ALICA_interfaces Documentation

Release 0.0.0

Marcel Stefko, Kyle M. Douglass

May 01, 2018

Contents

Interfaces to the primary components of ALICA.

CHAPTER 1

Javadoc

1.1 ch.epfl.leb.alica.interfaces

1.1.1 AbstractFactory

public abstract class **AbstractFactory**<ProductSetupPanel>

A template for factories that are set up using a JPanel. This template is currently used by the Analyzer and Controller interfaces.

Author Marcel Stefko

Parameters

- **<ProductSetupPanel>** – A JPanel for setting up the parameters.

Fields

selected_name

String **selected_name**

The name of analyzer/controller.

Constructors

AbstractFactory

public **AbstractFactory**()

Initializes the map which stores different setup panels.

Methods

addSetupPanel

protected void **addSetupPanel** (String *name*, ProductSetupPanel *panel*)
Adds a new setup panel to the list.

Parameters

- **name** – ID of the product belonging to the panel and displayed in the GUI.
- **panel** – The JPanel to setup the analyzer/controller.

getProductNameList

public Set<String> **getProductNameList** ()
Returns a list of all possible analyzer/controller names.

Returns A list of all possible product keys.

getProductSetupPanelCollection

public Collection<ProductSetupPanel> **getProductSetupPanelCollection** ()
Returns a group of all the possible setup panels for the available analyzers/controllers.

Returns A collection of all possible product setup panels.

getSelectedProductName

public String **getSelectedProductName** ()
Returns the name of the selected analyzer/controller.

Returns The name of currently selected product.

getSelectedSetupPanel

public ProductSetupPanel **getSelectedSetupPanel** ()
Returns the setup panel of the currently selected analyzer/controller.

Returns The setup panel of currently selected analyzer/product.

selectProduct

public void **selectProduct** (String *name*)
Selects a product from factory.

Parameters

- **name** – The product identifier.

1.1.2 Analyzer

public interface **Analyzer**

Processes an image or series of images. Analyzers compute quantities from an image or series of images that are relevant for real-time control. These quantities can include, for example, the number of fluorescent molecules or the resolution.

Author Marcel Stefko

Methods

dispose

public void **dispose**()

Close all windows that were opened and are owned by this analyzer.

getBatchOutput

public double **getBatchOutput**()

Returns the output of the analyzer for Controller input. If no output can be provided (such as when there are no new analyzed images), Double.NaN should be returned. This method can also modify the internal state of the analyzer (such as flushing the intermittent value cache).

Returns The analyzer's current output value for use by the controller

See also: *ch.epfl.leb.alica.interfaces.Controller*

getIntermittentOutput

public double **getIntermittentOutput**()

Returns the intermittent output of the analyzer. This is used to read the current output of the analyzer; its internal state is unchanged. *getBatchOutput*() should be used to obtain output that will be passed to the controller.

Returns The analyzer's current output value.

getName

public String **getName**()

Returns a unique name for the analyzer.

Returns The analyzer's name.

getShortReturnDescription

public String **getShortReturnDescription**()

Returns a short description of the values returned by the analyzer.

Returns A short description of the values returned by the analyzer.

getStatusPanel

public *AnalyzerStatusPanel* **getStatusPanel**()

Returns the analyzer's status panel that will be displayed in the GUI. If no panel is implemented, this method should return null. In this case, the corresponding space in the MonitorGUI will appear blank.

Returns The status panel of the analyzer, or null.

processImage

public void **processImage**(Object *image*, int *image_width*, int *image_height*, double *pixel_size_um*, long *time_ms*)

Processes an image and adjust the analyzer's internal state to reflect the results of the calculation. This method is called after each new image acquisition by the AnalysisWorker. You can use the synchronized(this) statement within the body of an implementation of an Analyzer to ensure that no output readout happens during code execution. When implementing new Analyzers, try to keep the computation time as short as possible.

Parameters

- **image** – The image to be processed as 1D raw pixel data.
- **image_width** – Image width in pixels.
- **image_height** – Image height in pixels.
- **pixel_size_um** – Length of a side of a square pixel in micrometers.
- **time_ms** – Image acquisition time in milliseconds.

setROI

public void **setROI**(Roi *roi*)

Sets the region of interest in the image so that only a portion of the whole image is analyzed.

