Design document 1.0

SQUID

Helsinki 27th March 2005

Software Engineering Project

UNIVERSITY OF HELSINKI
Department of Computer Science

Course

581260 Software Engineering Project (6 cr)

Project Group

Mikko Jormalainen
Samuli Kaipiainen
Aki Korpua
Esko Luontola
Aki Sysmäläinen

Client

Lauri J. Pesonen
Fabio Donadini
Tomas Kohout

Project Masters

Juha Taina
Jenni Valorinta

Homepage

`http://www.cs.helsinki.fi/group/squid/`

Change Log

Version	Date	Modifications
0.1	4.3.2005	First version with nothing in it (Samuli Kaipiainen)
0.2	8.3.2005	Some class descriptions (Aki Korpua, Samuli Kaipiainen)
0.25	9.3.2005	Macros for class/field/method documentation (Esko Luontola)
		RunQueue (Esko Luontola)
0.3	11.3.2005	Conventions added, Class diagrams improved (Esko Luontola)
		Subsystem sections (Samuli Kaipiainen
0.4	12.3.2005	Measurement controls (Samuli Kaipiainen)
		Calibration and Project Explorer (Samuli Kaipiainen)
		Main view, menu, settings, statusbar (Aki Korpua)
		Class diagram modified (Esko Luontola)
		Class descriptions for Project data (Esko Luontola)
0.45	13.3.2005	Squid-emu, Squid-interface (Aki Korpua)
		Fixes and refinements (Samuli Kaipiainen)
		Project class (Esko Luontola)
0.5	14.3.2005	Small fixes (Esko Luontola)
		Some fixes (Aki Korpua)
		Small and bigger fixes (Samuli Kaipiainen)
		Project class finished (Esko Luontola)
...		

Version	Date	Modifications
0.55	16.3.2005	Project data finished (Esko Luontola)
		Missing event to ProjectExplorerPanel (Samuli Kaipiainen)
		SquidEmu finished (hopefully), sisalto.tex changed (Aki Korpua)
		Squid, SquidEmu and MainWindow dias added (Aki Korpua)
		Introduction (Samuli Kaipiainen)
		Overview (Aki Korpua)
0.6	17.3.2005	Update to MeasurementSequencePanel (Mikko Jormalainen)
		SerialIO, Graphs (Aki Sysmäläinen)
		Architecture description sectioned (Samuli Kaipiainen)
		Squid and SquidEmu update (Aki Korpua)
		Squid interface corrections (Aki Korpua)
0.7	18.3.2005	Fixes, ManualControlsPanel (Samuli Kaipiainen)
		Stuff to Architecture desctiption (Samuli Kaipiainen)
		PE, Calibration and MC class diagrams (Samuli Kaipiainen)
		Exporting and manual control to Project class (Esko Luontola)
		Class diagram reorganizations (Samuli Kaipiainen)
		Tables of contents for methods (Esko Luontola)
		Renamed RunQueue to LastExecutor (Esko Luontola)
0.75	19.3.2005	Final fixes to PE and MC (Samuli Kaipiainen)
		All "small" class diagrams to same size (Samuli Kaipiainen)
		Added testing section (Mikko Jormalainen)
		Configuration architecture, fixes (Settings, Squid) (Aki Korpua)
		Changes for testing and Squid Emulator (Aki Korpua)
		GUI and SerialIO class diagram updates (Samuli Kaipiainen)
		Small, random fixes (Samuli Kaipiainen)
		Last class diagrams (Mikko Jormalainen)
0.8	20.3.2005	Document frozen until FTR (Esko Luontola)
1.0	27.3.2005	Fixes to Project data (Esko Luontola)
		Fixes to many places (Mikko Jormalainen)
		Squid- and configuration-diagrams, small fixes (Aki Korpua)
		Squid-interface corrections (Aki Korpua)
		Overview-diagram, gui-diagram references (Samuli Kaipiainen)
		Figure numbering to all diagrams (Samuli Kaipiainen)

Contents

1 Introduction

This document describes planned architecture for the SQUID magnetometer program that will be implemented as a software engineering student project at University of Helsinki, Department of Computer Science. The clients are Lauri Pesonen with his assistants Fabio Donadini and Tomas Kohout from Department of Geophysics.

The document serves as an internal guide to the project team for aiding implementation phase, and describes the software at about level of accuracy which allows to implement the software based on this document and requirements document (version 1.1), which has user interface prototypes as appendices, and on which this document is based.

1.1 Structure of the document

Section 1 (this section) describes the meaning and structure of the document.

Section 2 describes coding conventions used by the project group in this sowtware.

Section 3 describes the software and architecture at high abstarction.

Section 4 describes planned architecture at lower abstarction, including class diagrams and a short description of each subsystem.

Section 5 describes planned data classes.

Section 6 describes planned gui classes.

Section 7 describes testing plans.

2 Code conventions

Everybody will follow the Code Conventions for the Java Programming Language set by Sun, with the following refinements.

- Line length will be set to 120 characters, because we prefer coding in high resolutions.
- If possible, set your IDE to use spaces instead of tabs (to avoid problems if somebody has set tab to 4 spaces, although it should be 8). Indentation is 4 spaces, as set by Sun.
- Every method and non-trivial field must have Javadoc comments. Every parameter, return value and exception of methods must be mentioned (except for trivial getters and setters).
- Every if, for and while loop must use braces `{}`, even when there will be only one statement in the block, as set by Sun.

- The @author comment for every class should have the name of the person who wrote (and designed) the class. Then we will know who to ask, if there are some questions about the code.
- Every source file is subject to automatic code reformatting by a Java IDE, in which case the reformatter must follow these code conventions.
- TODO-comments should be set by the programmer, if there is some part that needs more work. The format is `"// TODO: comments"`

The Code Conventions are available at
`http://java.sun.com/docs/codeconv/`

This program will be written with Java 1.5. Every programmer should have a look at the new features that were introduced to the Java language. Especially noteworthy are Generics, Foreach-loop and Enums. The following article will explain them in a nutshell.
`http://java.sun.com/developer/technicalArticles/releases/j2se15/`

It is recommendable for everybody to have a quick glance at Design Patterns. Here are some useful links.
`http://sern.ucalgary.ca/courses/SENG/609.04/W98/notes/`
`http://www.dofactory.com/Patterns/Patterns.aspx`

3 Overview of the system

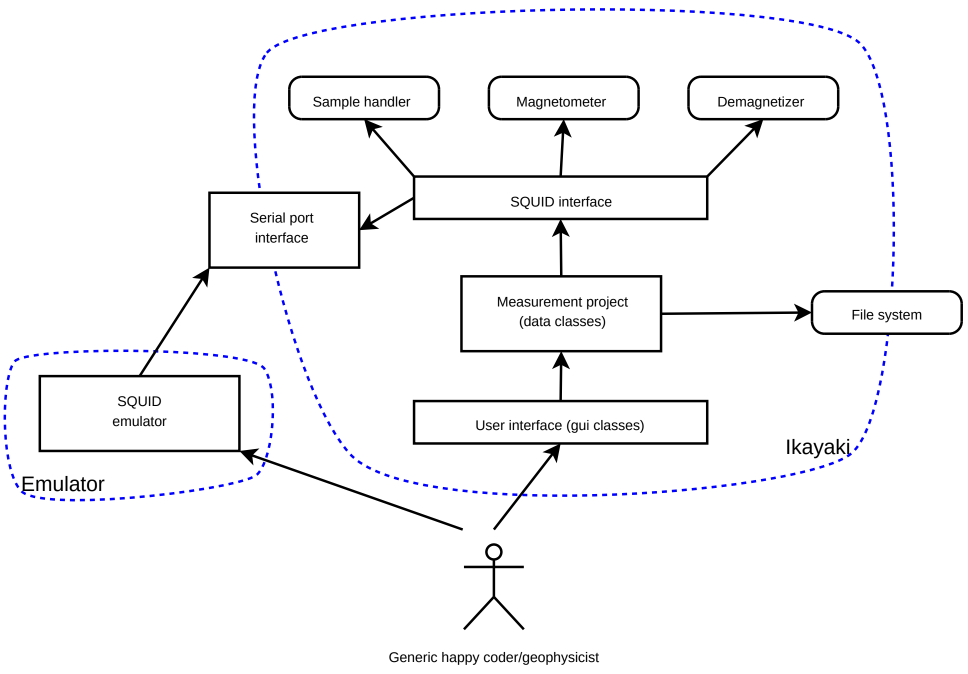


Figure 1: overview

Overview and architecture of the system is illustrated in Figure 1.

This system has two different separete project, Ikayaki-program and Squid Emulator which is subproject. Ikayaki, as main project, has graphical user interface (see requirements-document) and interface for communicating with SQUID magnetometer (Superconducting Quantum Interference Device) to measusure magnetization of minerals and rocks. Squid Emulator is vital for testing Ikayaki and it will be simple command-line program which emulates only data flow with random data and communication.

Software is split in two main parts in this document. Data section includes files needed for project-management, data flow in software and interface to control SQUID-system. There is also Settings for whole system and a subproject squid emulator. User Interface section documents all graphical interface classes, nothing more. There will be no graphical user interface for squid emulator.

4 Architecture description

Here we describe each subsystem shortly, and present a class diagram of each, as well as the whole system divided roughly into data and gui parts.

Note that the structure of this section is exactly the same as sections 5-6, but with only first two sectioning levels.

4.1 Data classes and methods

Class diagram of data classes is in Figure 2.

Data classes are in packages ikayaki, ikayaki.squid and ikayaki.util.

Package ikayaki holds generic data classes, ikayaki.squid holds classes loosely related to the squid interface, and ikayaki.util has utilities (section 4.1.6).

4.1.1 Project data

Responsible for holding all the measurement data and controlling the SQUID. Most of the GUI classes use the Project class. When the state of the project changes, the Project class fires ProjectEvents and MeasurementEvents to the GUI classes, which in turn will call the Project class to get the changed information.

The classes Project, MeasurementSequence, MeasurementStep, MeasurementResult and MeasurementValue are shown in Figure 2.

