaserPower_result* **getResult** (I *iface*, *setActivationLaserPower_args* *args*)

handleRuntimeExceptions

protected boolean **handleRuntimeExceptions** ()

isOneway

protected boolean **isOneway** ()

6.20.25 RemoteSimulationService.getNextImage_args

public static class **getNextImage_args** implements org.apache.thrift.TBase<*getNextImage_args*, *getNextImage_args*.*_Fields*>, java

Fields

metaDataMap

public static final java.util.Map<*_Fields*, org.apache.thrift.meta_data.FieldMetaData> **metaDataMap**

Constructors

getNextImage_args

public **getNextImage_args**()

getNextImage_args

public **getNextImage_args**(*getNextImage_args other*)
Performs a deep copy on *other*.

Methods

clear

public void **clear**()

compareTo

public int **compareTo**(*getNextImage_args other*)

deepCopy

public *getNextImage_args* **deepCopy**()

equals

public boolean **equals**(java.lang.Object *that*)

equals

public boolean **equals**(*getNextImage_args that*)

fieldForId

public *_Fields* **fieldForId**(int *fieldId*)

getFieldValue

public java.lang.Object **getFieldValue**(*_Fields field*)

hashCode

public int **hashCode**()

isSet

public boolean **isSet**(*_Fields field*)
Returns true if field corresponding to fieldID is set (has been assigned a value) and false otherwise

read

public void **read**(org.apache.thrift.protocol.TProtocol *iprot*)

setFieldValue

public void **setFieldValue**(*_Fields field*, java.lang.Object *value*)

toString

public java.lang.String **toString**()

validate

public void **validate**()

write

public void **write**(org.apache.thrift.protocol.TProtocol *oprot*)

6.20.26 RemoteSimulationService.getNextImage_args._Fields

public enum **_Fields** implements org.apache.thrift.TFieldIdEnum
The set of fields this struct contains, along with convenience methods for finding and manipulating them.

Enum Constants

6.20.27 RemoteSimulationService.getNextImage_result

public static class **getNextImage_result** implements org.apache.thrift.TBase<*getNextImage_result*, *getNextImage_result*.*_Fields*>

Fields

ex

public *ImageGenerationException* **ex**

metaDataMap

public static final java.util.Map<*_Fields*, org.apache.thrift.meta_data.FieldMetaData> **metaDataMap**

success

public java.nio.ByteBuffer **success**

Constructors

getNextImage_result

public **getNextImage_result**()

getNextImage_result

public **getNextImage_result**(java.nio.ByteBuffer *success*, *ImageGenerationException* *ex*)

getNextImage_result

public **getNextImage_result**(*getNextImage_result* *other*)
Performs a deep copy on *other*.

Methods

bufferForSuccess

public java.nio.ByteBuffer **bufferForSuccess**()

clear

public void **clear**()

compareTo

public int **compareTo**(*getNextImage_result* *other*)

deepCopy

public *getNextImage_result* **deepCopy**()

equals

public boolean **equals**(java.lang.Object *that*)

equals

public boolean **equals**(*getNextImage_result* *that*)

fieldForId

public *_Fields* **fieldForId**(int *fieldId*)

getEx

public *ImageGenerationException* **getEx**()

getFieldValue

public java.lang.Object **getFieldValue**(*_Fields* *field*)

getSuccess

public byte[] **getSuccess**()

hashCode

public int **hashCode**()

isSet

public boolean **isSet**(*_Fields* *field*)
Returns true if field corresponding to fieldID is set (has been assigned a value) and false otherwise

isSetEx

public boolean **isSetEx**()
Returns true if field ex is set (has been assigned a value) and false otherwise

isSetSuccess

public boolean **isSetSuccess**()
Returns true if field success is set (has been assigned a value) and false otherwise

read

public void **read**(org.apache.thrift.protocol.TProtocol *iprot*)

setEx

public *getNextImage_result* **setEx**(*ImageGenerationException ex*)

setExIsSet

public void **setExIsSet**(boolean *value*)

setFieldValue

public void **setFieldValue**(*_Fields field*, java.lang.Object *value*)

setSuccess

public *getNextImage_result* **setSuccess**(byte[] *success*)

setSuccess

public *getNextImage_result* **setSuccess**(java.nio.ByteBuffer *success*)

setSuccessIsSet

public void **setSuccessIsSet**(boolean *value*)

toString

public java.lang.String **toString**()

unsetEx

public void **unsetEx**()

unsetSuccess

public void **unsetSuccess**()

validate

public void **validate**()

write

public void **write**(org.apache.thrift.protocol.TProtocol *oprot*)

6.20.28 RemoteSimulationService.getNextImage_result._Fields

public enum **_Fields** implements org.apache.thrift.TFieldIdEnum
The set of fields this struct contains, along with convenience methods for finding and manipulating them.