Parameters

- **roi** – The Roi object corresponding to the region.

See also: `ij.gui.Roi`

1.1.3 Controller

public interface **Controller**

Adjusts control parameters in response to values provided by the analyzer. A controller receives the output of the analyzer as input, and then adjusts its internal state accordingly. It can be asked for its state at any time by the WorkerThread.

Author Marcel Stefko

Methods

getCurrentOutput

public double **getCurrentOutput**()

Returns an output value derived from the controller's internal state.

Returns The controller's new output value for the control signal.

getName

public String **getName**()
Returns a unique name for the controller.

Returns A unique name for the controller.

getSetpoint

public double **getSetpoint**()
Returns the current value for the setpoint.

Returns The current setpoint value.

getStatusPanel

public *ControllerStatusPanel* **getStatusPanel**()
Returns the controller's status panel that will be displayed in the GUI. If no panel is implemented, this method should return null. In this case, the corresponding space in the MonitorGUI will appear blank.

Returns The status panel of the controller, or null.

nextValue

public double **nextValue**(double *value*)
Receives next input from the WorkerThread (this the batched output of the analyzer). This method must be able to accept Double.NaN, which the analyzer will produce if if can not produce an output for some reason.

Parameters

- **value** – The analyzer's input value.

Returns The controller's new output value for the control signal.

setSetpoint

public void **setSetpoint**(double *new_setpoint*)
Sets a new values for the controller's setpoint.

Parameters

- **new_setpoint** – The desired value for the process variable.

1.2 ch.epfl.leb.alica.interfaces.analyzers

1.2.1 AnalyzerFactory

public class **AnalyzerFactory** extends *AbstractFactory*<*AnalyzerSetupPanel*>
Analyzer factory.

Author Marcel Stefko

Constructors

AnalyzerFactory

public **AnalyzerFactory**()
Adds the known algorithms to the list.

Methods

build

public *Analyzer* **build**()
Build the selected analyzer using current settings

Returns initialized analyzer

1.2.2 AnalyzerSetupPanel

public abstract class **AnalyzerSetupPanel** extends javax.swing.JPanel
Sets up the analyzer, allows to customize settings.

Author Marcel Stefko

Methods

getName

public abstract String **getName**()

initAnalyzer

public abstract *Analyzer* **initAnalyzer**()
Construct the analyzer from current settings.

Returns initialized analyzer

1.2.3 AnalyzerSetupPanelLoader

class **AnalyzerSetupPanelLoader**

Methods

getAnalyzerSetupPanels

public static ArrayList<*AnalyzerSetupPanel*> **getAnalyzerSetupPanels**()
Dynamically loads AnalyzerSetupPanels from available jar files. The jar filename MUST begin with "**ALICA_**" for the jar to be recognized.

Returns list of loaded setup panels

1.2.4 AnalyzerStatusPanel

public abstract class **AnalyzerStatusPanel** extends javax.swing.JPanel
A panel which will display status of the analyzer. Can be used to change settings during runtime (optional).

Author Marcel Stefko

1.3 ch.epfl.leb.alica.interfaces.controllers

1.3.1 ControllerFactory

public class **ControllerFactory** extends *AbstractFactory*<*ControllerSetupPanel*>
Controller Factory

Author Marcel Stefko

Constructors

ControllerFactory

public **ControllerFactory**()
Initialize the factory with known controllers

Methods

build

public *Controller* **build**()
Build the selected controller using current settings

Returns initialized controller

setControllerTickRateMs

public void **setControllerTickRateMs**(double *tick_rate_ms*)
Set the tick rate at which the controller will operate.

Parameters

- **tick_rate_ms** – tick rate in milliseconds

setMaxControllerOutput

public void **setMaxControllerOutput**(double *max_controller_output*)
Set maximal output value of the constructed controller

Parameters

- **max_controller_output** –

1.3.2 ControllerSetupPanel

public abstract class **ControllerSetupPanel** extends javax.swing.JPanel
Makes sure that the added setup panels know how to initialize Controllers.

Author Marcel Stefko

Methods

getName

public abstract String **getName**()

initController

public abstract *Controller* **initController**(double *max_controller_output*, double *tick_rate_ms*)
Initialize and return the controller based on GUI information.