4.1.2 Squid interface

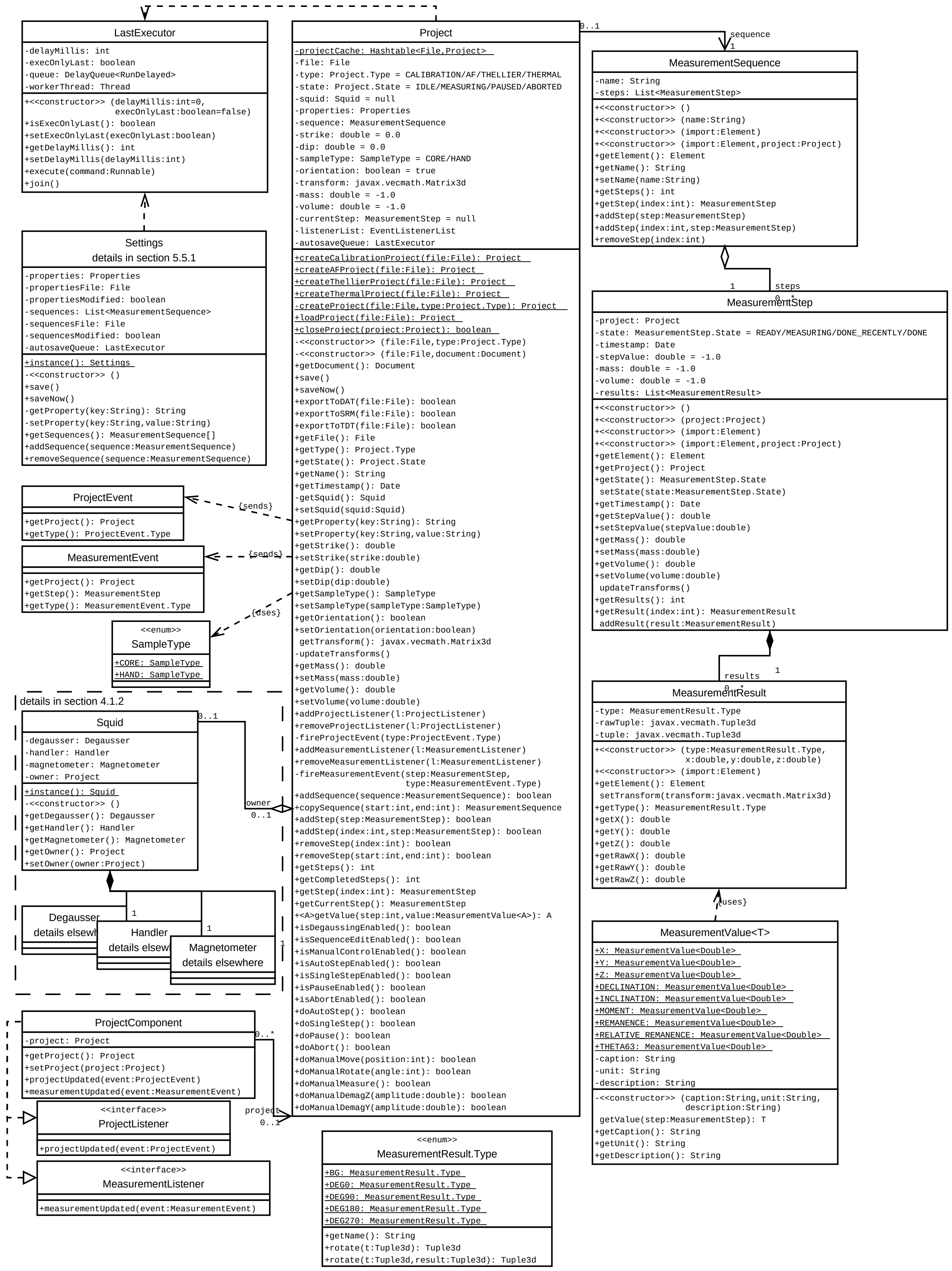

LastExecutor
-delayMillis: int
-execOnlyLast: boolean
-queue: DelayQueue<RunDelayed>
-workerThread: Thread
+<<constructor>> (delayMillis:int=0, execOnlyLast:boolean=false)
+isExecOnlyLast(): boolean
+setExecOnlyLast(execOnlyLast:boolean)
+getDelayMillis(): int
+setDelayMillis(delayMillis:int)
+execute(command:Runnable)
+join()
Settings
details in section 5.5.1
-properties: Properties
-propertiesFile: File
-propertiesModified: boolean
-sequences: List<MeasurementSequence>
-sequencesFile: File
-sequencesModified: boolean
-autosaveQueue: LastExecutor
+instance(): Settings
-<<constructor>> ()
+save()
+saveNow()
-getProperty(key:String): String
-setProperty(key:String,value:String)
+getSequences(): MeasurementSequence[]
+addSequence(sequence:MeasurementSequence)
+removeSequence(sequence:MeasurementSequence)
ProjectEvent
+getProject(): Project
+getType(): ProjectEvent.Type
{sends}
MeasurementEvent
+getProject(): Project
+getStep(): MeasurementStep
+getType(): MeasurementEvent.Type
{sends}
{uses}
<<enum>>
SampleType
+CORE: SampleType
+HAND: SampleType
details in section 4.1.2
Squid
-degausser: Degausser
-handler: Handler
-magnetometer: Magnetometer
-owner: Project
+instance(): Squid
-<<constructor>> ()
+getDegausser(): Degausser
+getHandler(): Handler
+getMagnetometer(): Magnetometer
+getOwner(): Project
+setOwner(owner:Project)
0..1
owner
0..1
1
1
1
Degausser
details elsewh
Handler
details elsew
Magnetometer
details elsewhere
ProjectComponent
-project: Project
+getProject(): Project
+setProject(project:Project)
+projectUpdated(event:ProjectEvent)
+measurementUpdated(event:MeasurementEvent)
0..*
project
0..1
<<interface>>
ProjectListener
+projectUpdated(event:ProjectEvent)
<<interface>>
MeasurementListener
+measurementUpdated(event:MeasurementEvent)
Project
-projectCache: Hashtable<File,Project>
-file: File
-type: Project.Type = CALIBRATION/AF/THELLIER/THERMAL
-state: Project.State = IDLE/MEASURING/PAUSED/ABORTED
-squid: Squid = null
-properties: Properties
-sequence: MeasurementSequence
-strike: double = 0.0
-dip: double = 0.0
-sampleType: SampleType = CORE/HAND
-orientation: boolean = true
-transform: javax.vecmath.Matrix3d
-mass: double = -1.0
-volume: double = -1.0
-currentStep: MeasurementStep = null
-listenerList: EventListenerList
-autosaveQueue: LastExecutor
+createCalibrationProject(file:File): Project
+createAFProject(file:File): Project
+createThellierProject(file:File): Project
+createThermalProject(file:File): Project
-createProject(file:File,type:Project.Type): Project
+loadProject(file:File): Project
+closeProject(project:Project): boolean
-<<constructor>> (file:File,type:Project.Type)
-<<constructor>> (file:File,document:Document)
+getDocument(): Document
+save()
+saveNow()
+exportToDAT(file:File): boolean
+exportToSRM(file:File): boolean
+exportToTDT(file:File): boolean
+getFile(): File
+getType(): Project.Type
+getState(): Project.State
+getName(): String
+getTimestamp(): Date
-getSquid(): Squid
+setSquid(squid:Squid)
+getProperty(key:String): String
+setProperty(key:String,value:String)
+getStrike(): double
+setStrike(strike:double)
+getDip(): double
+setDip(dip:double)
+getSampleType(): SampleType
+setSampleType(sampleType:SampleType)
+getOrientation(): boolean
+setOrientation(orientation:boolean)
getTransform(): javax.vecmath.Matrix3d
-updateTransforms()
+getMass(): double
+setMass(mass:double)
+getVolume(): double
+setVolume(volume:double)
+addProjectListener(l:ProjectListener)
+removeProjectListener(l:ProjectListener)
-fireProjectEvent(type:ProjectEvent.Type)
+addMeasurementListener(l:MeasurementListener)
+removeMeasurementListener(l:MeasurementListener)
-fireMeasurementEvent(step:MeasurementStep, type:MeasurementEvent.Type)
+addSequence(sequence:MeasurementSequence): boolean
+copySequence(start:int,end:int): MeasurementSequence
+addStep(step:MeasurementStep): boolean
+addStep(index:int,step:MeasurementStep): boolean
+removeStep(index:int): boolean
+removeStep(start:int,end:int): boolean
+getSteps(): int
+getCompletedSteps(): int
+getStep(index:int): MeasurementStep
+getCurrentStep(): MeasurementStep
+<A>getValue(step:int,value:MeasurementValue<A>): A
+isDegaussingEnabled(): boolean
+isSequenceEditEnabled(): boolean
+isManualControlEnabled(): boolean
+isAutoStepEnabled(): boolean
+isSingleStepEnabled(): boolean
+isPauseEnabled(): boolean
+isAbortEnabled(): boolean
+doAutoStep(): boolean
+doSingleStep(): boolean
+doPause(): boolean
+doAbort(): boolean
+doManualMove(position:int): boolean
+doManualRotate(angle:int): boolean
+doManualMeasure(): boolean
+doManualDemagZ(amplitude:double): boolean
+doManualDemagY(amplitude:double): boolean
<<enum>>
MeasurementResult.Type
+BG: MeasurementResult.Type
+DEG0: MeasurementResult.Type
+DEG90: MeasurementResult.Type
+DEG180: MeasurementResult.Type
+DEG270: MeasurementResult.Type
+getName(): String
+rotate(t:Tuple3d): Tuple3d
+rotate(t:Tuple3d,result:Tuple3d): Tuple3d
0..1
sequence
1
MeasurementSequence
-name: String
-steps: List<MeasurementStep>
+<<constructor>> ()
+<<constructor>> (name:String)
+<<constructor>> (import:Element)
+<<constructor>> (import:Element,project:Project)
+getElement(): Element
+getName(): String
+setName(name:String)
+getSteps(): int
+getStep(index:int): MeasurementStep
+addStep(step:MeasurementStep)
+addStep(index:int,step:MeasurementStep)
+removeStep(index:int)
1
steps
0..*
MeasurementStep
-project: Project
-state: MeasurementStep.State = READY/MEASURING/DONE_RECENTLY/DONE
-timestamp: Date
-stepValue: double = -1.0
-mass: double = -1.0
-volume: double = -1.0
-results: List<MeasurementResult>
+<<constructor>> ()
+<<constructor>> (project:Project)
+<<constructor>> (import:Element)
+<<constructor>> (import:Element,project:Project)
+getElement(): Element
+getProject(): Project
+getState(): MeasurementStep.State
setState(state:MeasurementStep.State)
+getTimestamp(): Date
+getStepValue(): double
+setStepValue(stepValue:double)
+getMass(): double
+setMass(mass:double)
+getVolume(): double
+setVolume(volume:double)
updateTransforms()
+getResults(): int
+getResult(index:int): MeasurementResult
addResult(result:MeasurementResult)
results
1
0..*
MeasurementResult
-type: MeasurementResult.Type
-rawTuple: javax.vecmath.Tuple3d
-tuple: javax.vecmath.Tuple3d
+<<constructor>> (type:MeasurementResult.Type, x:double,y:double,z:double)
+<<constructor>> (import:Element)
+getElement(): Element
setTransform(transform:javax.vecmath.Matrix3d)
+getType(): MeasurementResult.Type
+getX(): double
+getY(): double
+getZ(): double
+getRawX(): double
+getRawY(): double
+getRawZ(): double
{uses}
MeasurementValue<T>
+X: MeasurementValue<Double>
+Y: MeasurementValue<Double>
+Z: MeasurementValue<Double>
+DECLINATION: MeasurementValue<Double>
+INCLINATION: MeasurementValue<Double>
+MOMENT: MeasurementValue<Double>
+REMANENCE: MeasurementValue<Double>
+RELATIVE_REMANENCE: MeasurementValue<Double>
+THETA63: MeasurementValue<Double>
-caption: String
-unit: String
-description: String
-<<constructor>> (caption:String,unit:String, description:String)
getValue(step:MeasurementStep): T
+getCaption(): String
+getUnit(): String
+getDescription(): String

Figure 2: Squid class diagram: data classes

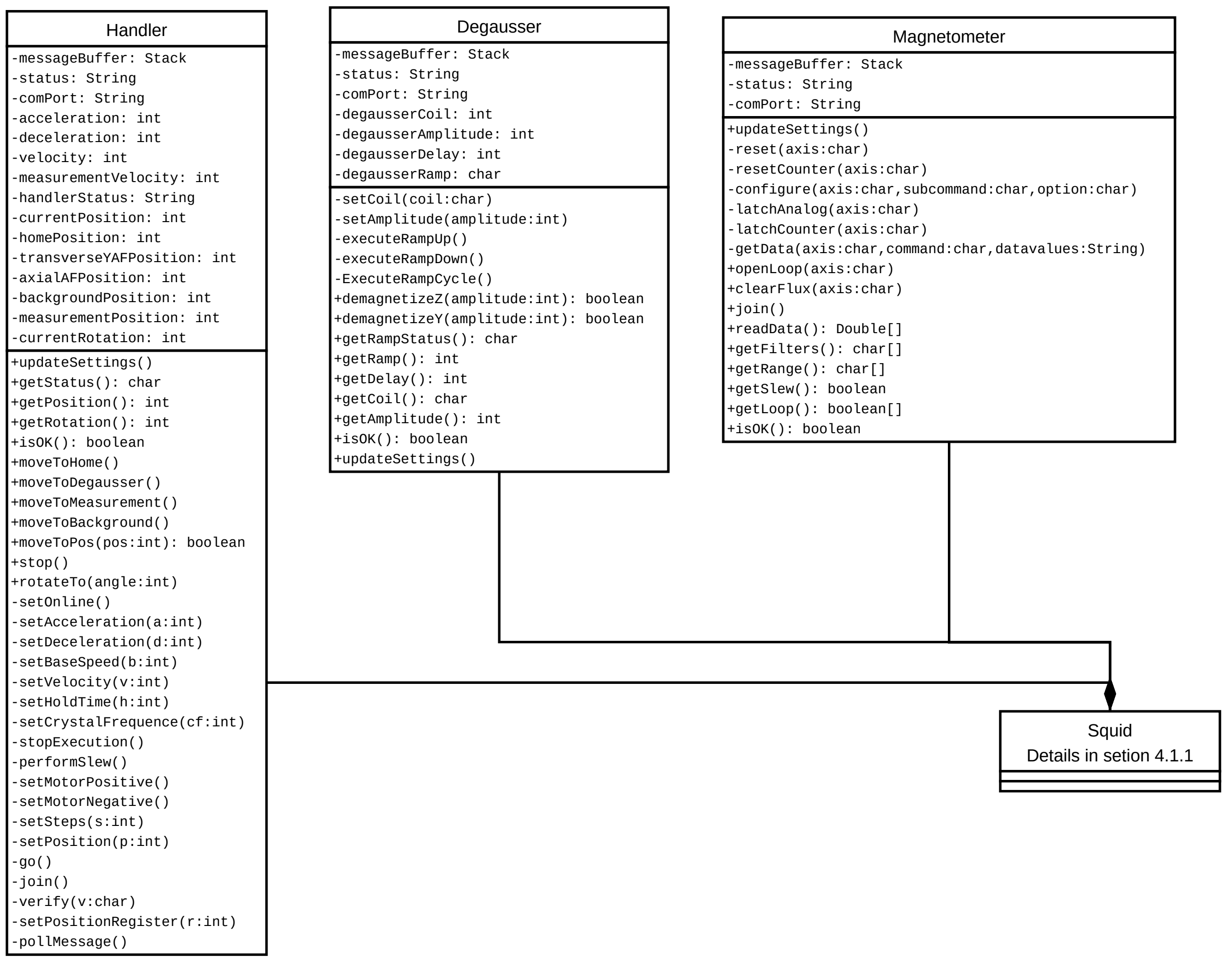


Figure 3: core-squid

Squid Interface offers the Project class an interface to safely control the SQUID magnetometer. The Squid class holds three classes that handle communication to to three separate parts of the SQUID (Handler, Degausser and Magnetometer).

Classes are Squid, Handler, Magnetometer, Degausser.

4.1.3 Squid emulator

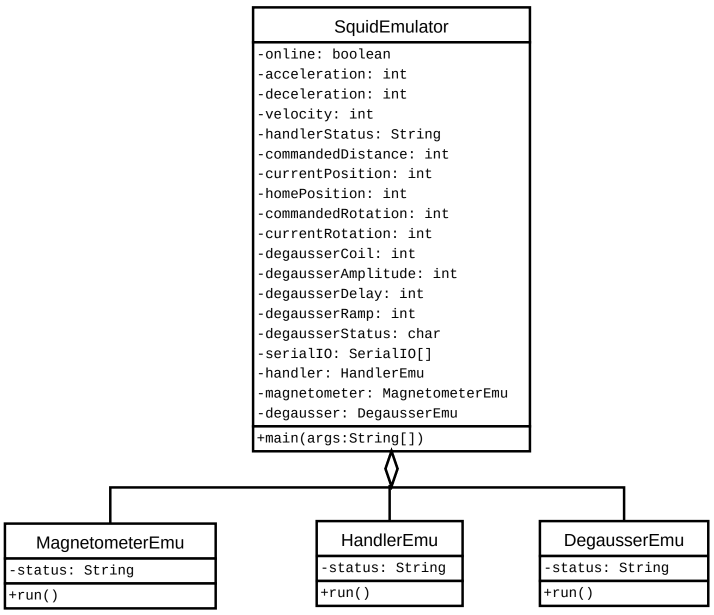


Figure 4: core-squidemu

Squid Emulator is separate from the rest of the program and it is used only for testing that the Squid Interface works correctly.

4.1.4 Serial communication

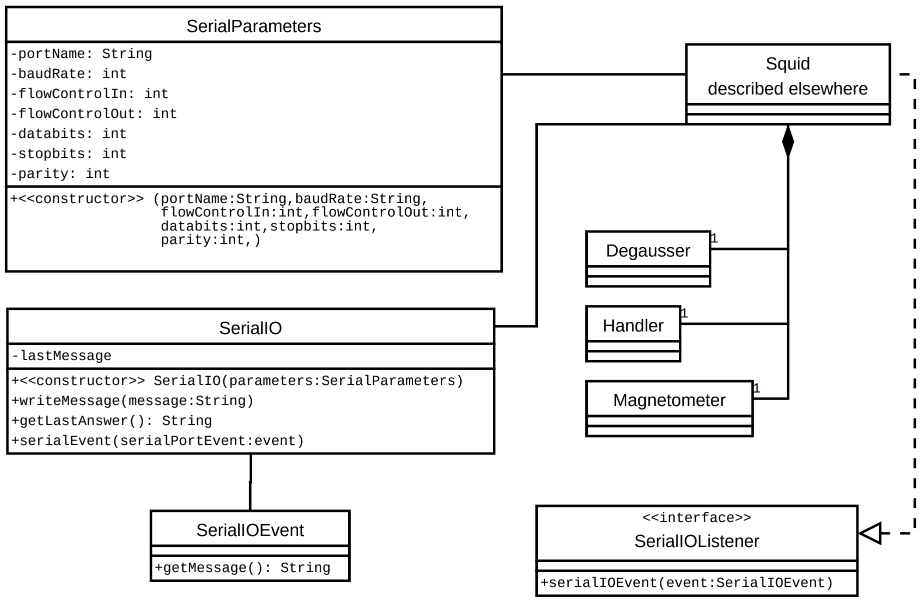


Figure 5: core-serial

SerialIO and classes related to it takes care of the harware layer of serial communication. Using these classes the program communicates with the Degausser, Samplehandler and

Magnetometer. SerialIO represents one serial port and when it's created it reserves the port to itself. SerialProperties class includes all the configuration data for the serial port.

4.1.5 Global settings

Global properties that are used all around the program. The Settings class provides a global point for retrieving and modifying the properties.

The class Settings is shown in Figure 2 and details of all the properties are in the documentation.

4.1.6 Utilities

Utility classes that are used in the program, but do not fit any of the other packages. At the moment includes only the LastExecutor class for thread management, but more classes can be added as necessary.

4.2 GUI classes and methods

Class diagram of gui classes is in Figure 6. This section is divided into sections by gui components, each of which has one or more classes.

All gui classes are in package ikayaki.gui.

4.2.1 Generic GUI components

ProjectComponent, a generic gui base class which handles registering Project- and MeasurementListeners to new projects, and which every project-dependant gui component subclasses. See Figure 6.