Enum Constants

EX

public static final *RemoteSimulationService.getNextImage_result._Fields* **EX**

SUCCESS

public static final *RemoteSimulationService.getNextImage_result._Fields* **SUCCESS**

6.20.29 RemoteSimulationService.getServerStatus_args

public static class **getServerStatus_args** implements org.apache.thrift.TBase<*getServerStatus_args*, *getServerStatus_args._Fiel*

Fields

metaDataMap

public static final java.util.Map<*_Fields*, org.apache.thrift.meta_data.FieldMetaData> **metaDataMap**

Constructors

getServerStatus_args

public **getServerStatus_args**()

getServerStatus_args

public **getServerStatus_args**(*getServerStatus_args other*)
Performs a deep copy on *other*.

Methods

clear

public void **clear**()

compareTo

public int **compareTo**(*getServerStatus_args other*)

deepCopy

public *getServerStatus_args* **deepCopy**()

equals

public boolean **equals**(java.lang.Object *that*)

equals

public boolean **equals**(*getServerStatus_args that*)

fieldForId

public *_Fields* **fieldForId**(int *fieldId*)

getFieldValue

public java.lang.Object **getFieldValue**(*_Fields field*)

hashCode

public int **hashCode**()

isSet

public boolean **isSet**(*_Fields field*)
Returns true if field corresponding to fieldID is set (has been assigned a value) and false otherwise

read

public void **read**(org.apache.thrift.protocol.TProtocol *iprot*)

setFieldValue

public void **setFieldValue**(*_Fields field*, java.lang.Object *value*)

toString

public java.lang.String **toString**()

validate

public void **validate**()

write

public void **write**(org.apache.thrift.protocol.TProtocol *oprot*)

6.20.30 RemoteSimulationService.getServerStatus_args._Fields

public enum **_Fields** implements org.apache.thrift.TFieldIdEnum
The set of fields this struct contains, along with convenience methods for finding and manipulating them.

Enum Constants

6.20.31 RemoteSimulationService.getServerStatus_result

public static class **getServerStatus_result** implements org.apache.thrift.TBase<*getServerStatus_result*, *getServerStatus_result*

Fields

metaDataMap

public static final java.util.Map<*_Fields*, org.apache.thrift.meta_data.FieldMetaData> **metaDataMap**

success

public java.lang.String **success**

Constructors

getServerStatus_result

public **getServerStatus_result**()

getServerStatus_result

public **getServerStatus_result** (java.lang.String *success*)

getServerStatus_result

public **getServerStatus_result** (*getServerStatus_result* *other*)
Performs a deep copy on *other*.

Methods

clear

public void **clear** ()

compareTo

public int **compareTo** (*getServerStatus_result* *other*)

deepCopy

public *getServerStatus_result* **deepCopy** ()

equals

public boolean **equals** (java.lang.Object *that*)

equals

public boolean **equals** (*getServerStatus_result* *that*)

fieldForId

public *_Fields* **fieldForId** (int *fieldId*)

getFieldValue

public java.lang.Object **getFieldValue** (*_Fields* *field*)

getSuccess

public java.lang.String **getSuccess** ()

hashCode

public int **hashCode**()

isSet

public boolean **isSet**(*_Fields field*)
Returns true if field corresponding to fieldID is set (has been assigned a value) and false otherwise

isSetSuccess

public boolean **isSetSuccess**()
Returns true if field success is set (has been assigned a value) and false otherwise

read

public void **read**(org.apache.thrift.protocol.TProtocol *iprot*)

setFieldValue

public void **setFieldValue**(*_Fields field*, java.lang.Object *value*)

setSuccess

public *getServerStatus_result* **setSuccess**(java.lang.String *success*)

setSuccessIsSet

public void **setSuccessIsSet**(boolean *value*)

toString

public java.lang.String **toString**()

unsetSuccess

public void **unsetSuccess**()

validate

public void **validate**()

write

public void **write** (org.apache.thrift.protocol.TProtocol *oprot*)

6.20.32 RemoteSimulationService.getServerStatus_result._Fields

public enum **_Fields** implements org.apache.thrift.TFieldIdEnum
The set of fields this struct contains, along with convenience methods for finding and manipulating them.

Enum Constants

SUCCESS

public static final *RemoteSimulationService.getServerStatus_result._Fields* **SUCCESS**

6.20.33 RemoteSimulationService.getSimulationState_args

public static class **getSimulationState_args** implements org.apache.thrift.TBase<*getSimulationState_args*, *getSimulationState*

Fields

metaDataMap

public static final java.util.Map<*_Fields*, org.apache.thrift.meta_data.FieldMetaData> **metaDataMap**

Constructors

getSimulationState_args

public **getSimulationState_args** ()

getSimulationState_args

public **getSimulationState_args** (*getSimulationState_args other*)
Performs a deep copy on *other*.