Parameters

- **max_controller_output** – maximal output value, this is passed from different part of the GUI
- **tick_rate_ms** – tick rate at which the controller will operate, passed from GUI

Returns initialized controller

1.3.3 ControllerSetupPanelLoader

class **ControllerSetupPanelLoader**

Methods

getControllerSetupPanels

public static ArrayList<*ControllerSetupPanel*> **getControllerSetupPanels**()
Dynamically loads ControllerSetupPanels from available jar files. The jar filename MUST begin with "ALICA_" for the jar to be recognized.

Returns list of loaded setup panels

1.3.4 ControllerStatusPanel

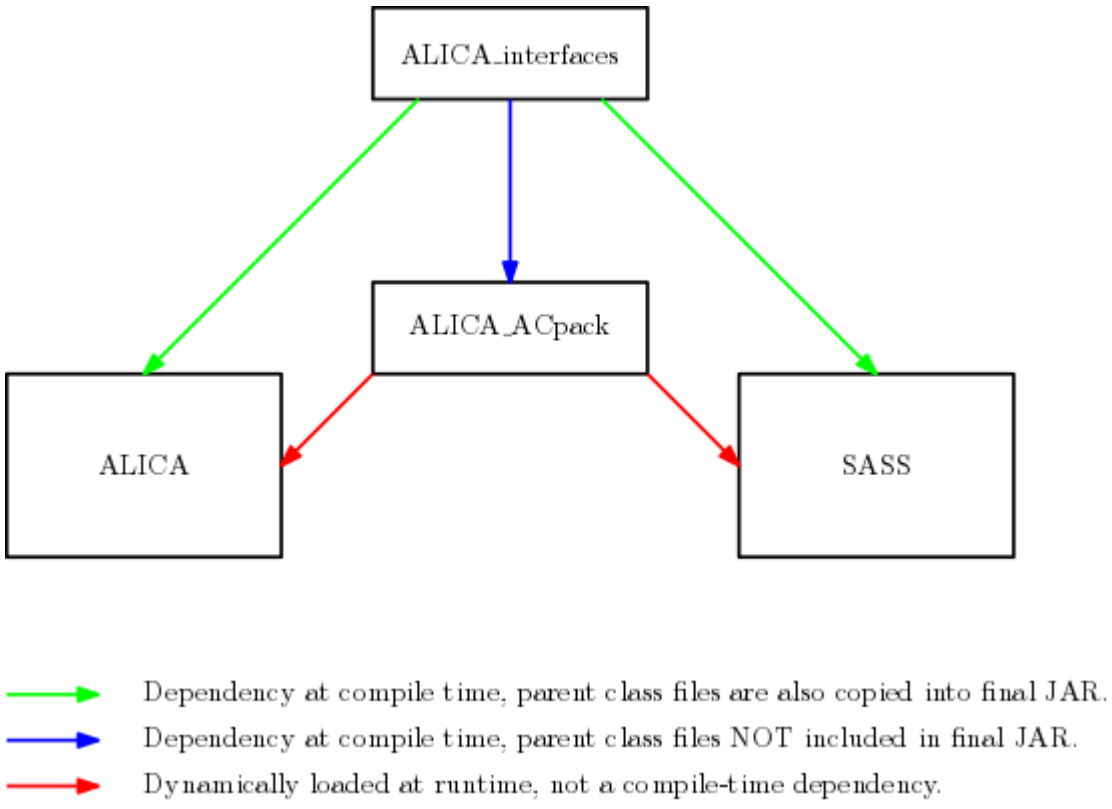
public abstract class **ControllerStatusPanel** extends javax.swing.JPanel
A panel which will display status info of the controller. Can be used to change settings during runtime.

Author Marcel Stefko

This package contains the interface definitions for the primary components of ALICA, the automatic illumination control package for light microscopy. It is a compile-time dependency of ALICA. These interfaces are maintained separately of ALICA and SASS because they are not expected to change frequently and should be quite stable.

CHAPTER 2

Relationship with ALICA and SASS



CHAPTER 3

Authors

- Marcel Štefko
- Kyle M. Douglass

CHAPTER 4

See Also

- ALICA - Automated Laser Illumination Control Algorithm
- SASS - SMLM Acquistion Simulation Software

CHAPTER 5

Indices and tables

- genindex

Index