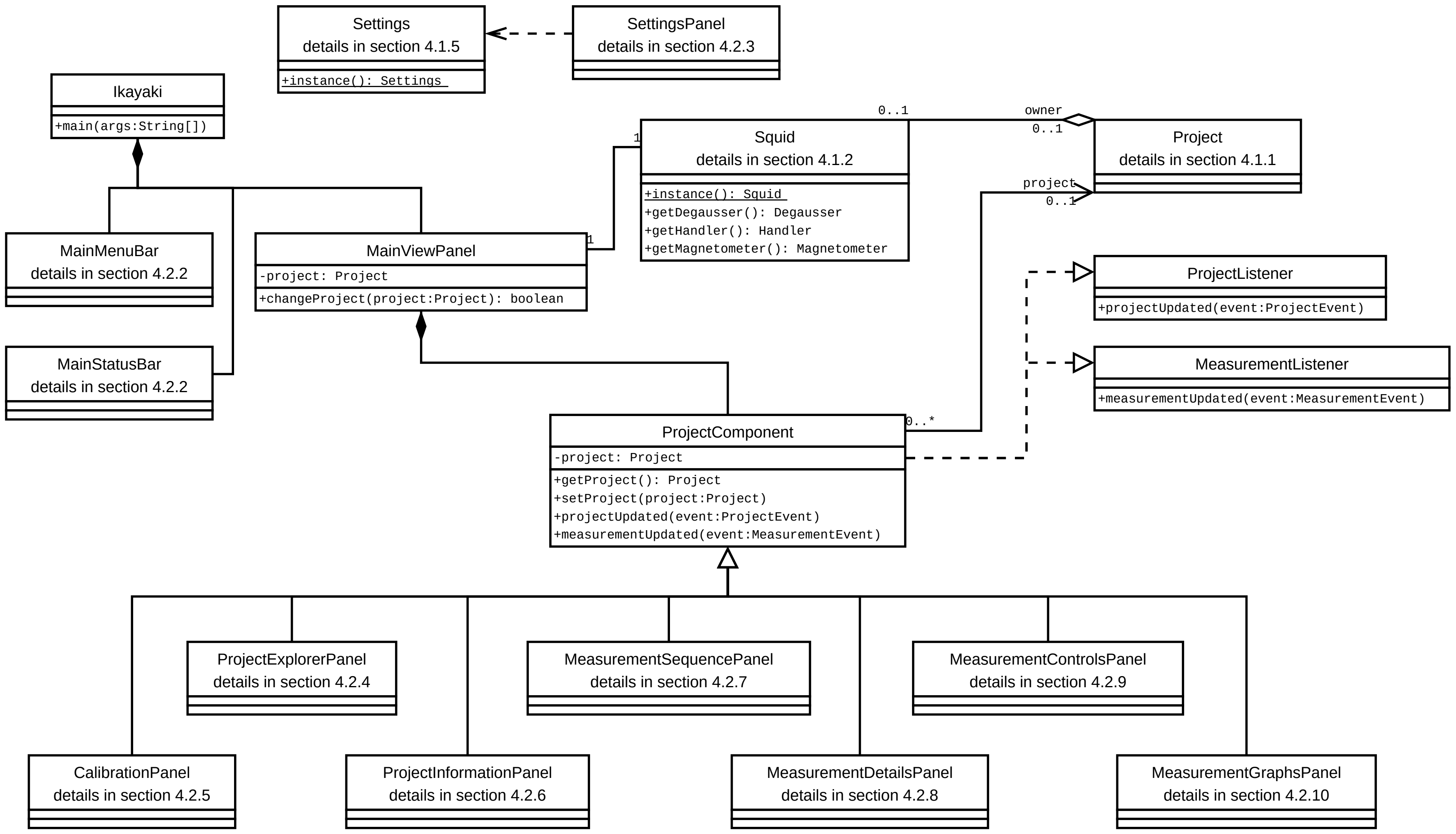


Figure 6: Squid class diagram: GUI classes

4.2.2 Main window

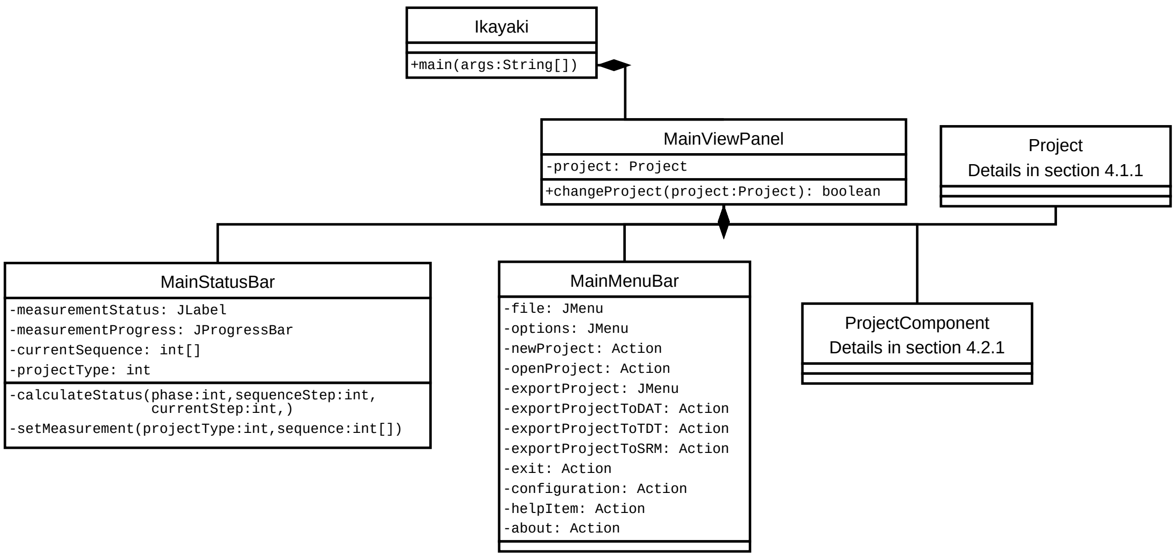
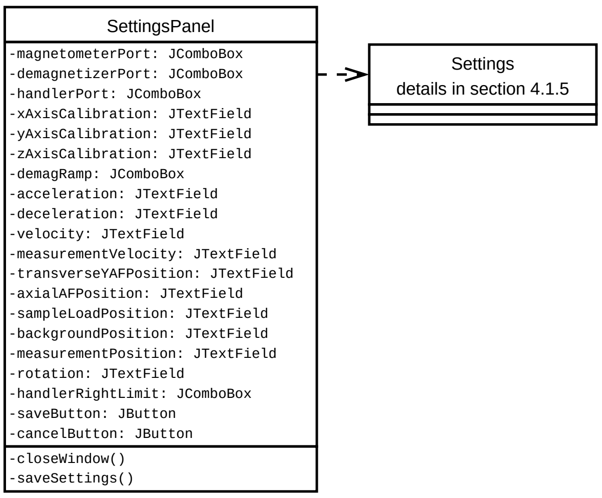


Figure 7: gui-main

Creates main view for GUI. Menubar at top, Statusbar at bottom and makes panels and splitpanels for other GUI components to middle. This also tells other GUI components if someone changes project file.

4.2.3 Configuration window



SettingsPanel
-magnetometerPort: JComboBox
-demagnetizerPort: JComboBox
-handlerPort: JComboBox
-xAxisCalibration: JTextField
-yAxisCalibration: JTextField
-zAxisCalibration: JTextField
-demagRamp: JComboBox
-acceleration: JTextField
-deceleration: JTextField
-velocity: JTextField
-measurementVelocity: JTextField
-transverseYAFPosition: JTextField
-axialAFPosition: JTextField
-sampleLoadPosition: JTextField
-backgroundPosition: JTextField
-measurementPosition: JTextField
-rotation: JTextField
-handlerRightLimit: JComboBox
-saveButton: JButton
-cancelButton: JButton
-closeWindow()
-saveSettings()
Settings
details in section 4.1.5

Figure 8: gui-configuration

Separate window which is opened from menubar and it updates settings for Squid interface. Used usually only when system is installed to setup it.

4.2.4 Project Explorer

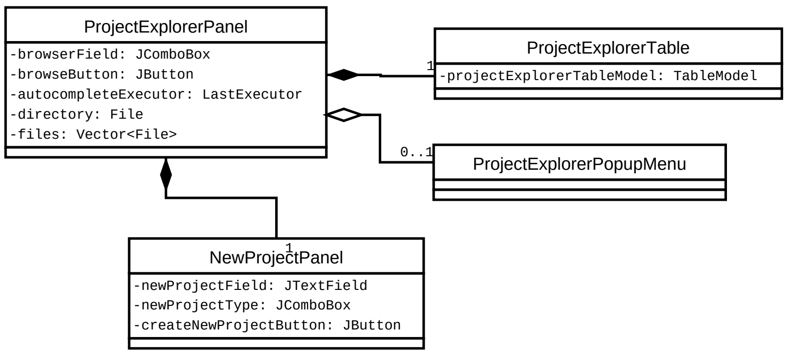


Figure 9: gui-explorer

Located at middle left side of main window.

ProjectExplorerPanel handles loading existing projects and creating new ones. Shows a listing of project files in current directory, which can be changed by typing new directory into ComboBox text field, or using the browse-button and a standard directory chooser dialog (JFileChooser). ComboBox also holds directory history, and, when typing text into its text field, automatically shows autocomplete results.

NewProjectPanel has components for creating a new project.

ProjectExplorerTable is a JTable with the project file listing, including "filename", "type" and "last modified" columns.

ProjectExplorerPopupMenu has options for exporting project files into different formats.

4.2.5 Calibration

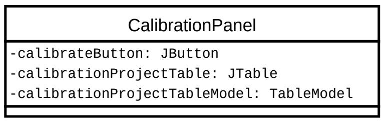

CalibrationPanel
-calibrateButton: JButton
-calibrationProjectTable: JTable
-calibrationProjectTableModel: TableModel

Figure 10: gui-calibration

Located at upper left corner of main window.

CalibrationPanel holds predefined "Holder noise" and "Standard sample" projects for calibration in a similar table as Project Explorer. Also has a "Calibrate" button, which executes selected calibration project, similarly to clicking "Single step" in normal projects.

4.2.6 Project information

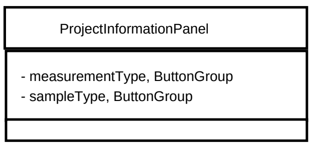


Figure 11: gui-projectinfo

Located at lower left corner of main window.

Contains and allows editing of basic information of sample. Includes such fields as volume, strike and dip, which are used to make calculations. Includes also fields whose information is only for users benefit.

4.2.7 Sequence and measurement data

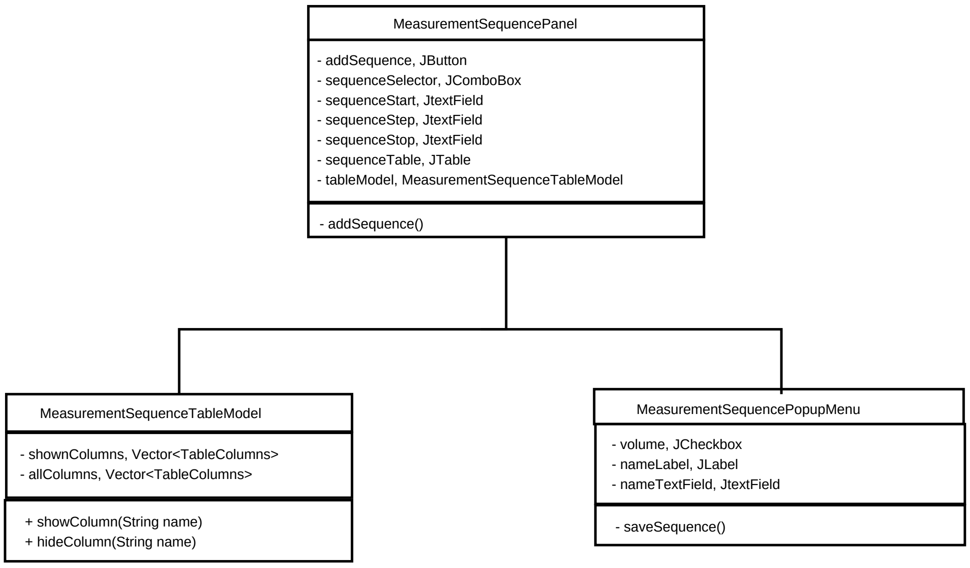

MeasurementSequencePanel
- addSequence, JButton
- sequenceSelector, JComboBox
- sequenceStart, JtextField
- sequenceStep, JtextField
- sequenceStop, JtextField
- sequenceTable, JTable
- tableModel, MeasurementSequenceTableModel
- addSequence()
MeasurementSequenceTableModel
- shownColumns, Vector<TableColumns>
- allColumns, Vector<TableColumns>
+ showColumn(String name)
+ hideColumn(String name)
MeasurementSequencePopupMenu
- volume, JCheckbox
- nameLabel, JLabel
- nameTextField, JtextField
- saveSequence()

Figure 12: gui-sequence

Located at center of main window.

Contains and allows editing of measurement sequence. Whenever measurement step is finished its data is added here. This data is also recalculated whenever some field affecting it in project infromation is changed.

4.2.8 Measurement details

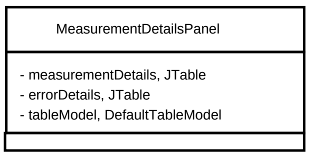


Figure 13: gui-mdetails

Located at middle bottom of main window.

Contains details of ongoing measurement or of row selected in measrurement sequence. If details of ongoing measurement is shown, they are updated whenever new measurement data is received and when measurement step is finished next steps details are shown.

4.2.9 Measurement controls

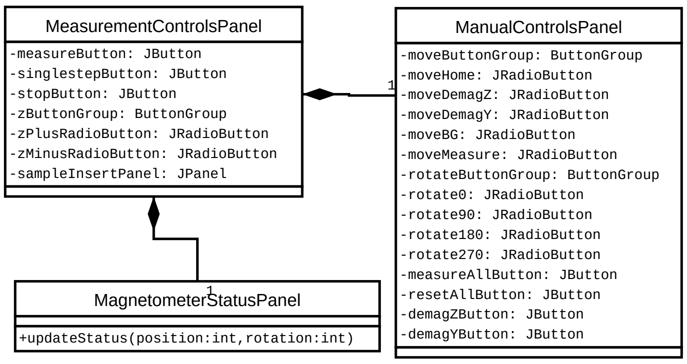

MeasurementControlsPanel
-measureButton: JButton
-singlestepButton: JButton
-stopButton: JButton
-zButtonGroup: ButtonGroup
-zPlusRadioButton: JRadioButton
-zMinusRadioButton: JRadioButton
-sampleInsertPanel: JPanel
ManualControlsPanel
-moveButtonGroup: ButtonGroup
-moveHome: JRadioButton
-moveDemagZ: JRadioButton
-moveDemagY: JRadioButton
-moveBG: JRadioButton
-moveMeasure: JRadioButton
-rotateButtonGroup: ButtonGroup
-rotate0: JRadioButton
-rotate90: JRadioButton
-rotate180: JRadioButton
-rotate270: JRadioButton
-measureAllButton: JButton
-resetAllButton: JButton
-demagZButton: JButton
-demagYButton: JButton
1
MagnetometerStatusPanel
1
+updateStatus(position:int,rotation:int)

Figure 14: gui-mcontrols

Located at upper right corner of main window.

MeasurementControlsPanel holds the buttons for controlling measurements, a help picture for sample inserting, radiobuttons for changing +z/-z orientation of sample, magnetometer status picture and manual controlling components.

MagnetometerStatusPanel shows an image of current magnetometer status, as in sample holder position and rotation. Image is updated according to MeasurementEvents received.

ManualControlsPanel has controls for fully manual measuring, which are enabled when no "normal" measurements are happening.

4.2.10 Graphs

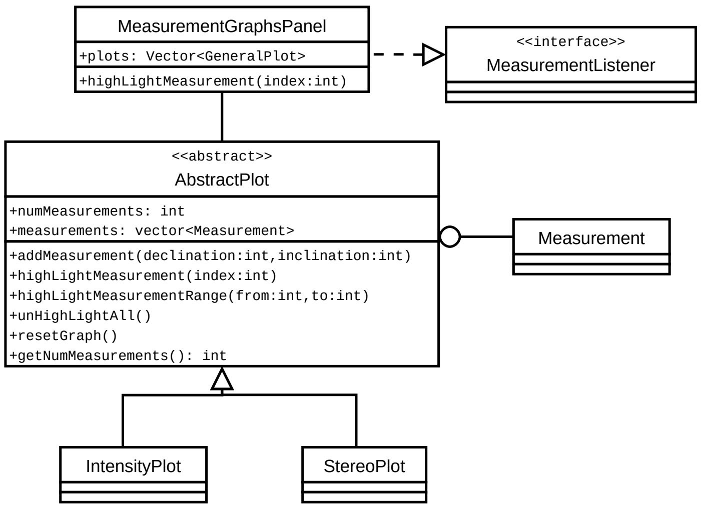

MeasurementGraphsPanel
+plots: Vector<GeneralPlot>
+highLightMeasurement(index:int)
<<interface>>
MeasurementListener
<<abstract>>
AbstractPlot
+numMeasurements: int
+measurements: vector<Measurement>
+addMeasurement(declination:int,inclination:int)
+highLightMeasurement(index:int)
+highLightMeasurementRange(from:int,to:int)
+unHighLightAll()
+resetGraph()
+getNumMeasurements(): int
Measurement
IntensityPlot
StereoPlot

Figure 15: gui-graphs

Located at lower right corner of main window.

Graph panels visualize the measurement data. MeasurementGraphsPanel listens to MeasurementEvents to update the measurement data in plots. AbstractPlot is an abstract class which implements all the general features of graph plots. IntensityPlot and StereoPlot extend the functionality of AbstractPlot and implement their special drawing features accordingly.