Methods

clear

public void **clear** ()

compareTo

public int **compareTo** (*getSimulationState_args other*)

deepCopy

public *getSimulationState_args* **deepCopy**()

equals

public boolean **equals**(java.lang.Object *that*)

equals

public boolean **equals**(*getSimulationState_args* *that*)

fieldForId

public *_Fields* **fieldForId**(int *fieldId*)

getFieldValue

public java.lang.Object **getFieldValue**(*_Fields* *field*)

hashCode

public int **hashCode**()

isSet

public boolean **isSet**(*_Fields* *field*)
Returns true if field corresponding to fieldID is set (has been assigned a value) and false otherwise

read

public void **read**(org.apache.thrift.protocol.TProtocol *iprot*)

setFieldValue

public void **setFieldValue**(*_Fields* *field*, java.lang.Object *value*)

toString

public java.lang.String **toString**()

validate

public void **validate**()

write

public void **write** (org.apache.thrift.protocol.TProtocol *oprot*)

6.20.34 RemoteSimulationService.getSimulationState_args._Fields

public enum **_Fields** implements org.apache.thrift.TFieldIdEnum
The set of fields this struct contains, along with convenience methods for finding and manipulating them.

Enum Constants

6.20.35 RemoteSimulationService.getSimulationState_result

public static class **getSimulationState_result** implements org.apache.thrift.TBase<*getSimulationState_result*, *getSimulationS*

Fields

metaDataMap

public static final java.util.Map<*_Fields*, org.apache.thrift.meta_data.FieldMetaData> **metaDataMap**

success

public java.lang.String **success**

Constructors

getSimulationState_result

public **getSimulationState_result** ()

getSimulationState_result

public **getSimulationState_result** (java.lang.String *success*)

getSimulationState_result

public **getSimulationState_result** (*getSimulationState_result other*)
Performs a deep copy on *other*.

Methods

clear

public void **clear** ()

compareTo

public int **compareTo**(*getSimulationState_result other*)

deepCopy

public *getSimulationState_result* **deepCopy**()

equals

public boolean **equals**(java.lang.Object *that*)

equals

public boolean **equals**(*getSimulationState_result that*)

fieldForId

public *_Fields* **fieldForId**(int *fieldId*)

getFieldValue

public java.lang.Object **getFieldValue**(*_Fields field*)

getSuccess

public java.lang.String **getSuccess**()

hashCode

public int **hashCode**()

isSet

public boolean **isSet**(*_Fields field*)
Returns true if field corresponding to fieldID is set (has been assigned a value) and false otherwise

isSetSuccess

public boolean **isSetSuccess**()
Returns true if field success is set (has been assigned a value) and false otherwise

read

public void **read**(org.apache.thrift.protocol.TProtocol *iprot*)

setFieldValue

public void **setFieldValue**(*_Fields field*, java.lang.Object *value*)

setSuccess

public *getSimulationState_result* **setSuccess**(java.lang.String *success*)

setSuccessIsSet

public void **setSuccessIsSet**(boolean *value*)

toString

public java.lang.String **toString**()

unsetSuccess

public void **unsetSuccess**()

validate

public void **validate**()

write

public void **write**(org.apache.thrift.protocol.TProtocol *oprot*)

6.20.36 RemoteSimulationService.getSimulationState_result._Fields

public enum **_Fields** implements org.apache.thrift.TFieldIdEnum
The set of fields this struct contains, along with convenience methods for finding and manipulating them.

Enum Constants

SUCCESS

public static final *RemoteSimulationService.getSimulationState_result._Fields* **SUCCESS**

6.20.37 RemoteSimulationService.setActivationLaserPower_args

public static class **setActivationLaserPower_args** implements org.apache.thrift.TBase<*setActivationLaserPower_args*, *setAc*

Fields

metaDataMap

public static final java.util.Map<*_Fields*, org.apache.thrift.meta_data.FieldMetaData> **metaDataMap**

power

public double **power**

Constructors

setActivationLaserPower_args

public **setActivationLaserPower_args** ( )

setActivationLaserPower_args

public **setActivationLaserPower_args** (double *power*)

setActivationLaserPower_args

public **setActivationLaserPower_args** (*setActivationLaserPower_args other*)
Performs a deep copy on *other*.

Methods

clear

public void **clear** ( )

compareTo

public int **compareTo** (*setActivationLaserPower_args other*)

deepCopy

public *setActivationLaserPower_args* **deepCopy** ( )

equals

public boolean **equals** (java.lang.Object *that*)

equals

public boolean **equals** (*setActivationLaserPower_args* *that*)

fieldForId

public *_Fields* **fieldForId** (int *fieldId*)

getFieldValue

public java.lang.Object **getFieldValue** (*_Fields* *field*)

getPower

public double **getPower** ()

hashCode

public int **hashCode** ()

isSet

public boolean **isSet** (*_Fields* *field*)
Returns true if field corresponding to fieldID is set (has been assigned a value) and false otherwise

isSetPower

public boolean **isSetPower** ()
Returns true if field power is set (has been assigned a value) and false otherwise

read

public void **read** (org.apache.thrift.protocol.TProtocol *iprot*)

setFieldValue

public void **setFieldValue** (*_Fields* *field*, java.lang.Object *value*)

setPower

public *setActivationLaserPower_args* **setPower**(double *power*)

setPowerIsSet

public void **setPowerIsSet**(boolean *value*)

toString

public java.lang.String **toString**()

unsetPower

public void **unsetPower**()

validate

public void **validate**()

write

public void **write**(org.apache.thrift.protocol.TProtocol *oprot*)

6.20.38 RemoteSimulationService.setActivationLaserPower_args._Fields

public enum **_Fields** implements org.apache.thrift.TFieldIdEnum
The set of fields this struct contains, along with convenience methods for finding and manipulating them.

Enum Constants

POWER

public static final *RemoteSimulationService.setActivationLaserPower_args._Fields* **POWER**

6.20.39 RemoteSimulationService.setActivationLaserPower_result

public static class **setActivationLaserPower_result** implements org.apache.thrift.TBase<*setActivationLaserPower_result*, *s*

Fields

metaDataMap

public static final java.util.Map<*_Fields*, org.apache.thrift.meta_data.FieldMetaData> **metaDataMap**

Constructors

setActivationLaserPower_result

public **setActivationLaserPower_result**()

setActivationLaserPower_result

public **setActivationLaserPower_result**(*setActivationLaserPower_result other*)
Performs a deep copy on *other*.

Methods

clear

public void **clear**()

compareTo

public int **compareTo**(*setActivationLaserPower_result other*)

deepCopy

public *setActivationLaserPower_result* **deepCopy**()

equals

public boolean **equals**(java.lang.Object *that*)

equals

public boolean **equals**(*setActivationLaserPower_result that*)

fieldForId

public *_Fields* **fieldForId**(int *fieldId*)

getFieldValue

public java.lang.Object **getFieldValue**(*_Fields field*)

hashCode

public int **hashCode**()

isSet

public boolean **isSet** (*_Fields field*)
Returns true if field corresponding to fieldID is set (has been assigned a value) and false otherwise

read

public void **read** (org.apache.thrift.protocol.TProtocol *iprot*)

setFieldValue

public void **setFieldValue** (*_Fields field*, java.lang.Object *value*)

toString

public java.lang.String **toString** ()

validate

public void **validate** ()

write

public void **write** (org.apache.thrift.protocol.TProtocol *oprot*)

6.20.40 RemoteSimulationService.setActivationLaserPower_result._Fields

public enum **_Fields** implements org.apache.thrift.TFieldIdEnum
The set of fields this struct contains, along with convenience methods for finding and manipulating them.

Enum Constants

6.20.41 RemoteSimulationServiceHandler

public class **RemoteSimulationServiceHandler** implements *RemoteSimulationService.Iface*
Implements the remote simulation service functions.

Author Kyle M. Douglass

Constructors

RemoteSimulationServiceHandler

public **RemoteSimulationServiceHandler** (*Simulator simulator*)

Methods

getNextImage

public ByteBuffer **getNextImage**()
Advances the simulator by one time step and returns the image.

Throws

- ***ch.epfl.leb.sass.server.ImageGenerationException*** –

Returns A buffer containing the TIFF-encoded byte string of the simulator's next image.

getServerStatus

public String **getServerStatus**()
This method is used to determine whether the server is running.

Returns Basic information concerning the status of the server.

getSimulationState

public String **getSimulationState**()
Collects information about the simulation's current state and returns it.

Returns JSON string containing the current state of the simulation.

setActivationLaserPower

public void **setActivationLaserPower**(double *power*)
Sets the activation laser power in the simulation.

Parameters

- **power** – The power of the laser.

6.20.42 RemoteSimulationServiceHandlerTest

public class **RemoteSimulationServiceHandlerTest**

Author kmdouglass

Constructors

RemoteSimulationServiceHandlerTest

public **RemoteSimulationServiceHandlerTest**()

Methods

testGetNextImage

public void **testGetNextImage**()
Test of getNextImage method, of class RemoteSimulationServiceHandler.

testGetServerStatus

public void **testGetServerStatus**()
Test of getServerStatus method, of class RemoteSimulationServiceHandler.

testGetSimulationState

public void **testGetSimulationState**()
Test of getSimulationState method, of class RemoteSimulationServiceHandler.