5 Data classes and methods

5.1 Project data

5.1.1 Project

```
/*
* Project.java
*
* Copyright (C) 2005 Project SQUID, http://www.cs.helsinki.fi/group/squid/
*
* This file is part of Ikayaki.
*
* Ikayaki is free software; you can redistribute it and/or modify
* it under the terms of the GNU General Public License as published by
* the Free Software Foundation; either version 2 of the License, or
* (at your option) any later version.
*
* Ikayaki is distributed in the hope that it will be useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
* GNU General Public License for more details.
*
* You should have received a copy of the GNU General Public License
* along with Ikayaki; if not, write to the Free Software
* Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
*/
package ikayaki;

/**
*  Represents a measurement project file. Project is responsible for managing and storing
the data that is recieved from the magnetometer measurements. Any changes made to the
project will be written to file regularly (autosave).

Project is responsible for controlling the magnetometer through the SQUID API. Controlling
the SQUID will be done in a private worker thread. Only one project at a time may access
the SQUID.

All operations are thread-safe.
*
* @author
```

```
*/
public class Project
/*
Event A: On property change - Autosaving will be invoked and the project written to file
after a short delay.
*/

/*
Event B: On measurement started/ended/paused/aborted - ProjectEvent about the state
change will be fired to all project listeners.
*/

/*
Event C: On measurement subphase started/completed - MeasurementEvent will be fired
to all measurement listeners.
*/

/*
Event D: On strike/dip/volume etc. changed - The updated transformation matrix will be
applied to all measurements and a ProjectEvent about the data change will be fired to all
project listeners.
*/

/*
Event E: On project file saved - ProjectEvent about the file saving will be fired to all
project listeners.
*/

/**
*  Caches the created and loaded Project objects to make sure that no more than one object
will be created for each physical file.
*/
private static Hashtable<File,Project> projectCache;

/**
*  Location of the project file in the local file system. Autosaving will save the project to
this file.
*/
private File file;

/**
*  Type of the measurement project.  This will affect which features of the project are
enabled and disabled.
*/
```

```
private Type type;

/**
*  Current state of the measurements.  If no measurement is running, then state is IDLE.
Only one measurement may be running at a time.
*/
private State state = IDLE;

/**
*  Pointer to the SQUID device interface, or null if this project is not its owner.
*/
private Squid squid = null;

/**
*  Custom properties of this project stored in a map. The project is not interested in what
properties are stored; it only saves them.
*/
private Properties properties;

/**
*  Measurement sequence of this project. In the beginning are all completed measurement
steps, and in the end are planned measurement steps. Completed measurements may NOT
be deleted.
*/
private MeasurementSequence sequence;

/**
*  Strike of the sample. Will be used to create the transform matrix.
*/
private double strike = 0.0;

/**
*  Dip of the sample. Will be used to create the transform matrix.
*/
private double dip = 0.0;

/**
*  Type of the sample. Will be used to create the transform matrix.
*/
private SampleType sampleType = CORE;

/**
```

```
*  Orientation of the sample. true if the sample orientation is +Z, or false if it is -Z. Will
be used to create the transform matrix.
*/
private boolean orientation = true;

/**
*  Matrix for correcting the sample's orientation. The matrix will be updated whenever
the strike, dip, sampleType or orientation is changed. After that the updated matrix will
be applied to all measurements.
*/
private Matrix3d transform = new Matrix3d().setIdentity();

/**
*  Mass of the sample, or a negative value if no mass is defined.
*/
private double mass = -1.0;

/**
*  Volume of the sample, or a negative value if no volume is defined.
*/
private double volume = -1.0;

/**
*  Current measurement step, or null if no measurement is running.
*/
private MeasurementStep currentStep = null;

/**
*  Listeners for this project.
*/
private EventListenerList listenerList = new EventListenerList();

/**
*  Scheduler for automatically writing the modified project to file after a short delay.
*/
private LastExecutor autosaveQueue = new LastExecutor(500, true);

/**
*  Creates a calibration project file.
* @param file path for the new project file.
* @return the created project, or null if file was not writable.
* @throws NullPointerException if file is null.
```

```
*/
public static Project createCalibrationProject(File file){
return null; // TODO
}

/**
*  Creates an AF project file.
* @param file path for the new project file.
* @return the created project, or null if file was not writable.
* @throws NullPointerException if file is null.
*/
public static Project createAFProject(File file){
return null; // TODO
}

/**
*  Creates a thellier project file.
* @param file path for the new project file.
* @return the created project, or null if file was not writable.
* @throws NullPointerException if file is null.
*/
public static Project createThellierProject(File file){
return null; // TODO
}

/**
*  Creates a thermal project file.
* @param file path for the new project file.
* @return the created project, or null if file was not writable.
* @throws NullPointerException if file is null.
*/
public static Project createThermalProject(File file){
return null; // TODO
}

/**
*  Creates a project file of the specified type. Ensures that the project file has been written
to disk. Adds the created Project object to projectCache.
* @param file path for the new project file.
* @param type type of the project.
* @return the created project, or null if file was not writable.
* @throws NullPointerException if file is null.
*/
```

```
private static Project createProject(File file, Type type){
return null; // TODO
}

/**
*  Loads a saved project file. If the file has already been loaded, will return a reference to the existing Project object.
* @param file project file to be loaded.
* @return the loaded project, or null if file is not a valid project file or it was not readable.
* @throws NullPointerException if file is null.
*/
public static Project loadProject(File file){
return null; // TODO
}

/**
*  Ensures that the project file is saved and frees the resources taken by the project.  A project should not be used after it has been closed – any further use of the object is undefined (probably will create NullPointerExceptions).  The closed project is removed from the projectCache. A project can not be closed if it has a measurement running.
* @param project project to be closed.
* @return true if the project has been closed, false if a measurement is running and the project can not be closed.
* @throws NullPointerException if the project is null.
*/
public static boolean closeProject(Project project){
return null; // TODO
}

/**
*  Creates a new project of the specified type.  This constructor will not write to file, so the user of this method should call the saveNow() method after the project is initialized.
* @param file path for this project file.  The file should exist (may be empty) and be writable, but this constructor will not check it.
* @param type type of the project.
* @return the created project.
* @throws NullPointerException if any of the parameters is null.
*/
private Project(File file, Type type){
return null; // TODO
}

/**
```

```
*  Creates a new project from the specified document.  This constructor will assume that
the specified file is the same from which the document was read.
* @param file path for this project file.  The file should be the same from which document
was read and be writable, but this constructor will not check it.
* @param document the document from which this project will be created.
* @return the created project.
* @throws NullPointerException if any of the parameters is null.
* @throws IllegalArgumentException if the document was not in the right format.
*/
private Project(File file, Document document){
return null; // TODO
}

/**
*  Exports this project to a DOM document.
*/
public synchronized Document getDocument(){
return null; // TODO
}

/**
*  Invokes autosaving.  This method will schedule a saving operation and return.  After
this method has not been called for a short while, the project will be written to file.
*/
public synchronized void save(){
return null; // TODO
}

/**
*  Writes this project to its project file and waits for the operation to complete.
(NOTE: Synchronizing is done inside the method)
* @throws IOException if there was an error when writing to file.
*/
public void saveNow(){
return null; // TODO
}

/**
*  Writes the project to a file in DAT format.
* @param file the file to save to.
* @return true if the file was successfully written, otherwise false.
* @throws NullPointerException if file is null.
*/
```

```
public boolean exportToDAT(File file){
return null; // TODO
}

/**
*  Writes the project to a file in SRM format.
* @param file the file to save to.
* @return true if the file was successfully written, otherwise false.
* @throws NullPointerException if file is null.
*/
public boolean exportToSRM(File file){
return null; // TODO
}

/**
*  Writes the project to a file in TDT format.
* @param file the file to save to.
* @return true if the file was successfully written, otherwise false.
* @throws NullPointerException if file is null.
*/
public boolean exportToTDT(File file){
return null; // TODO
}

/**
*  Returns the project file of this project.
*/
public synchronized File getFile(){
return null; // TODO
}

/**
*  Returns the type of this project.
*/
public synchronized Type getType(){
return null; // TODO
}

/**
*  Returns the current measurement state of this project.
*/
public synchronized State getState(){
return null; // TODO
}

/**
*  Returns the name of this project.  The name is equal to the name of the project file
without the file extension.
*/
public synchronized String getName(){
return null; // TODO
}

/**
*  Returns the timestamp of the last completed measurement.  This is usually less than
the last modified date of the file, because this is not affected by changing the project's
properties.
*/
public synchronized Date getTimestamp(){
return null; // TODO
}

/**
*  Returns the Squid if this project is its owner, otherwise returns null.
(NOTE: Make this method public? Or return a Proxy (see design patterns), so others can
know where the handler is moving but not control it?)
*/
private synchronized Squid getSquid(){
return null; // TODO
}

/**
*  Sets this project the owner of the Squid.  Uses the setOwner() method of the specified
Squid.
Only one project may own the Squid at a time.  The Squid must be first detached with
"setSquid(null)" from its owner before it can be given to another project.  Detaching the
Squid is possible only when the project's state is IDLE.
* @param squid pointer to the SQUID interface, or null to detach this project from it.
* @return true if the operation was completed, false if the Squid has another owner or a
measurement is running (in which case nothing was changed).
*/
public synchronized boolean setSquid(Squid squid){
return null; // TODO
}
```

```
/**
*  Returns a project information property.
* @param key the key which is associated with the property.
* @return the specified property, or an empty String if the property is not set.
*/
public synchronized String getProperty(String key){
return null; // TODO
}

/**
*  Sets a project information property.
* @param key the key which is associated with the property.
* @param value new value for the property, or null to remove the property.
*/
public synchronized void setProperty(String key, String value){
return null; // TODO
}

/**
*  Returns the strike of the sample.
*/
public synchronized double getStrike(){
return null; // TODO
}

/**
*  Sets the strike of the sample and calls updateTransforms().
*/
public synchronized void setStrike(double strike){
return null; // TODO
}

/**
*  Returns the dip of the sample.
*/
public synchronized double getDip(){
return null; // TODO
}

/**
*  Sets the dip of the sample and calls updateTransforms().
*/
public synchronized void setDip(double dip){
```

```
return null; // TODO
}

/**
*  Returns the type of the sample.
*/
public synchronized SampleType getSampleType(){
return null; // TODO
}

/**
*  Sets the type of the sample and calls updateTransforms().
* @throws NullPointerException if sampleType is null.
*/
public synchronized void setSampleType(SampleType sampleType){
return null; // TODO
}

/**
*  Returns the orientation of the sample.
* @return true if the sample orientation is +Z, or false if it is -Z.
*/
public synchronized boolean getOrientation(){
return null; // TODO
}

/**
*  Sets the orientation of the sample and calls updateTransforms().
* @param orientation true if the sample orientation is +Z, or false if it is -Z.
*/
public synchronized void setOrientation(boolean orientation){
return null; // TODO
}

/**
*  Returns the current transformation matrix for the sample.  For performance reasons,
this method returns a reference to the internal data structure and not a copy of it.

WARNING!!! Absolutely NO modification of the data contained in this matrix should be
made – if any such manipulation is necessary, it should be done on a copy of the matrix
returned rather than the matrix itself.
* @return reference to the transformation matrix.
*/
synchronized Matrix3d getTransform(){
```

```
return null; // TODO
}

/**
*  Recalculates the transformation matrix and updates all measurements.  This method is
called automatically by the setStrike(), setDip() and setSampleType() methods.
*/
private synchronized void updateTransforms(){
return null; // TODO
}

/**
*  Returns the mass of the sample.
* @return mass of the sample, or a negative number if no mass is specified.
*/
public synchronized double getMass(){
return null; // TODO
}

/**
*  Sets the mass of the sample.
* @param mass mass of the sample, or a negative number to clear it.
*/
public synchronized void setMass(double mass){
return null; // TODO
}

/**
*  Returns the volume of the sample.
* @return volume of the sample, or a negative number if no volume is specified.
*/
public synchronized double getVolume(){
return null; // TODO
}

/**
*  Sets the volume of the sample.
* @param volume volume of the sample, or a negative number to clear it.
*/
public synchronized void setVolume(double volume){
return null; // TODO
}
```

```
/**
*  Adds a ProjectListener to the project.
* @param l the listener to be added.
*/
public synchronized void addProjectListener(ProjectListener l){
return null; // TODO
}

/**
*  Removes a ProjectListener from the project.
* @param l the listener to be removed
*/
public synchronized void removeProjectListener(ProjectListener l){
return null; // TODO
}

/**
*  Notifies all listeners that have registered for ProjectEvents.
* @param type type of the event.
*/
private synchronized void fireProjectEvent(ProjectEvent.Type type){
return null; // TODO
}

/**
*  Adds a MeasurementListener to the project.
* @param l the listener to be added.
*/
public synchronized void addMeasurementListener(MeasurementListener l){
return null; // TODO
}

/**
*  Removes a MeasurementListener from the project.
* @param l the listener to be removed
*/
public synchronized void removeMeasurementListener(MeasurementListener l){
return null; // TODO
}

/**
*  Notifies all listeners that have registered for MeasurementEvents.
* @param step the measurement step that has generated the event.
* @param type the type of the event.
*/
private synchronized void fireMeasurementEvent(MeasurementStep step, MeasurementEvent.Type
type){
return null; // TODO
}

/**
*  Appends a sequence to this project’s sequence.  Only the stepValues will be copied
from the specified sequence and added as new steps to this project.
If isSequenceEditEnabled() is false, nothing will be done.
* @param sequence the measurement sequence to be added.
* @return true if the steps were added, or false if isSequenceEditEnabled() was false.
* @throws NullPointerException if sequence is null.
*/
public synchronized boolean addSequence(MeasurementSequence sequence){
return null; // TODO
}

/**
*  Returns a copy of this project’s sequence. Only the stepValues will be copied from this
project’s sequence. The returned sequence will have no name.
* @param start index of the first step in the sequence.
* @param end index of the last step in the sequence. If end < start, then an empty sequence
will be returned.
* @return copy of the sequence with only stepValues and no results.
* @throws IndexOutOfBoundsException if the index is out of range (start < 0 || end >=
getSteps()).
*/
public synchronized MeasurementSequence copySequence(int start, int end){
return null; // TODO
}

/**
*  Appends a step to this project’s sequence. Only the stepValue will be copied from the
specified step and added as a new step to this project.
If isSequenceEditEnabled() is false, nothing will be done.
* @param step the measurement step to be added.
* @return true if the step was added, or false if isSequenceEditEnabled() was false.
* @throws NullPointerException if step is null.
*/
public synchronized boolean addStep(MeasurementStep step){
return null; // TODO
```

```
}

/**
*  Adds a step to the specified index of this project's sequence.  Only the stepValue will
be copied from the specified step and added as a new step to this project.

The index must be such, that the indices of the completed measurements will not change.

If isSequenceEditEnabled() is false, nothing will be done.
* @param index the index to which the step will be added.
* @param step the measurement step to be added.
* @return true if the step was added, or false if isSequenceEditEnabled() was false.
* @throws IndexOutOfBoundsException if the index is out of range (index < getCompletedSteps()
|| index > getSteps()).
* @throws NullPointerException if step is null.
*/
public synchronized boolean addStep(int index, MeasurementStep step){
return null; // TODO
}

/**
*  Removes a step from this project's sequence.  Completed measurements can not be
removed.

If isSequenceEditEnabled() is false, nothing will be done.
* @param index the index of the step to be removed.
* @return true if the step was removed, or false if isSequenceEditEnabled() was false.
* @throws IndexOutOfBoundsException if the index is out of range (index < getCompletedSteps()
|| index >= getSteps()).
*/
public synchronized boolean removeStep(int index){
return null; // TODO
}

/**
*  Removes a series of steps from this project's sequence. Completed measurements can
not be removed.

If isSequenceEditEnabled() is false, nothing will be done.
* @param start the first index to be removed.
* @param end the last index to be removed. If end < start, no steps will be removed.
* @return true if the steps were removed, or false if isSequenceEditEnabled() was false.
* @throws IndexOutOfBoundsException if the index is out of range (start < getCompletedSteps()
|| end >= getSteps()).
*/
public synchronized boolean removeStep(int start, int end){
```

```
return null; // TODO
}

/**
*  Returns the number of steps in this project.
*/
public synchronized int getSteps(){
return null; // TODO
}

/**
*  Returns the number of completed steps in this project.  Steps that are currently being
measured, are included in this count. Completed steps are always first in the sequence.
*/
public synchronized int getCompletedSteps(){
return null; // TODO
}

/**
*  Returns a step from the sequence.
* @param index the index of the step.
* @return the specified step.
* @throws IndexOutOfBoundsException if the index is out of range (index < 0 || index
>= getSteps()).
*/
public synchronized MeasurementStep getStep(int index){
return null; // TODO
}

/**
*  Returns the step that is currently being measured.
* @return the currently measured step, or null if no measurement is active.
*/
public synchronized MeasurementStep getCurrentStep(){
return null; // TODO
}

/**
*  Calculates and returns a value from a measurement step. The specified MeasurementValue’s
algorithm will be used and the results returned.
* @param step the measurement step from which the value is calculated.
* @param algorithm the algorithm for calculating the desired value.
* @return the value returned by the algorithm, or null if it was not possible to calculate it.
* @throws NullPointerException if algorithm is null.
*/
public synchronized <A> A getValue(int step, MeasurementValue<A> algorithm){
return null; // TODO
}

/**
* Tells whether it is allowed to use the degausser in this project. The returned value depends on the type and state of this project.
*/
public synchronized boolean isDegaussingEnabled(){
return null; // TODO
}

/**
* Tells whether it is allowed to edit the sequence. The returned value depends on the type and state of this project.
*/
public synchronized boolean isSequenceEditEnabled(){
return null; // TODO
}

/**
* Tells whether it is allowed to control the Squid manually. The returned value depends on the type and state of this project.
*/
public synchronized boolean isManualControlEnabled(){
return null; // TODO
}

/**
* Tells whether it is allowed to do an auto step measurement. The returned value depends on the type and state of this project.
*/
public synchronized boolean isAutoStepEnabled(){
return null; // TODO
}

/**
* Tells whether it is allowed to do a single step measurement. The returned value depends on the type and state of this project.
*/
public synchronized boolean isSingleStepEnabled(){
```

```
return null; // TODO
}

/**
*  Tells whether it is possible to pause the measurement.  The returned value depends on
the type and state of this project.
*/
public synchronized boolean isPauseEnabled(){
return null; // TODO
}

/**
*  Tells whether it is possible to abort the measurement.  The returned value depends on
the type and state of this project.
*/
public synchronized boolean isAbortEnabled(){
return null; // TODO
}

/**
*  Starts an auto step measurement. Will do nothing if isAutoStepEnabled() is false.

The measurement will run in its own thread, and this method will not wait for it to finish.
* @return true if the measurement was started, otherwise false.
*/
public synchronized boolean doAutoStep(){
return null; // TODO
}

/**
*  Starts a single step measurement. Will do nothing if isSingleStepEnabled() is false.

The measurement will run in its own thread, and this method will not wait for it to finish.
* @return true if the measurement was started, otherwise false.
*/
public synchronized boolean doSingleStep(){
return null; // TODO
}

/**
*  Pauses the currently running measurement.  A paused measurement will halt after it
finishes the current measurement step. Will do nothing if isPauseEnabled() is false.

This method will notify the measurement thread to pause, but will not wait for it to finish.
* @return true if the measurement will pause, otherwise false.
```

```
*/
public synchronized boolean doPause(){
return null; // TODO
}

/**
*  Aborts the currently running measurement.  An aborted measurement will halt immediately
and leave the handler where it was (enables manual control).  Will do nothing if isAbortEnabled()
is false.

This method will notify the measurement thread to abort, but will not wait for it to finish.
* @return true if the measurement will abort, otherwise false.
*/
public synchronized boolean doAbort(){
return null; // TODO
}

/**
*  Moves the sample handler to the specified position.  Will do nothing if isManualControlEnabled()
is false.

The operation will run in its own thread, and this method will not wait for it to finish.
* @param position the position to move the handler to.
* @return true if the operation was started, otherwise false.
*/
public synchronized boolean doManualMove(int position){
return null; // TODO
}

/**
*  Rotates the sample handler to the specified angle.  Will do nothing if isManualControlEnabled()
is false.

The operation will run in its own thread, and this method will not wait for it to finish.
* @param angle the angle to rotate the handler to.
* @return true if the operation was started, otherwise false.
*/
public synchronized boolean doManualRotate(int angle){
return null; // TODO
}

/**
*  Measures the X, Y and Z of the sample. Adds the results as a new measurement step to
the project. Will do nothing if isManualControlEnabled() is false.

The operation will run in its own thread, and this method will not wait for it to finish.
```

```
* @return true if the operation was started, otherwise false.
*/
public synchronized boolean doManualMeasure(){
return null; // TODO
}

/**
*  Demagnetizes the sample in Z direction with the specified amplitude. Will do nothing
if isManualControlEnabled() is false.
The operation will run in its own thread, and this method will not wait for it to finish.
* @param amplitude the amplitude to demagnetize in mT.
* @return true if the operation was started, otherwise false.
*/
public synchronized boolean doManualDemagZ(double amplitude){
return null; // TODO
}

/**
*  Demagnetizes the sample in Y direction with the specified amplitude. Will do nothing
if isManualControlEnabled() is false.
The operation will run in its own thread, and this method will not wait for it to finish.
* @param amplitude the amplitude to demagnetize in mT.
* @return true if the operation was started, otherwise false.
*/
public synchronized boolean doManualDemagY(double amplitude){
return null; // TODO
}
```