6.21 ch.epfl.leb.sass.simulator

6.21.1 Simulator

public interface **Simulator**
The interface that defines everything that a Simulator should do.

Author Marcel Stefko, Kyle M. Douglass

Methods

getControlSignal

public double **getControlSignal**()
Returns currently set control signal of the generator (e.g. laser power settings).

Returns control signal value

getCustomParameters

public HashMap<String, Double> **getCustomParameters**()
Returns custom parameters of the generator.

Returns map of custom parameters

getFOVSize

public double **getFOVSize**()

Returns FOV size in square micrometers

getImageCount

public int **getImageCount**()

Returns the number of images simulated. Because the simulation can advance without generating an image, this value will be less than or equal to the number of simulation time steps. Use *incrementTimeStep()* to advance the simulation one time step without generating an image.

Returns The number of images that have been simulated.

getNextImage

public *ImageS* **getNextImage**()

Generates a new image and adds it to the internal stack.

Returns newly generated image

getObjectSpacePixelSize

public double **getObjectSpacePixelSize**()

Returns length of one pixel side in micrometers

getShortTrueSignalDescription

public String **getShortTrueSignalDescription**()

Returns A short description of the truth signal, typically its units.

getSimulationState

public String **getSimulationState**()

Retrieves the current state of the simulation. This returns the simulation's current state, which includes all relevant properties. These may include, for example, the fluorescence state of every fluorophore.

Returns JSON string encoding the simulation state.

getStack

public *ImageS* **getStack**()

Returns internal stack with all generated images.

Returns internal stack

getTrueSignal

public double **getTrueSignal**(int *image_no*)

Returns the actual value of signal (if applicable) for given image.

Parameters

- **image_no** – 1-based image number in history

Returns value of signal (e.g. no. of active emitters)

incrementTimeStep

public void **incrementTimeStep**()
Increments the simulation by one time step without creating an image.

saveStack

public void **saveStack**(File *selectedFile*)
Saves .tif stack to selected file.

Parameters

- **selectedFile** – file to save to

setControlSignal

public void **setControlSignal**(double *value*)
Sets control signal of the generator (e.g. laser power). This should be used by the controller.

Parameters

- **value** – new value of the control signal

setCustomParameters

public void **setCustomParameters**(HashMap<String, Double> *map*)
Sets custom parameters of the generator.

Parameters

- **map** – map of custom parameters

6.22 ch.epfl.leb.sass.simulator.internal

6.22.1 AbstractSimulator

public abstract class **AbstractSimulator** implements *Simulator*

Author Marcel Stefko

Fields

parameters

protected HashMap<String, Double> **parameters**
Map of custom parameters for the generator.

stack

protected *ImageS* **stack**
Stack to which the generated images are appended.

Constructors

AbstractSimulator

public **AbstractSimulator**()
Initializes the empty parameters map.

Methods

getImageCount

public int **getImageCount**()

getSimulationState

public String **getSimulationState**()

getStack

public *ImageS* **getStack**()

saveStack

public void **saveStack**(File *file*)

6.22.2 DefaultSimulator

public class **DefaultSimulator** extends *AbstractSimulator*
The basic simulation engine from which others may be derived.

Author Marcel Stefko

Constructors

DefaultSimulator

public **DefaultSimulator**(*Microscope microscope*)
Initialize the generator.

Parameters

- **microscope** –

Methods

getControlSignal

public double **getControlSignal**()

getCustomParameters

public HashMap<String, Double> **getCustomParameters**()

getFOVSize

public double **getFOVSize**()

Returns The size of the FOV in square object-space units.

getNextImage

public *ImageS* **getNextImage**()

getObjectSpacePixelSize

public double **getObjectSpacePixelSize**()

Returns Length of one pixel side in object-space units.

getShortTrueSignalDescription

public String **getShortTrueSignalDescription**()

getTrueSignal

public double **getTrueSignal**(int *image_no*)

incrementTimeStep

public void **incrementTimeStep**()
Advance the simulation by one time step (i.e. one frame). Simulates a frame but does not create an image.

setControlSignal

public void **setControlSignal**(double *value*)

setCustomParameters

public void **setCustomParameters**(HashMap<String, Double> *map*)

6.22.3 ImageJSimulator

public class **ImageJSimulator** extends *DefaultSimulator*

The default simulator that is run as, for example, the ImageJ plugin.

Author Marcel Stefko

Fields

TIMEPERFRAME

protected final long **TIMEPERFRAME**

The time duration of each frame. This is here only for compatibility with ALICA's analyzers, which require a time argument.

analyzer

protected final Analyzer **analyzer**

Analyzer which analyzes generated images

controller

protected final Controller **controller**

Takes the output of a single analyzer, processes it, and outputs a signal to the generator, for feedback loop control.

history

protected HashMap<Integer, JSONObject> **history**

Records of values of output of analyzer, controller.

image_count

protected int **image_count**

Number of already-generated images.

positionLogger

protected *PositionLogger* **positionLogger**

Logs the ground truth positions of the molecules.

stateLogger

protected *StateLogger* **stateLogger**

Logs the state transitions of the molecules.