5.1.2 Project.Type

```
/*
* Project.Type.java
*
* Copyright (C) 2005 Project SQUID, http://www.cs.helsinki.fi/group/squid/
*
* This file is part of Ikayaki.
*
* Ikayaki is free software; you can redistribute it and/or modify
* it under the terms of the GNU General Public License as published by
* the Free Software Foundation; either version 2 of the License, or
* (at your option) any later version.
*
```

```
* Ikayaki is distributed in the hope that it will be useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
* GNU General Public License for more details.
*
* You should have received a copy of the GNU General Public License
* along with Ikayaki; if not, write to the Free Software
* Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
*/
package ikayaki;

/**
*  The type of the project. Options are CALIBRATION, AF, THELLIER and THERMAL.
*
* @author
*/
public enum Type
```

5.1.3 Project.State

```
/*
* Project.State.java
*
* Copyright (C) 2005 Project SQUID, http://www.cs.helsinki.fi/group/squid/
*
* This file is part of Ikayaki.
*
* Ikayaki is free software; you can redistribute it and/or modify
* it under the terms of the GNU General Public License as published by
* the Free Software Foundation; either version 2 of the License, or
* (at your option) any later version.
*
* Ikayaki is distributed in the hope that it will be useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
* GNU General Public License for more details.
*
* You should have received a copy of the GNU General Public License
* along with Ikayaki; if not, write to the Free Software
* Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
*/
package ikayaki;

/**
```

```
*  The state of the project's measurements. Options are IDLE, MEASURING, PAUSED,
ABORTED.
*
* @author
*/
public enum State
```

5.1.4 MeasurementSequence

```
/*
* MeasurementSequence.java
*
* Copyright (C) 2005 Project SQUID, http://www.cs.helsinki.fi/group/squid/
*
* This file is part of Ikayaki.
*
* Ikayaki is free software; you can redistribute it and/or modify
* it under the terms of the GNU General Public License as published by
* the Free Software Foundation; either version 2 of the License, or
* (at your option) any later version.
*
* Ikayaki is distributed in the hope that it will be useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
* GNU General Public License for more details.
*
* You should have received a copy of the GNU General Public License
* along with Ikayaki; if not, write to the Free Software
* Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
*/
package ikayaki;

/**
*  A list of measurement steps. Steps can be added or removed from the sequence.
All operations are thread-safe.
*
* @author
*/
public class MeasurementSequence
/**
*  Name of the sequence or null if it has no name.
*/
private String name = null;
```

```
/**
*  The measurement steps of this sequence.
*/
private List<MeasurementStep> steps = new ArrayList<MeasurementStep>();

/**
*  Creates an empty sequence with no name.
* @return the created sequence.
*/
public MeasurementSequence(){
return null; // TODO
}

/**
*  Creates an empty sequence with the specified name.
* @param name name of the sequence.
* @return the created sequence.
*/
public MeasurementSequence(String name){
return null; // TODO
}

/**
*  Creates a sequence from the specified element.
* @param import the element from which this sequence will be created.
* @return the created sequence.
* @throws NullPointerException if import is null.
* @throws IllegalArgumentException if the element was not in the right format.
*/
public MeasurementSequence(Element import){
return null; // TODO
}

/**
*  Creates a sequence from the specified element for a project.
* @param import the element from which this sequence will be created.
* @param project the project whose sequence this will be.  Needed for importing the
measurement steps correctly.
* @return the created sequence.
* @throws NullPointerException if import is null.
* @throws IllegalArgumentException if the element was not in the right format.
*/
public MeasurementSequence(Element import, Project project){
```

```
return null; // TODO
}

/**
*  Exports this sequence to a DOM element.
*/
public synchronized Element getElement(){
return null; // TODO
}

/**
*  Returns the name of this sequence.
* @return the name, or null if it has no name
*/
public synchronized String getName(){
return null; // TODO
}

/**
*  Sets the name of this sequence.
*/
public synchronized void setName(String name){
return null; // TODO
}

/**
*  Returns the number of steps in this sequence.
*/
public synchronized int getSteps(){
return null; // TODO
}

/**
*  Returns the specified step from this sequence.
* @param index the index of the step.
* @return the specified step.
* @throws IndexOutOfBoundsException if the index is out of range (index < 0 || index
>= getSteps()).
*/
public synchronized MeasurementStep getStep(int index){
return null; // TODO
}
```

```
/**
*  Appends a step to this sequence.
* @param step the measurement step to be added.
* @throws NullPointerException if step is null.
*/
public synchronized void addStep(MeasurementStep step){
return null; // TODO
}

/**
*  Adds a step to the specified index of this sequence.
* @param index the index to which the step will be added.
* @param step the measurement step to be added.
* @throws IndexOutOfBoundsException if the index is out of range (index < 0 || index >
getSteps()).
* @throws NullPointerException if step is null.
*/
public synchronized void addStep(int index, MeasurementStep step){
return null; // TODO
}

/**
*  Removes a step from this sequence.
* @param index the index of the step to be removed.
* @throws IndexOutOfBoundsException if the index is out of range (index < 0 || index
>= getSteps()).
*/
public synchronized void removeStep(int index){
return null; // TODO
}
```

5.1.5 MeasurementStep

```
/*
* MeasurementStep.java
*
* Copyright (C) 2005 Project SQUID, http://www.cs.helsinki.fi/group/squid/
*
* This file is part of Ikayaki.
*
* Ikayaki is free software; you can redistribute it and/or modify
* it under the terms of the GNU General Public License as published by
* the Free Software Foundation; either version 2 of the License, or
* (at your option) any later version.
*
* Ikayaki is distributed in the hope that it will be useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
* GNU General Public License for more details.
*
* You should have received a copy of the GNU General Public License
* along with Ikayaki; if not, write to the Free Software
* Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
*/
package ikayaki;

/**
*  A single step in a measurement sequence. Each step can include multiple measurements
for improved precision.  A step can have a different volume and mass than the related
project, but by default the volume and mass of the project will be used. Only the project
may change the state and results of a measurement step.
All operations are thread-safe.
*
* @author
*/
public class MeasurementStep
/**
*  The project that owns this step, or null if there is no owner.
*/
private Project project = null;

/**
*  Tells if this step has been completed or not, or if a measurement is still running.
*/
private State state = READY;

/**
*  The time the measurements were completed, or null if that has not yet happened.
*/
private Date timestamp = null;

/**
*  The AF/Thermal value of this step, or a negative number if it has not been specified.
*/
private double stepValue = -1.0;
```

```
/**
*  The mass of this step’s sample, or a negative number to use the project’s default mass.
*/
private double mass = -1.0;

/**
*  The volume of this step’s sample, or a negative number to use the project’s default
volume.
*/
private double volume = -1.0;

/**
*  The individual measurement results that are part of this measurement step.
*/
private List<MeasurementResult> results = new ArrayList<MeasurementResult>();

/**
*  Creates a blank measurement step.
* @return the created measurement step.
*/
public MeasurementStep(){
return null; // TODO
}

/**
*  Creates a blank measurement step for a project.
* @param project the project who is the owner of this step.
* @return the created measurement step.
*/
public MeasurementStep(Project project){
return null; // TODO
}

/**
*  Creates a measurement step from the specified element.
* @param import the element from which this step will be created.
* @return the created measurement step.
* @throws NullPointerException if import is null.
* @throws IllegalArgumentException if the element was not in the right format.
*/
public MeasurementStep(Element import){
return null; // TODO
}
```

```
/**
*  Creates a measurement step from the specified element for a project.
* @param import the element from which this step will be created.
* @param project the project who is the owner of this step.
* @return the created measurement step.
* @throws NullPointerException if import is null.
* @throws IllegalArgumentException if the element was not in the right format.
*/
public MeasurementStep(Element import, Project project){
return null; // TODO
}

/**
*  Exports this step to a DOM element.
*/
public synchronized Element getElement(){
return null; // TODO
}

/**
*  Returns the owner project of this step, or null if there is no owner.
*/
public synchronized Project getProject(){
return null; // TODO
}

/**
*  Tells if this step has been completed or not, or if a measurement is still running.
*/
public synchronized State getState(){
return null; // TODO
}

/**
*  Sets the completion status of this step. Only the owner project may set the state.
* @throws NullPointerException if state is null.
*/
void synchronized setState(State state){
return null; // TODO
}

/**
```

```
*  Returns the time the measurements were completed, or null if that has not yet happened.
*/
public synchronized Date getTimestamp(){
return null; // TODO
}

/**
*  Returns the AF/Thermal value of this step, or a negative number if it has not been
specified.
*/
public synchronized double getStepValue(){
return null; // TODO
}

/**
*  Sets the value of this step. A negative value will clear it.
*/
public synchronized void setStepValue(double stepValue){
return null; // TODO
}

/**
*  Returns the mass of this step’s sample, or a negative number to use the project’s default
mass.
*/
public synchronized double getMass(){
return null; // TODO
}

/**
*  Sets the mass of this step’s sample. A negative value will clear it.
*/
public synchronized void setMass(double mass){
return null; // TODO
}

/**
*  Returns the volume of this step’s sample, or a negative number to use the project’s
default volume.
*/
public synchronized double getVolume(){
return null; // TODO
}
```

```
/**
*  Sets the volume of this step’s sample. A negative value will clear it.
*/
public synchronized void setVolume(double volume){
return null; // TODO
}

/**
*  Updates all of the measurement results with the owner project’s transformation matrix.
If there is no owner, an identity matrix will be used.
*/
synchronized void updateTransforms(){
return null; // TODO
}

/**
*  Returns the number of results in this step.
*/
public synchronized int getResults(){
return null; // TODO
}

/**
*  Returns the specified result from this step.
* @param index the index of the result.
* @return the specified result.
* @throws IndexOutOfBoundsException if the index is out of range (index < 0 || index
>= getResults()).
*/
public synchronized MeasurementResult getResult(int index){
return null; // TODO
}

/**
*  Appends a measurement result to this step. The transformation matrix of the result will
be updated automatically.
* @param result the result to be added.
* @throws NullPointerException if result is null.
*/
public synchronized void addResult(MeasurementResult result){
return null; // TODO
}
```

5.1.6 MeasurementStep.State

```
/*
* MeasurementStep.State.java
*
* Copyright (C) 2005 Project SQUID, http://www.cs.helsinki.fi/group/squid/
*
* This file is part of Ikayaki.
*
* Ikayaki is free software; you can redistribute it and/or modify
* it under the terms of the GNU General Public License as published by
* the Free Software Foundation; either version 2 of the License, or
* (at your option) any later version.
*
* Ikayaki is distributed in the hope that it will be useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
* GNU General Public License for more details.
*
* You should have received a copy of the GNU General Public License
* along with Ikayaki; if not, write to the Free Software
* Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
*/
package ikayaki;

/**
* The state of a measurement step. Options are READY, MEASURING, DONE_RECENTLY
and DONE.
*
* @author
*/
public enum State
```

5.1.7 MeasurementResult

```
/*
* MeasurementResult.java
*
* Copyright (C) 2005 Project SQUID, http://www.cs.helsinki.fi/group/squid/
*
* This file is part of Ikayaki.
*
```

```
* Ikayaki is free software; you can redistribute it and/or modify
* it under the terms of the GNU General Public License as published by
* the Free Software Foundation; either version 2 of the License, or
* (at your option) any later version.
*
* Ikayaki is distributed in the hope that it will be useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
* GNU General Public License for more details.
*
* You should have received a copy of the GNU General Public License
* along with Ikayaki; if not, write to the Free Software
* Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
*/
package ikayaki;

/**
*  A set of X, Y and Z values measured by the magnetometer. The raw XYZ values will
be rotated in 3D space by using a transformation matrix. The project will set and update
the transformation whenever its parameters are changed.
*
* @author
*/
public class MeasurementResult
/**
*  The type of this result. Can be either a background noise measurement, or a sample in
one of four rotations (0, 90, 180 or 270 degrees).
*/
private Type type;

/**
*  The unmodified measurements recieved from the squid.
*/
private Tuple3d rawTuple = new Tuple3d();

/**
*  The measurements with the rotation and transformation matrix applied.
*/
private Tuple3d tuple = new Tuple3d();

/**
*  Creates a new measurement result.
* @param type the type (background or rotation) of this result.
* @param x the measured X coordinate value.
* @param y the measured Y coordinate value.
* @param z the measured Z coordinate value.
* @return the created measurement result.
* @throws NullPointerException if type is null.
*/
public MeasurementResult(Type type, double x, double y, double z){
return null; // TODO
}

/**
*  Creates a measurement result from the specified element.  This will not apply the
transformation matrix.
* @param import the element from which this result will be created.
* @return the created measurement result.
* @throws NullPointerException if import is null.
* @throws IllegalArgumentException if the element was not in the right format.
*/
public MeasurementResult(Element import){
return null; // TODO
}

/**
*  Exports this result to a DOM element.
*/
public Element getElement(){
return null; // TODO
}

/**
*  Applies a transformation matrix to this result.
* @param transform the matrix to be applied. If null, will assume identity matrix.
*/
void setTransform(Matrix3d transform){
return null; // TODO
}

/**
*  Returns the type of this result (background or rotation).
*/
public Type getType(){
return null; // TODO
}
```

```
/**
*  Returns the rotated and transformed X coordinate of this result.
*/
public double getX(){
return null; // TODO
}

/**
*  Returns the rotated and transformed Y coordinate of this result.
*/
public double getY(){
return null; // TODO
}

/**
*  Returns the rotated and transformed Z coordinate of this result.
*/
public double getZ(){
return null; // TODO
}

/**
*  Returns the unmodified X coordinate of this result as recieved from the Squid.
*/
public double getRawX(){
return null; // TODO
}

/**
*  Returns the unmodified Y coordinate of this result as recieved from the Squid.
*/
public double getRawY(){
return null; // TODO
}

/**
*  Returns the unmodified Z coordinate of this result as recieved from the Squid.
*/
public double getRawZ(){
return null; // TODO
}
```

5.1.8 MeasurementResult.Type

```
/*
* MeasurementResult.Type.java
*
* Copyright (C) 2005 Project SQUID, http://www.cs.helsinki.fi/group/squid/
*
* This file is part of Ikayaki.
*
* Ikayaki is free software; you can redistribute it and/or modify
* it under the terms of the GNU General Public License as published by
* the Free Software Foundation; either version 2 of the License, or
* (at your option) any later version.
*
* Ikayaki is distributed in the hope that it will be useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
* GNU General Public License for more details.
*
* You should have received a copy of the GNU General Public License
* along with Ikayaki; if not, write to the Free Software
* Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
*/
package ikayaki;

/**
*  The orientation of the sample when it was measured. Options are BG, DEG0, DEG90,
DEG180 and DEG270.
*
* @author
*/
public enum Type
/**
* @return "BG", "0", "90", "180" or "270"
*/
public String getName(){
return null; // TODO
}


/**
*  Rotates the specified tuple from the orientation of this object to that of DEG0. Rotating
a BG or DEG0 will just copy the values directly.
* @param t old values that need to be rotated.
* @return a new object with the rotated values.
```

```
*/
public Tuple3d rotate(Tuple3d t){
return null; // TODO
}

/**
*  Rotates the specified tuple from the orientation of this object to that of DEG0. Rotating
a BG or DEG0 will just copy the values directly.
* @param t old values that need to be rotated.
* @param result where the new values will be saved.
* @return the same as the result parameter, or a new object if it was null.
*/
public Tuple3d rotate(Tuple3d t, Tuple3d result){
return null; // TODO
}
```

5.1.9 MeasurementValue

```
/*
* MeasurementValue.java
*
* Copyright (C) 2005 Project SQUID, http://www.cs.helsinki.fi/group/squid/
*
* This file is part of Ikayaki.
*
* Ikayaki is free software; you can redistribute it and/or modify
* it under the terms of the GNU General Public License as published by
* the Free Software Foundation; either version 2 of the License, or
* (at your option) any later version.
*
* Ikayaki is distributed in the hope that it will be useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
* GNU General Public License for more details.
*
* You should have received a copy of the GNU General Public License
* along with Ikayaki; if not, write to the Free Software
* Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
*/
package ikayaki;

/**
*  Algorithms for calculating values from the measurements. A MeasurementValue object
```

```
will be passed to the getValue() method of a project to retrieve the desired value.
*
* @author
*/
public abstract class MeasurementValue<T>
/**
*  Calculates the average of all X components.
*/
public static final MeasurementValue<Double> X;

/**
*  Calculates the average of all Y components.
*/
public static final MeasurementValue<Double> Y;

/**
*  Calculates the average of all Z components.
*/
public static final MeasurementValue<Double> Z;

/**
*  Calculates the declination from the component averages.
*/
public static final MeasurementValue<Double> DECLINATION;

/**
*  Calculates the inclination from the component averages.
*/
public static final MeasurementValue<Double> INCLINATION;

/**
*  Calculates the length of the vector from the component averages.
*/
public static final MeasurementValue<Double> MOMENT;

/**
*  Calculates the remanence from the component averages and the sample's volume.
*/
public static final MeasurementValue<Double> REMANENCE;

/**
*  Calculates the remanence relative to the first measurement's remanence.
```

```
*/
public static final MeasurementValue<Double> RELATIVE_REMANENCE;

/**
*  Calculates the Theta 63 value from the measurement result set.
*/
public static final MeasurementValue<Double> THETA63;

/**
*  A short name for the value.
*/
private String caption;

/**
*  The unit of the value.
*/
private String unit;

/**
*  A long description of the value.
*/
private String description;

/**
*  Creates a new measurement value.
* @param caption a short name for the value.
* @param unit the unit of the value.
* @param description a long description of the value.
* @return the created measurement value.
* @throws NullPointerException if any of the arguments is null.
*/
public MeasurementValue(String caption, String unit, String description){
return null; // TODO
}

/**
*  Calculates a specific value from a measurement step.
* @param step the step from which the value will be calculated.
* @return the calculated value, or null if it was not possible to calculate it.
* @throws NullPointerException if step is null.
*/
abstract T getValue(MeasurementStep step){
return null; // TODO
}

/**
*  Returns a short name for the value.
*/
public String getCaption(){
return null; // TODO
}

/**
*  Returns the unit of the value.
*/
public String getUnit(){
return null; // TODO
}

/**
*  Returns a long description of the value.
*/
public String getDescription(){
return null; // TODO
}
```

5.1.10 ProjectEvent

```
/*
* ProjectEvent.java
*
* Copyright (C) 2005 Project SQUID, http://www.cs.helsinki.fi/group/squid/
*
* This file is part of Ikayaki.
*
* Ikayaki is free software; you can redistribute it and/or modify
* it under the terms of the GNU General Public License as published by
* the Free Software Foundation; either version 2 of the License, or
* (at your option) any later version.
*
* Ikayaki is distributed in the hope that it will be useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
* GNU General Public License for more details.
*
```

```
* You should have received a copy of the GNU General Public License
* along with Ikayaki; if not, write to the Free Software
* Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
*/
package ikayaki;

/**
*  ProjectEvent is used to notify others about the state change of a project.
*
* @author
*/
public class ProjectEvent   extends EventObject
/**
*  The project that sent this event.
*/
private Project project;

/**
*  The type of event this is.
*/
private Type type;

/**
*  Creates a new project event.
* @param project the project that sends this event.
* @param type the type of the event.
* @return the created event.
* @throws NullPointerException if any of the arguments is null.
*/
public ProjectEvent(Project project, Type type){
return null; // TODO
}

/**
*  Returns the project that sent this event.
*/
public Project getProject(){
return null; // TODO
}

/**
*  Returns the type of this event.
*/
```

```
public Type getType(){
return null; // TODO
}
```

5.1.11 ProjectEvent.Type

```
/*
* ProjectEvent.Type.java
*
* Copyright (C) 2005 Project SQUID, http://www.cs.helsinki.fi/group/squid/
*
* This file is part of Ikayaki.
*
* Ikayaki is free software; you can redistribute it and/or modify
* it under the terms of the GNU General Public License as published by
* the Free Software Foundation; either version 2 of the License, or
* (at your option) any later version.
*
* Ikayaki is distributed in the hope that it will be useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
* GNU General Public License for more details.
*
* You should have received a copy of the GNU General Public License
* along with Ikayaki; if not, write to the Free Software
* Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
*/
package ikayaki;

/**
*  The type of a project event.  Options are STATE_CHANGED, DATA_CHANGED,
FILE_SAVED.
*
* @author
*/
public enum Type
```

5.1.12 ProjectListener

```
/*
* ProjectListener.java
*
* Copyright (C) 2005 Project SQUID, http://www.cs.helsinki.fi/group/squid/
```

```
*
* This file is part of Ikayaki.
*
* Ikayaki is free software; you can redistribute it and/or modify
* it under the terms of the GNU General Public License as published by
* the Free Software Foundation; either version 2 of the License, or
* (at your option) any later version.
*
* Ikayaki is distributed in the hope that it will be useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
* GNU General Public License for more details.
*
* You should have received a copy of the GNU General Public License
* along with Ikayaki; if not, write to the Free Software
* Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
*/
package ikayaki;

/**
*  Defines a listener for project events.
*
* @author
*/
public interface ProjectListener   extends EventListener
/**
*  Will be invoked whenever a project event happens.
* @param event the event that happened.
*/
public void projectUpdated(ProjectEvent event){
return null; // TODO
}
```

5.1.13 MeasurementEvent

```
/*
* MeasurementEvent.java
*
* Copyright (C) 2005 Project SQUID, http://www.cs.helsinki.fi/group/squid/
*
* This file is part of Ikayaki.
*
* Ikayaki is free software; you can redistribute it and/or modify
* it under the terms of the GNU General Public License as published by
* the Free Software Foundation; either version 2 of the License, or
* (at your option) any later version.
*
* Ikayaki is distributed in the hope that it will be useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
* GNU General Public License for more details.
*
* You should have received a copy of the GNU General Public License
* along with Ikayaki; if not, write to the Free Software
* Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
*/
package ikayaki;

/**
*  MeasurementEvent is used to notify listeners about the stages of an ongoing measurement.
*
* @author
*/
public class MeasurementEvent   extends EventObject
/**
*  The project whose measurement sent this event.
*/
private Project project;

/**
*  The measurement that sent this event.
*/
private MeasurementStep step;

/**
*  The type of event this is.
*/
private Type type;

/**
*  Creates a new measurement event.
* @param project the project whose measurement sent this event.
* @param step the measurement that sent this event.
* @param type the type of event this is.
* @return the created event.
* @throws NullPointerException if any of the arguments is null.
```

```
*/
public MeasurementEvent(Project project, MeasurementStep step, Type type){
return null; // TODO
}

/**
*  Returns the project whose measurement sent this event.
*/
public Project getProject(){
return null; // TODO
}

/**
*  Returns the measurement that sent this event.
*/
public MeasurementStep getStep(){
return null; // TODO
}

/**
*  Returns the type of event this is.
*/
public Type getType(){
return null; // TODO
}
```

5.1.14 MeasurementEvent.Type

```
/*
* MeasurementEvent.Type.java
*
* Copyright (C) 2005 Project SQUID, http://www.cs.helsinki.fi/group/squid/
*
* This file is part of Ikayaki.
*
* Ikayaki is free software; you can redistribute it and/or modify
* it under the terms of the GNU General Public License as published by
* the Free Software Foundation; either version 2 of the License, or
* (at your option) any later version.
*
* Ikayaki is distributed in the hope that it will be useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
* GNU General Public License for more details.
*
* You should have received a copy of the GNU General Public License
* along with Ikayaki; if not, write to the Free Software
* Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
*/
package ikayaki;

/**
*  The type of a measurement event. Options are STEP_START, STEP_END, STEP_ABORTED,
HANDLER_MOVE, HANDLER_ROTATE, HANDLER_STOP, DEMAGNETIZE_START,
DEMAGNETIZE_END, VALUE_MEASURED.
*
* @author
*/
public enum Type
```

5.1.15 MeasurementListener

```
/*
* MeasurementListener.java
*
* Copyright (C) 2005 Project SQUID, http://www.cs.helsinki.fi/group/squid/
*
* This file is part of Ikayaki.
*
* Ikayaki is free software; you can redistribute it and/or modify
* it under the terms of the GNU General Public License as published by
* the Free Software Foundation; either version 2 of the License, or
* (at your option) any later version.
*
* Ikayaki is distributed in the hope that it will be useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
* GNU General Public License for more details.
*
* You should have received a copy of the GNU General Public License
* along with Ikayaki; if not, write to the Free Software
* Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
*/
package ikayaki;

/**
```

```
*  Defines a listener for measurement events.
*
* @author
*/
public interface MeasurementListener   extends EventListener
/**
*  Will be invoked whenever a measurement event happens.
* @param event the event that happened.
*/
public void measurementUpdated(MeasurementEvent event){
return null; // TODO
}
```

5.2 Squid interface

5.2.1 Squid

```
/*
* Squid.java
*
* Copyright (C) 2005 Project SQUID, http://www.cs.helsinki.fi/group/squid/
*
* This file is part of Ikayaki.
*
* Ikayaki is free software; you can redistribute it and/or modify
* it under the terms of the GNU General Public License as published by
* the Free Software Foundation; either version 2 of the License, or
* (at your option) any later version.
*
* Ikayaki is distributed in the hope that it will be useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
* GNU General Public License for more details.
*
* You should have received a copy of the GNU General Public License
* along with Ikayaki; if not, write to the Free Software
* Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
*/
package ikayaki.squid;

/**
*  Offers an interface for controlling the SQUID system. Reads settings from the Settings
class.  Creates instances of the degausser, handler and magnetometer classes and offers
```

```
handles for them.
*
* @author
*/
public class Squid
/**
* Instance of the Squid interface.
*/
private static final Squid instance;

/**
* The project that is currently using the Squid, or null if no project is using it.
*/
private Project owner;

/**
* Instance of the degausser interface.
*/
private final Degausser degausser;

/**
* Instance of the handler interface.
*/
private final Handler handler;

/**
* Instance of the magnetometer interface.
*/
private final Magnetometer magnetometer;

/**
*  Returns a reference to the Squid. If it has not yet been created, will create one.
*/
public static synchronized Squid instance(){
return null; // TODO
}

/**
*  Initializes the Squid interface.  Creates instances of Degausser, Handler and Magnetometer.
*/
private Squid(){
return null; // TODO
```

```
}

/**
*  Returns an interface for controlling the degausser.
* @return Handler for Degausser if available
*/
public Degausser getDegausser(){
return null; // TODO
}

/**
*  Returns an interface for controlling the handler.
* @return Handler for Handler if available
*/
public Handler getHandler(){
return null; // TODO
}

/**
*  Returns an interface for controlling the magnetometer.
* @return Handler for Magnetometer if available
*/
public Magnetometer getMagnetometer(){
return null; // TODO
}

/**
*  Checks which settings have changed and updates all the device interfaces. This method
should be called when changes are made to the device parameters.

This method starts a worker thread that will update the settings. If the current project has a
measurement running, the thread will keep on retrying until the measurement is finished.
Multiple calls to this method within a short period of time will update the settings only
once.
*/
public synchronized void updateSettings(){
return null; // TODO
}

/**
*  Checks whether all devices are working correctly.
* @return true if everything is correct, otherwise false.
*/
public synchronized boolean isOK(){
```

```
return null; // TODO
}

/**
*  Sets the owner of the Squid. Only one project may have access to the Squid at a time.
This method may be called only from the Project class.
* @param owner the project that will have exclusive access to the Squid.
* @return true if successful, false if the existing owner had a running measurement.
*/
public synchronized boolean setOwner(Project owner){
return null; // TODO
}

/**
*  Returns project that is currently using the Squid.
* @return the project, or null if none is using the Squid.
*/
public synchronized Project getOwner(){
return null; // TODO
}
```