Constructors

ImageJSimulator

public **ImageJSimulator**(*Microscope microscope*, Analyzer *analyzer*, Controller *controller*)
Initialize the simulator from user-specified components.

Parameters

- **microscope** – The microscope to be simulated.
- **analyzer** – An analyzer for processing images from the microscope.
- **controller** – A controller that adjusts the state of the microscope.

Methods

execute

public *ImageS* **execute**(int *no_of_images*, int *controller_refresh_rate*, String *csv_save_path*, String *tiff_save_path*)
An example simulation

Parameters

- **no_of_images** –
- **controller_refresh_rate** –
- **csv_save_path** –
- **tiff_save_path** –

getImageCount

public int **getImageCount**()
Returns the number of generated images since simulation start.

Returns number of generated images

getPositionLogger

public *PositionLogger* **getPositionLogger**()

Returns The emitter position logger.

getStateLogger

public *StateLogger* **getStateLogger**()

Returns The state transition logger.

incrementCounter

public void **incrementCounter**()
Increments image counter in case an image was generated outside of this class.

saveStack

public void **saveStack**(File *tiff_file*)
Save current ImageStack to TIFF file

Parameters

- **tiff_file** – file to save to

saveToCsv

public void **saveToCsv**(File *file*)
Saves the data for generator, analyzer and controller for each frame into a .csv file

Parameters

- **file** – destination csv file

6.22.4 RPCSimulator

public class **RPCSimulator** extends *DefaultSimulator*
A simulator that is specialized for control by remote procedure calls (RPCs).

Author Kyle M. Douglass

Constructors

RPCSimulator

public **RPCSimulator**(*Microscope microscope*)
Initializes the SimpleSimulator and connects it to the simulation engine.

Parameters

- **microscope** – The engine that runs the simulation.

Methods

getSimulationState

public String **getSimulationState**()
Returns the simulation's current state as a JSON-encoded string.

Returns JSON string containing information about the simulation state.

6.23 ch.epfl.leb.sass.utils

6.23.1 RNG

public final class **RNG**
Random number generator for STORMsim. Ensures repeatability.

Author stefko

Methods

getGammaGenerator

public static Gamma **getGammaGenerator**()

Returns Gamma distribution RNG

getGaussianGenerator

public static Normal **getGaussianGenerator**()

Returns Gaussian distribution RNG

getPoissonGenerator

public static Poisson **getPoissonGenerator**()

Returns Poisson RNG

getUniformGenerator

public static Random **getUniformGenerator**()

Returns uniform RNG

setSeed

public static void **setSeed**(int *seed*)
This resets the generators

Parameters

- **seed** –

6.23.2 TiffParser

public class **TiffParser**
Parses the ImageStack into RAM out of a .tiff file.

Author Marcel Stefko

Methods

loadGeneralTiff

public final ImageStack **loadGeneralTiff** (File *file*)
Loads a tiff stack from a file on disk into RAM

Parameters

- **file** – tiff file to be loaded

Returns loaded image stack

6.24 ch.epfl.leb.sass.utils.images

6.24.1 ImageS

public interface **ImageS**
An abstraction layer for a 3-dimensional, 16-bit image stack in SASS. This interface allows developers to more easily substitute other backends for image data into SASS. For example, one could write an implementation for ImgLib2 datatypes to replace ImageJ's original ImageStack. This interface should be used everywhere image data is passed between SASS components.

Author Kyle M. Douglass

Methods

addImage

public void **addImage** (short[][] *image*)
Adds a single image to the dataset. This method accepts a 2D array of pixels and adds it to the end of the dataset. The size of the image in X and Y must be the same as the existing images.

Parameters

- **image** – The image data to add to the dataset.

Throws

- *ch.epfl.leb.sass.utils.images.ImageShapeException* –

addImage

public void **addImage** (int[][] *image*)
Adds a single image to the dataset. This method accepts a 2D array of pixels and adds it to the end of the dataset. The size of the image in X and Y must be the same as the existing images. Integer data will be truncated into shorts.

Parameters

- **image** – The image data to add to the dataset.

Throws

- *ch.epfl.leb.sass.utils.images.ImageShapeException* –

addImage

public void **addImage**(float[][] *image*)

Adds a single image to the dataset. This method accepts a 2D array of pixels and adds it to the end of the dataset. The size of the image in X and Y must be the same as the existing images. Float data will be truncated into shorts.

Parameters

- **image** – The image data to add to the dataset.

Throws

- ***ch.epfl.leb.sass.utils.images.ImageShapeException*** –

concatenate

public void **concatenate**(*ImageS dataset*)

Appends another ImageS dataset to the end of this one.

Parameters

- **dataset** – The images to add to the dataset.

Throws

- ***ch.epfl.leb.sass.utils.images.ImageShapeException*** –

getBitDepth

public int **getBitDepth**()

Returns the bit depth of the pixels.

Returns The bit depth of the pixels.

getHeight

public int **getHeight**()

Returns the height of the images in the dataset.

Returns The height of the images in the dataset.

getPixelData

public short[] **getPixelData**(int *index*)

Returns the image data at the slice corresponding to index.

Parameters

- **index** –

getSize

public int **getSize**()
Returns the number of images in the dataset.

Returns The number of images in the dataset.

getSlice

public int **getSlice**()
Gets the active slice of the dataset (0-indexed). This is the image that will be displayed in the viewer.

Returns The index of the current slice.

getTitle

public String **getTitle**()
Returns the title (or, equivalently, the name) of the image dataset.

Returns The title of the dataset.

getWidth

public int **getWidth**()
Returns the width of the images in the dataset.

Returns The width of the images in the dataset.

saveAsTiffStack

public void **saveAsTiffStack**(File *file*)
Saves the images to a TIFF file.

Parameters

- **file** – The TIFF file where the dataset will be saved.

serializeToArray

public byte[] **serializeToArray**()
Serializes the dataset into a TIFF-encoded byte array.

Returns The image data encoded as a TIFF-file byte string.

serializeToBuffer

public ByteBuffer **serializeToBuffer**()
Returns a buffer containing the dataset in a TIFF-encoded byte array.

Returns A ByteBuffer containing the TIFF-encoded dataset.

setSlice

public void **setSlice**(int *index*)
Sets the active slice of the dataset (0-indexed). * This is the image that will be displayed in the viewer.

Parameters

- **index** – The index of the slice to activate.

setTitle

public void **setTitle**(String *title*)
Sets the title (or, equivalently, the name) of the dataset.

Parameters

- **title** – The title to give to the image dataset.

updateView

public void **updateView**()
Updates the dataset viewer to show the currently active slice.

view

public void **view**()
Displays the images.

6.24.2 ImageShapeException

public class **ImageShapeException** extends Exception
Raised when trying to add data to ImageS datasets of the wrong XY shape.

Author Kyle M. Douglass

Constructors

ImageShapeException

public **ImageShapeException**()

ImageShapeException

public **ImageShapeException**(String *message*)

6.25 ch.epfl.leb.sass.utils.images.internal

6.25.1 DefaultImageS

public class **DefaultImageS** implements *ImageS*

The default implementation of the ImageS interface. The default implementation currently wraps ImageJ1's ImageStack class. See https://imagej.nih.gov/ij/developer/api/ij/ImagePlus.html for more information.

Author Kyle M. Douglass

Constructors

DefaultImageS

public **DefaultImageS**(int *width*, int *height*)

Creates a new and empty DefaultImageS.

DefaultImageS

public **DefaultImageS**(int[][] *pixels*)

Creates a new DefaultImageS object from a 2D array of ints. The first index of the input array should correspond to x; the second corresponds to y.

Parameters

- **pixels** – The 2D array of pixel values.

DefaultImageS

public **DefaultImageS**(float[][] *pixels*)

Creates a new DefaultImageS object from a 2D array of floats. The first index of the input array should correspond to x; the second corresponds to y.

Parameters

- **pixels** – The 2D array of pixel values.

Methods

addImage

public void **addImage**(short[][] *image*)

Adds a 2D array of shorts to the dataset.

Parameters

- **image** – A 2D array of shorts.

addImage

public void **addImage**(int[][] *image*)
Converts a 2D array of ints to 16-bit shorts and adds it to the dataset.

Parameters

- **image** – A 2D array of ints indexed by xy.

Throws

- ***ch.epfl.leb.sass.utils.images.ImageShapeException*** –

addImage

public void **addImage**(float[][] *image*)
Converts a 2D array of floats to 16-bit shorts and adds it to the dataset.

Parameters

- **image** – A 2D array of floats indexed by xy.

Throws

- ***ch.epfl.leb.sass.utils.images.ImageShapeException*** –

concatenate

public void **concatenate**(*ImageS dataset*)
Appends another ImageS dataset to the end of this one.

Parameters

- **dataset** – The images to add to the dataset.

getBitDepth

public int **getBitDepth**()

getHeight

public int **getHeight**()
Returns the height of the images in the dataset.

Returns The height of the images in the dataset.

getPixelData

public short[] **getPixelData**(int *index*)
Returns the pixel data at the given index as a 1D array.

Parameters

- **index** – The index of the corresponding slice.