5.2.2 Degausser

```
/*
* Degausser.java
*
* Copyright (C) 2005 Project SQUID, http://www.cs.helsinki.fi/group/squid/
*
* This file is part of Ikayaki.
*
* Ikayaki is free software; you can redistribute it and/or modify
* it under the terms of the GNU General Public License as published by
* the Free Software Foundation; either version 2 of the License, or
* (at your option) any later version.
*
* Ikayaki is distributed in the hope that it will be useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
* GNU General Public License for more details.
*
* You should have received a copy of the GNU General Public License
* along with Ikayaki; if not, write to the Free Software
* Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
```

```
*/
package ikayaki.squid;

/**
*  Offers an interface for controlling the degausser (demagnetizer).  Because the data link is implemented in the degausser by a single board computer running a small basic program, the response time of the degausser to commands is slow. This class will make sure that commands are not sent faster than the device can handle.
*
* @author
*/
public class Degausser   implements SerialIOListener
/*
Event A: On SerialIOEvent - reads the message and puts it in a buffer
*/


/**
* buffer for incoming messages, readed when needed.
*/
private Stack messageBuffer;

/**
* Degaussers current status
*/
private String status;

/**
* COM port for communication.
*/
private SerialIO serialIO;

/**
* (X, Y, Z) = (0,1,2) default axis Z
*/
private int degausserCoil;

/**
* 0->3000 default amp 0
*/
private int degausserAmplitude;

/**
* 1-9 seconds default delay 1 second
```

```
*/
private int degausserDelay;

/**
* (3, 5, 7, 9) default 3
*/
private int degausserRamp;

/**
* Z=Zero, T=Tracking, ?=Unknown
*/
private char degausserRamp;

/**
*  Creates a new degausser interface.  Opens connection to degausser COM port (if not
open yet) and reads settings from the Setting class.
*/
public Degausser(){
return null; // TODO
}

/**
*  Checks which settings have changed and updates the degausser interface. This method
will be called by the Squid class.
*/
public void updateSettings(){
return null; // TODO
}

/**
*  Sets coil X,Y,Z.
* @param coil coil to set on.
*/
private void setCoil(char coil){
return null; // TODO
}

/**
*  Sets amplitude to ramp, range 0 to 3000.
* @param amplitude amplitude to demag.
*/
private void setAmplitude(int amplitude){
return null; // TODO
}

/**
*  Performs Ramp up.
*/
private void executeRampUp(){
return null; // TODO
}

/**
*  Brings Ramp down.
*/
private void executeRampDown(){
return null; // TODO
}

/**
*  Performs Ramp up and down.
*/
private void executeRampCycle(){
return null; // TODO
}

/**
*  Performs full sequence to demagnetize Z coil with the given amplitude.  Blocking
method.
* @param amplitude amplitude to demag.
* @return true if process was sended succesfully, otherwise false.
*/
public boolean demagnetizeZ(int amplitude){
return null; // TODO
}

/**
*  Performs full sequence to demagnetize Y (and X) coil with the given amplitude.
Blocking method.
* @param amplitude amplitude to demag.
* @return true if process was sended succesfully, otherwise false.
*/
public boolean demagnetizeY(int amplitude){
return null; // TODO
}
```

```
/**
*  Sends status query to degausser and returns answer. Blocking.
* @return Z=Zero, T=Tracking, ?=Unknown
*/
public char getRampStatus(){
return null; // TODO
}

/**
*  Sends ramp query to degausser and returns answer. Blocking.
* @return 3, 5, 7 or 9
*/
public int getRamp(){
return null; // TODO
}

/**
*  Sends delay query to degausser and returns answer. Blocking.
* @return 1 to 9 as seconds
*/
public int getDelay(){
return null; // TODO
}

/**
*  Sends coil query to degausser and returns answer. Blocking.
* @return X=X Axis, Y=Y Axis, Z=Z Axis, ?=Unknown
*/
public char getCoil(){
return null; // TODO
}

/**
*  Sends amplitude query to degausser and returns answer. Blocking.
* @return 0 to 3000
*/
public int getAmplitude(){
return null; // TODO
}

/**
*  Checks if connection is ok.
* @return true if ok.
```

```
*/
public boolean isOK(){
return null; // TODO
}
```

5.2.3 Handler

```
/*
* Handler.java
*
* Copyright (C) 2005 Project SQUID, http://www.cs.helsinki.fi/group/squid/
*
* This file is part of Ikayaki.
*
* Ikayaki is free software; you can redistribute it and/or modify
* it under the terms of the GNU General Public License as published by
* the Free Software Foundation; either version 2 of the License, or
* (at your option) any later version.
*
* Ikayaki is distributed in the hope that it will be useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
* GNU General Public License for more details.
*
* You should have received a copy of the GNU General Public License
* along with Ikayaki; if not, write to the Free Software
* Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
*/
package ikayaki.squid;

/**
*  Offers an interface for controlling the sample handler.
*
* @author
*/
public class Handler   implements SerialIOListener
/*
Event A: On SerialIOEvent - reads message and puts it in a buffer
*/


/**
* Buffer for incoming messages, readed when needed.
*/
```

```
private Stack messageBuffer;

/**
* Handlers current status.
*/
private String status;

/**
* COM port for communication.
*/
private SerialIO serialIO;

/**
* Value between 0 and 127 default 5. Settings in the 20-50 range are usually employed.
*/
private int acceleration;

/**
* Value between 0 and 127 default 10. Settings in the 20-50 range are usually employed.
*/
private int deceleration;

/**
* Value between 50 and 12 000. The decimal number issued is 10 times the actual pulse
rate to the motor. Since the motor requires 200 pulses (full step) or 400 pulses (half step)
per revolution, a speed setting of M10000 sets the motor to revolve at 5 revolutions per
second in full step or 2.5 revolutions in half step. This rate is one-half the sample rate
rotation due to the pulley ratios. The sample handler is set up at the factory for half
stepping.
*/
private int velocity;

/**
* Speed in measurement, should be small.
*/
private int measurementVelocity;

/**
* 5 end of move, previous G command complete, 7 hard limit stop, G motor is currently
indexing.
*/
private String handlerStatus;
```

```
/**
* Value between 1 and 16,777,215.
*/
private int currentPosition;

/**
* Value between 1 and 16,777,215.
*/
private int homePosition;

/**
* AF demag position for transverse.
*/
private int transverseYAFPosition;

/**
* Axial AF demag position in steps, must be divisible by 10. Relative to Home.
*/
private int axialAFPosition;

/**
* Position in steps, must be divisible by 10. Relative to Home.
*/
private int backgroundPosition;

/**
* Position in steps, must be divisible by 10. Relative to Home.
*/
private int measurementPosition;

/**
* Angles are between 0 (0) and 2000 (360).
*/
private int currentRotation;

/**
*  Creates a new handler interface.  Opens connection to handler COM port and reads
settings from the Settings class.
*/
public Handler(){
return null; // TODO
}

/**
*  Checks which settings have changed and updates the handler interface.  This method
will be called by the Squid class.
*/
public void updateSettings(){
return null; // TODO
}

/**
*  Returns current status on Sample Handler.
* @return
0 Normal, no service required
1 Command error, illegal command sent
2 Range error, an out of range numeric parameter was sent
3 Command invalid while moving (e.g. G, S, H)
4 Command only valid in program (e.g. I, U, L)
5 End of move notice, a previous G command is complete
6 End of wait notice, a previous W command is complete
7 Hard limit stop, the move was stopped by the hard limit
8 End of program notice, internal program has completed
G Motor is indexing and no other notice pending.
*/
public char getStatus(){
return null; // TODO
}

/**
*  Returns current known position.
* @return Value between 1 and 16,777,215
*/
public int getPosition(){
return null; // TODO
}

/**
*  Returns current known rotation.
* @return Value between 0 and 2000
*/
```

```
public int getRotation(){
return null; // TODO
}

/**
*  checks if connection is ok.
* @return True if ok
*/
public boolean isOK(){
return null; // TODO
}

/**
*  Commands the holder to move to home position. Only starts movement, needs to poll
with join() when movement is finished.
*/
public void moveToHome(){
return null; // TODO
}

/**
*  Commands the holder to move to degauss position. Only starts movement, needs to
poll with join() when movement is finished.
*/
public void moveToDegausser(){
return null; // TODO
}

/**
*  Commands the holder to move to degauss position. Only starts movement, needs to
poll with join() when movement is finished.
*/
public void moveToDegausser(){
return null; // TODO
}

/**
*  Commands the holder to move to measure position. Only starts movement, needs to
poll with join() when movement is finished.
*/
public void moveToMeasurement(){
return null; // TODO
}
```

```
/**
*  Commands the holder to move to background position.  Only starts movement, needs to poll with join() when movement is finished.
*/
public void moveToBackground(){
return null; // TODO
}

/**
*  Commands the holder to move to the specified position.  Value must be between 1 and 16,777,215.  Return true if good pos-value and moves handler there.  Only starts movement, needs to poll with join() when movement is finished.
* @param pos the position where the handler will move to.
* @return true if given position was ok, otherwise false.
*/
public boolean moveToPos(int pos){
return null; // TODO
}

/**
*  Commands the handler to stop its current job.
*/
public void stop(){
return null; // TODO
}

/**
*  Rotates the handler to the specified angle. If angle is over than 360 or lower than 0, it is divided by 360 and value is remainder. Only starts movement, needs to poll with join() when movement is finished.
* @param angle the angle in degrees to rotate the handler to.
*/
public void rotateTo(int angle){
return null; // TODO
}

/**
*  Sends message to handler go online (@0).
*/
private void setOnline(){
return null; // TODO
}
```

```
/**
*  Sends message to handler to set acceleration (Aa).
* @param a Acceleration is a number from 0 to 127
*/
private void setAcceleration(int a){
return null; // TODO
}

/**
*  Sends message to handler to set deceleration (Dd).
* @param a Deceleration is a number from 0 to 127
*/
private void setDeceleration(int d){
return null; // TODO
}

/**
*  Sends message to handler to set base speed. Base rate is the speed at which the motor
motion starts and stops. (Bb).
* @param b Base Speed is pulses per second and has a range of 50 to 5000.
*/
private void setBaseSpeed(int b){
return null; // TODO
}

/**
*  Sends message to handler to set maximum velocity (Mv).
* @param v Velocity range is 50 to 20,000
*/
private void setVelocity(int v){
return null; // TODO
}

/**
*  This command causes the POWER pin to be pulled low within a specified number of
ticks after a move of the motor is completed and it will stay low until just prior to the start
of the next move command. This command allows the holding torque of the motor to be
turned off automatically after a delay for the mechanical system stabilize thus reducing
power consumption and allowing the motor to be turned by hand if this feature is required.
If the value is zero then the power is left on forever. (CH h).
* @param h value from 0 to 127 representing the number of clock ticks to leave power
on the motor after a move.
```

```
*/
private void setHoldTime(int h){
return null; // TODO
}

/**
*  numbers.  The crystal frequency is used by the chip for setting the base speed and
maximum speed and for controlling the time for the wait command. (CX cf).
* @param cf frequence range is 4,000,000 to 8,000.000
*/
private void setCrystalFrequence(int cf){
return null; // TODO
}

/**
*  This command stops execution of the internal program if it is used in the program.  If
the motor is indexing it will ramp down and then stop.  Use this command to stop the
motor after issuing a slew command. (Q).
*/
private void stopExecution(){
return null; // TODO
}

/**
*  Slew the motor up to maximum speed and continue until reaching a hard limit switch
or receiving a quit (Q) command. (S).
*/
private void performSlew(){
return null; // TODO
}

/**
*  Set the motor direction of movement to positive. (+).
*/
private void setMotorPositive(){
return null; // TODO
}

/**
*  Set the motor direction of movement to negative. (-).
*/
private void setMotorNegative(){
return null; // TODO
}

/**
*  Set the number of steps to move for the G command. (N s).
* @param s steps range is 0 to 16,777,215
*/
private void setSteps(int s){
return null; // TODO
}

/**
*  Set absolute position to move for the G command. (P p).
* @param p position range is 0 to 16,777,215
*/
private void setPosition(int p){
return null; // TODO
}

/**
*  Send handler on move (G).
*/
private void go(){
return null; // TODO
}

/**
*  Wait for handler to be idle.  Blocking (F). Without the this command the SMC25
(Handler system) will continue to accept commands while the motor is moving.  This
may be desirable, as when changing speed during a move or working with the inputs or
outputs.  Or it may be undesirable, such as when you wish to make a series of indexes.
Without the this command any subsequent Go commands received while the motor is
indexing would set the "Not allowed while moving" message. Caution: If this command
is used while the motor is executing a Slew command the only way to stop is with a reset
or a hard limit switch input.
*/
private void join(){
return null; // TODO
}

/**
*  Gives result for wanted registery. (V v).
* @param v A Acceleration
B Base speed
```

```
D Deceleration

E Internal program

G Steps remaining in current move. Zero if not indexing.

H Hold time

I Input pins

J Slow jog speed

M Maximum speed

N Number of steps to index

O Output pins

P Position. If motor is indexing this returns the position at the end of the index.

R Internal program pointer used by trace (T) or continue (X) commands.  Also updated
by enter (E) command.

W Ticks remaining on wait counter

X Crystal frequency
* @return returns registery as string
*/
private String verify(char v){
return null; // TODO
}

/**
*  Set the position register.  This command sets the internal absolute position counter to
the value of r. (Z r).
* @param r position range is 0 to 16,777,215
*/
private void setPositionRegister(int r){
return null; // TODO
}

/**
*  Poll the device for any waiting messages such as errors or end of move. (
* @return 0 Normal, no service required

1 Command error, illegal command sent

2 Range error, an out of range numeric parameter was sent

3 Command invalid while moving (e.g. G, S, H)

4 Command only valid in program (e.g. I, U, L)

5 End of move notice, a previous G command is complete

6 End of wait notice, a previous W command is complete
```

```
7 Hard limit stop, the move was stopped by the hard limit
8 End of program notice, internal program has completed
G Motor is indexing and no other notice pending
*/
private char pollMessage(){
return null; // TODO
}
```

5.2.4 Magnetometer

```
/*
* Magnetometer.java
*
* Copyright (C) 2005 Project SQUID, http://www.cs.helsinki.fi/group/squid/
*
* This file is part of Ikayaki.
*
* Ikayaki is free software; you can redistribute it and/or modify
* it under the terms of the GNU General Public License as published by
* the Free Software Foundation; either version 2 of the License, or
* (at your option) any later version.
*
* Ikayaki is distributed in the hope that it will be useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
* GNU General Public License for more details.
*
* You should have received a copy of the GNU General Public License
* along with Ikayaki; if not, write to the Free Software
* Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
*/
package ikayaki.squid;

/**
*  Offers an interface for controlling the magnetometer."
*
* @author
*/
public class Magnetometer   implements SerialIOListener
/*
Event A: On SerialIOEvent - reads the message and puts it in a buffer
*/
```

```
/**
* Buffer for incoming messages, readed when needed.
*/
private Stack messageBuffer;

/**
* Magnetometer’s current status.
*/
private String status;

/**
* COM port for communication.
*/
private SerialIO serialIO;

/**
*  Creates a new magnetometer interface. Opens connection to Magnetometer COM port
(if its not open already) and reads settings from the Setting class.
*/
public Magnetometer(){
return null; // TODO
}

/**
*  Checks which settings have changed and updates the magnetometer interface.  This
method will be called by the Squid class.
*/
public void updateSettings(){
return null; // TODO
}

/**
*  Reset settings for axis.
* @param axis x,y,z or a (all). In lower case.
*/
private String reset(char axis){
return null; // TODO
}

/**
*  Reset counter for axis.
* @param axis x,y,z or a (all). In lower case.
*/
```

```
private void resetCounter(char axis){
return null; // TODO
}

/**
* Used for configuring Magnetometer parameters. See subcommand for usages.
* @param axis x,y,z or a (all)
* @param subcommand  The CONFIGURE subcommands follow:
```

"F" Set filter configuration. The data subfield sets the filter to the indicated range. The four possible data values are: "1" One Hertz Filter; 1 Hz "T" Ten Hertz Filter; 10 Hz "H" One hundred Hertz Filter; 100 Hz "W" Wide band filter; WB

"R" Set DC SQUID electronic range. The data subfield selects the range desired. The four possible data values are: "1" One time range; 1x "T" Ten times range; 10x "H" One hundred times range; 100x "E" Extended range; 1000x

"S" Set/Reset the fast-slew option. Two data values are possible: "E" Enable the fast-slew; turn it on. "D" Disable the fast-slew; turn it off.