Returns The pixel data at the provided index.

getSize

public int **getSize**()
Returns the number of images in the dataset.

Returns The number of images in the dataset.

getSlice

public int **getSlice**()
Gets the active slice of the dataset (0-indexed). This is the image that will be displayed in the viewer.

Returns The index of the active slice.

getTitle

public String **getTitle**()
Returns the title of the image stack.

Returns The title of the image stack.

getWidth

public int **getWidth**()
Returns the width of the images in the dataset.

Returns The width of the images in the dataset.

saveAsTiffStack

public void **saveAsTiffStack**(File *file*)
Saves the images to a TIFF file.

serializeToArray

public byte[] **serializeToArray**()
Serializes the image stack to a TIFF-encoded byte array.

Returns A TIFF-encoded byte array.

serializeToBuffer

public ByteBuffer **serializeToBuffer**()
Returns a buffer containing the dataset in a TIFF-encoded byte array.

Returns A buffer containing the dataset in a TIFF-encoded byte array.

setSlice

public void **setSlice**(int *index*)
Sets the active slice of the dataset (0-indexed). This is the image that will be displayed in the viewer.

Parameters

- **index** – The index of the slice to activate.

setTitle

public void **setTitle**(String *title*)
Sets the title of the image stack.

Parameters

- **title** – The title of the image stack.

updateView

public void **updateView**()
Updates the dataset viewer to show the currently active slice.

view

public void **view**()
Displays the images in a ImagePlus window.

6.25.2 DefaultImageSTest

public class **DefaultImageSTest**
Test suite for DefaultImageS.

Author Kyle M. Douglass

Fields

instance

DefaultImageS **instance**

tempDir

public TemporaryFolder **tempDir**

Methods

setUp

public void **setUp**()

testAddImage_floatArrArr

public void **testAddImage_floatArrArr**()
Test of addImage method, of class DefaultImageS.

testAddImage_floatArrArr_wrongSize

public void **testAddImage_floatArrArr_wrongSize**()
Test of addImage method, of class DefaultImageS.

testAddImage_intArrArr

public void **testAddImage_intArrArr**()
Test of addImage method, of class DefaultImageS.

testAddImage_intArrArr_wrongSize

public void **testAddImage_intArrArr_wrongSize**()
Test of addImage method, of class DefaultImageS.

testAddImage_shortArrArr

public void **testAddImage_shortArrArr**()
Test of addImage method, of class DefaultImageS.

testAddImage_shortArrArr_wrongSize

public void **testAddImage_shortArrArr_wrongSize**()
Test of addImage method, of class DefaultImageS.

testConcatenate

public void **testConcatenate**()
Test of concatenate method, of class DefaultImageS.

testConcatenate_wrongSize

public void **testConcatenate_wrongSize**()
Test of concatenate method, of class DefaultImageS.

testGetBitDepth

public void **testGetBitDepth**()
Test of getBitDepth method, of class DefaultImageS.

testGetHeight

public void **testGetHeight**()
Test of getHeight method, of class DefaultImageS.

testGetPixelData

public void **testGetPixelData**()
Test of getPixelData method, of class DefaultImageS.

testGetSize

public void **testGetSize**()
Test of getSize method, of class DefaultImageS.

testGetSlice

public void **testGetSlice**()
Test of getSlice method, of class DefaultImageS.

testGetTitle

public void **testGetTitle**()
Test of getTitle method, of class DefaultImageS.

testGetWidth

public void **testGetWidth**()
Test of getWidth method, of class DefaultImageS.

testSaveAsTiffStack

public void **testSaveAsTiffStack**()
Test of saveAsTiffStack method, of class DefaultImageS.

testSaveAsTiffStackEmpty

public void **testSaveAsTiffStackEmpty**()
Test of saveAsTiffStack method, of class DefaultImageS.

testSerializeToArray

public void **testSerializeToArray**()
Test of serializeToArray method, of class DefaultImageS.

testSerializeToBuffer

public void **testSerializeToBuffer**()
Test of serializeToBuffer method, of class DefaultImageS.

testSetSlice

public void **testSetSlice**()
Test of setSlice method, of class DefaultImageS.

testSetTitle

public void **testSetTitle**()
Test of setTitle method, of class DefaultImageS.

CHAPTER 7

About

SASS is an open-source Fiji plugin for simulating localization microscopy experiments and fluorophore photophysics.

CHAPTER 8

Acknowledgements

8.1 Authors

- Marcel Štefko
- Kyle M. Douglass
- Baptiste Ottino

CHAPTER 9

See Also

- ALICA - Automated Laser Illumination Control Algorithm

CHAPTER 10

Indices and tables

- genindex
- modindex
- search

Index

Symbols

A

B

C

D

E

F

G

H

I

K

L

M

N

O

T

U

V

W

X

Y

Z