"L" This subcommand opens or closes the SQUID feedback loop or resets the analog signal to +/- 1/2 flux quantum about zero. The three possible data values are: "O" Open the feedback loop. (This command also zeros the flux counter) "C" Close the feedback loop. "P" Pulse-reset (open then close) the feedback loop. (This command also zeros the flux counter)

```
* @param option see data values from subcommands.
*/
private void configure(char axis, char subcommand, char option){
return null; // TODO
}

/**
*  axis is x,y,x or a (all).
* @param axis x,y,z or a (all). In lower case.
*/
private void latchAnalog(char axis){
return null; // TODO
}

/**
*  axis is x,y,x or a (all).
* @param axis x,y,z or a (all). In lower case.
*/
private void latchCounter(char axis){
return null; // TODO
}
```

```
/**
*  Generic send message sender, use with caution and knowledge.  Checks if commands are good.
* @param axis x,y,z. In lower case.
* @param command  "D" Send back the analog data last captured with the LATCH command. The <data> field is not required.

"C" Send back the counter value last captured with the LATCH command.  The <data> field is not required.

"S" Send back status. Various pieces of status can be sent by the magnetometer electronics.
* @param datavalues  Datavalues one or more:

"A" Send back all status.

"F" Send back all filter status.

"R" Send back all range status.

"S" Send back slew status.

"L" Send back SQUID feedback loop status. Return feedback, waiting time?
* @return Returns data wanted, see command and datavalue
*/
private String getData(char axis, char command, String datavalues){
return null; // TODO
}

/**
*  Pulse reset and opens feedback loop for axis.  Need to be done to all axes before measuring.
* @param axis x,y,z or a (all). In lower case.
*/
public void openLoop(char axis){
return null; // TODO
}

/**
*  Clears flux counter for axis. Need to be done to all axes before measuring.
* @param axis x,y,z or a (all). In lower case.
*/
public void clearFlux(char axis){
return null; // TODO
}

/**
*  Waits for magnetometer to settle down. Blocking.
```

```
*/
public void join(){
return null; // TODO
}

/**
*  Calls first openLoop(a) and clearFlux(a).  Latches axes, reads counters and analog.
Calculates data from them and returns them.
* @return Returns 3 double values in following order: (x,y,z)
*/
public Double[] readData(){
return null; // TODO
}

/**
*  Returns filter configurations for all axis. Blocking.
* @return return filter values for all axis in order (x,y,z).
Values
"1" One Hertz Filter; 1 Hz
"T" Ten Hertz Filter; 10 Hz
"H" One hundred Hertz Filter; 100 Hz
"W" Wide band filter; WB
*/
public char[] getFilters(){
return null; // TODO
}

/**
*  Returns range configurations for all axis. Blocking.
* @return return filter values for all axis in order (x,y,z).
Values:
"1" One time range; 1x
"T" Ten times range; 10x
"H" One hundred times range; 100x
"E" Extended range; 1000x
*/
public char[] getRange(){
return null; // TODO
}
```

```
/**
*  Returns Fast Slew options value. Blocking.
* @return true if Fast Slew is on, false if not
*/
public boolean getSlew(){
return null; // TODO
}

/**
*  Returns if Loops have been opened on axes. Blocking.
* @return return Loop status for all axis in order (x,y,z). Values true = on, false = off.
*/
public boolean[] getLoop(){
return null; // TODO
}

/**
*  Checks if connection is ok.
* @return true if ok.
*/
public boolean isOK(){
return null; // TODO
}
```

5.3 Squid emulator

5.3.1 SquidEmulator

```
/*
* SquidEmulator.java
*
* Copyright (C) 2005 Project SQUID, http://www.cs.helsinki.fi/group/squid/
*
* This file is part of Ikayaki.
*
* Ikayaki is free software; you can redistribute it and/or modify
* it under the terms of the GNU General Public License as published by
* the Free Software Foundation; either version 2 of the License, or
* (at your option) any later version.
*
* Ikayaki is distributed in the hope that it will be useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
* GNU General Public License for more details.
*
* You should have received a copy of the GNU General Public License
* along with Ikayaki; if not, write to the Free Software
* Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
*/
package ikayaki.squid;

/**
* This class tries to emulate behavior of real squid-system. It starts 3 threads (handler,magnetometer,dega
opens COM-ports for them and adds SerialIO Listeners. Threads generates random
data values or loaded values as results and generates random error situations to see that
program using real squid system does survive those. Uses 2-3 COM ports. Usage SquidEmulator
x z.. filename where x is 0 or 1 and indicates if Magnetometer and Demagnetizer are on
same COM port. z... values are COM ports. filename is name of log file we are using or
it is existing log file, which is used to generate same sequence used to verify that old and
new program behaves same way.
*
* @author
*/
public class SquidEmulation
/*
Event A: On New IO Message - reads message and puts it in Buffer
*/


/**
* indicates if system have been started
*/
private bool online;


/**
* log file we are using read or write
*/
private File logFile;


/**
* indicates have we loaded log file for using or are we writing it
*/
private boolean usingOldLog;


/**
* value between 0 and 127 default 5. Settings in the 20-50 range are usually employed.
*/
```

```
private int acceleration;

/**
* value between 0 and 127 default 10. Settings in the 20-50 range are usually employed.
*/
private int deceleration;

/**
* value between 50 and 12 000. The decimal number issued is 10 times the actual pulse
rate to the motor. Since the motor requires 200 pulses (full step) or 400 pulses (half step)
per revolution, a speed setting of M10000 sets the motor to revolve at 5 revolutions per
second in full step or 2.5 revolutions in half step. This rate is one-half the sample rate
rotation due to the pulley ratios. The sample handler is set up at the factory for half
stepping.
*/
private int velocity;

/**
* 5 end of move, previous G command complete, 7 hard limit stop, G motor is currently
indexing
*/
private String handlerStatus;

/**
* value between 1 and 16,777,215
*/
private int commandedDistance;

/**
* value between 1 and 16,777,215
*/
private int currentPosition;

/**
* value between 1 and 16,777,215
*/
private int homePosition;

/**
* angles are between 0 (0) and 2000 (360)
*/
private int commandedRotation;
```

```
/**
* angles are between 0 (0) and 2000 (360)
*/
private int currentRotation;

/**
* (X, Y, Z) = (0,1,2) default axis Z
*/
private int degausserCoil;

/**
* 0->3000 default amp 0
*/
private int degausserAmplitude;

/**
* 1-9 seconds default delay 1 second
*/
private int degausserDelay;

/**
* (3, 5, 7, 9) default 3
*/
private int degausserRamp;

/**
* Z=Zero, T=Tracking, ?=Unknown
*/
private char degausserRamp;

/**
* starts Threads which reads messages from selected COM port. Own listener for each.
Offers write commads to port too.
*/
private SerialIO[] serialIO;

/**
* private class which implements Thread and runs handler emulation process. Process
incoming messages and sends data back. When message comes, process it (wait if needed
for a while), updates own status and sends result back.
*/
```

```
private HandlerEmu handler;

/**
* private class which implements Thread and runs magnetometer emulation process.
Process incoming messages and sends data back. When message comes, process it (wait
if needed for a while), updates own status and sends result back.
*/
private MagnetometerEmu magnetometer;

/**
* private class which implements Thread and runs degausser emulation process. Process
incoming messages and sends data back. When message comes, process it (wait if needed
for a while), updates own status and sends result back.
*/
private DegausserEmu degausser;

/**
*  send message to SerialIO to be sented.
* @param message any message reply we are sending back
* @param port port number to be sent
*/
public void writeMessage(String message ,int port)){
return null; // TODO
}

/**
*  First creates or loads log file and sets settings. Runs sequence where read data from
buffer and run cheduled actions (move, rotate, demag, measure) and send feedback to
COM ports.
*/
public static void main(String[] args){
return null; // TODO
}
```

5.4 Serial communication

5.4.1 SerialIO

```
/*
* SerialIO.java
*
* Copyright (C) 2005 Project SQUID, http://www.cs.helsinki.fi/group/squid/
```

```
*
* This file is part of Ikayaki.
*
* Ikayaki is free software; you can redistribute it and/or modify
* it under the terms of the GNU General Public License as published by
* the Free Software Foundation; either version 2 of the License, or
* (at your option) any later version.
*
* Ikayaki is distributed in the hope that it will be useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
* GNU General Public License for more details.
*
* You should have received a copy of the GNU General Public License
* along with Ikayaki; if not, write to the Free Software
* Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
*/
package ikayaki.squid;

/**
* This class represents hardware layer to serial port communications.
*
* @author
*/
public class SerialIO   implements SerialPortEventListener
/*
Event A: On new SerialPortEvent - generates new SerialMessageArrivedEvent if a data
message from serial port is received.
*/


/**
* contains last received message from the serial port that this SerialIO represents.
*/
private String lastMessge = null;


/**
* Creates an instance of SerialIO which represents one serial port.
* @param parameters parameters for the serial port being opened.
* @return SerialIO object.
* @throws NoSuchPortException if no such port is found.
* @throws PortInUseException if the serial port is already in use.
*/
public void SerialIO(SerialParameters parameters){
return null; // TODO
}

/**
* Writes an ASCII format message to serial port.
* @param message message to be send
* @throws NoSuchPortException if no such port is found.
* @throws PortInUseException if serial port is already in use.
*/
public void writeMessage(String message){
return null; // TODO
}

/**
* Writes an ASCII format message to serial port. SerialIO sends and SerialPortEvent if it
gets answer to this message.
* @return last answer received from serial port or null if no last message is available.
*/
public String getLastAnswer(){
return null; // TODO
}

/**
* This method is run when a serial message is received from serial port. It generates a
new SerialIOEvent.
*/
private void serialEvent(SerialPortEvent event){
return null; // TODO
}
```

5.4.2 SerialParameters

```
/*
* SerialParameters.java
*
* Copyright (C) 2005 Project SQUID, http://www.cs.helsinki.fi/group/squid/
*
* This file is part of Ikayaki.
*
* Ikayaki is free software; you can redistribute it and/or modify
* it under the terms of the GNU General Public License as published by
* the Free Software Foundation; either version 2 of the License, or
* (at your option) any later version.
```

```
*
* Ikayaki is distributed in the hope that it will be useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
* GNU General Public License for more details.
*
* You should have received a copy of the GNU General Public License
* along with Ikayaki; if not, write to the Free Software
* Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
*/
package ikayaki.squid;

/**
* Contains all the serial communication parameters which SerialIO uses when opening
the port.
*
* @author
*/
public class SerialParameters
/**
* The name of the serial port.
*/
private String portName;

/**
* The baud rate.
*/
private int baudRate;

/**
* Type of flow control for receiving.
*/
private int flowControlIn;

/**
* Type of flow control for sending.
*/
private int flowControlOut;

/**
* The number of data bits.
*/
private int databits;
```

```
/**
* The number of stop bits.
*/
private int stopbits;

/**
* The type of parity.
*/
private int parity;

/**
* Creates a SerialParameter object containing settings for serial port communication.
* @return Parameter settings object.
* @param portName The name of the serial port.
* @param baudRate The baud rate.
* @param flowControlIn Type of flow control for receiving.
* @param flowControlOut Type of flow control for sending.
* @param databits The number of data bits.
* @param stopbits The number of stop bits.
* @param parity The type of parity.
*/
public SerialParameters(String portName, int baudRate, int flowControlIn, int flowControlOut,
int databits, int stopbits, int parity){
return null; // TODO
}
```

5.4.3 SerialIOEvent

```
/*
* SerialIOEvent.java
*
* Copyright (C) 2005 Project SQUID, http://www.cs.helsinki.fi/group/squid/
*
* This file is part of Ikayaki.
*
* Ikayaki is free software; you can redistribute it and/or modify
* it under the terms of the GNU General Public License as published by
* the Free Software Foundation; either version 2 of the License, or
* (at your option) any later version.
*
* Ikayaki is distributed in the hope that it will be useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
* GNU General Public License for more details.
*
* You should have received a copy of the GNU General Public License
* along with Ikayaki; if not, write to the Free Software
* Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
*/
package ikayaki.squid;

/**
* An event that is generated when SerialIO receives data from serial port.
*
* @author
*/
public class SerialIOEvent extends java.util.EventObject
/**
* ASCII message recieved from serial port.
*/
private String message;


/**
* Returns received serial message.
* @return The message in ASCII form that was received from serial port.
*/
public String getMessage(){
return null; // TODO
}
```

5.4.4 SerialIOListener

```
/*
* SerialIOListener.java
*
* Copyright (C) 2005 Project SQUID, http://www.cs.helsinki.fi/group/squid/
*
* This file is part of Ikayaki.
*
* Ikayaki is free software; you can redistribute it and/or modify
* it under the terms of the GNU General Public License as published by
* the Free Software Foundation; either version 2 of the License, or
* (at your option) any later version.
*
```

```
* Ikayaki is distributed in the hope that it will be useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
* GNU General Public License for more details.
*
* You should have received a copy of the GNU General Public License
* along with Ikayaki; if not, write to the Free Software
* Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
*/
package ikayaki.squid;

/**
* If a class wants to receive SerialIOEvents it must implement this interface.
*
* @author
*/
public interface SerialIOListener extends java.util.EventListener
/**
* Propagates serial port message event.
* @param event the event that happened.
*/
public void serialIOEvent(SerialIOEvent event) {
return null; // TODO
}
```

5.5 Global settings

5.5.1 Settings

```
/*
* Settings.java
*
* Copyright (C) 2005 Project SQUID, http://www.cs.helsinki.fi/group/squid/
*
* This file is part of Ikayaki.
*
* Ikayaki is free software; you can redistribute it and/or modify
* it under the terms of the GNU General Public License as published by
* the Free Software Foundation; either version 2 of the License, or
* (at your option) any later version.
*
* Ikayaki is distributed in the hope that it will be useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
* GNU General Public License for more details.
*
* You should have received a copy of the GNU General Public License
* along with Ikayaki; if not, write to the Free Software
* Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
*/
package ikayaki;

/**
*  Singleton class for holding all global settings. All changes are automatically written to
file after a short delay.
*
* @author
*/
public class Settings
/**
* All properties in a map. Keys are: magnetometerPort(String), demagnetizerPort(String),
PorthandlerPort(String), xAxisCalibration(double), yAxisCalibration(double), zAxisCalibration(double)
demagRamp(int), demagDelay(int), acceleration(int), deceleration(int), velocity(int), measurementVeloc
transverseYAFPosition(int), axialAFPosition(int), sampleLoadPosition(int), backgroundPosition(int),
measurementPosition(int), rotation(int), handlerRightLimit(boolean)
*/
private Properties properties = new Properties();

/**
* File where the properties will be saved in XML format
*/
private File propertiesFile;

/**
* true if the properties have been modified, otherwise false
*/
private boolean propertiesModified;

/**
* All saved sequences
*/
private List<MeasurementSequence> sequences = new ArrayList<MeasurementSequence>();

/**
* File where the sequences will be saved in XML format
*/
```

```
private File sequencesFile;

/**
* true if the sequences have been modified, otherwise false
*/
private bool sequencesModified;

/**
* Queue for scheduling save operations after properties/sequences have been changed
*/
private LastExecutor autosaveQueue;

/**
/**
*  Returns the global Settings object. If not yet created, will first create one.
*/
*/
public static Settings instance(){
return null; // TODO
}

/**
*  Creates a new Settings instance. Loads settings from the configuration files.
*/
private Settings(){
return null; // TODO
}

/**
*  Saves the settings after a while when no changes have come.  The method call will
return immediately and will not wait for the file to be written.
*/
public void save(){
return null; // TODO
}

/**
*  Saves the settings and keeps waiting until its done. If no settings have been modified,
will do nothing.
*/
public void saveNow(){
return null; // TODO
}
```

```
/**
*  Returns the value that maps to the specified key.
* @param key key whose associated value is to be returned.
* @return Value associated with key, or an empty string if none exists.
*/
private String getProperty(String key){
return null; // TODO
}

/**
*  Associates the specified value with the specified key. Will invoke autosaving.
* @param key key with which the specified value is to be associated.
* @param value value to be associated with the specified key.
*/
private void setProperty(String key, String value){
return null; // TODO
}

/**
*  Generic accessor for all properties.  Returns the value from Properties in appropriate
type.
* @return Value associated with key
*/
public Type getXXX(){
return null; // TODO
}

/**
*  Generic accessor for all properties.  Checks whether the value is ok and sets it.  Will
invoke autosaving.
* @return true if value was correct, otherwise false.
*/
public boolean setXXX(Type value){
return null; // TODO
}

/**
*  Returns all saved Sequences.
*/
public MeasurementSequence[] getSequences(){
return null; // TODO
}
```

```
/**
*  Adds a sequence to the sequence list.
*/
public void addSequence(MeasurementSequence sequence){
return null; // TODO
}

/**
*  Removes a sequence from the sequence list. If the specified sequence is not in the list,
it will be ignored.
*/
public void removeSequence(MeasurementSequence sequence){
return null; // TODO
}
```

5.6 Utilities

5.6.1 LastExecutor

```
/*
* LastExecutor.java
*
* Copyright (C) 2005 Project SQUID, http://www.cs.helsinki.fi/group/squid/
*
* This file is part of Ikayaki.
*
* Ikayaki is free software; you can redistribute it and/or modify
* it under the terms of the GNU General Public License as published by
* the Free Software Foundation; either version 2 of the License, or
* (at your option) any later version.
*
* Ikayaki is distributed in the hope that it will be useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
* GNU General Public License for more details.
*
* You should have received a copy of the GNU General Public License
* along with Ikayaki; if not, write to the Free Software
* Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
*/
package ikayaki.util;
```

```
/**
*  Executes the last Runnable tasks of a series of tasks after a delay. The worker thread will terminate automatically when there are no runnables to be executed. Optionally executes all of the tasks and not only the last one. All operations are thread-safe.
This class can be used for example in connection with a "continuous search" invoked by a series of GUI events (such as a DocumentListener), but it is necessary to react to only the last event after a short period of user inactivity.
*
* @author
*/
public class LastExecutor   implements Executor
/**
*  Defines how long is the delay in milliseconds, after which the events need to be run.
*/
private int delayMillis = 0;

/**
*  Defines if only the last event should be executed.  If false, then all of the events are executed in the order of appearance.
*/
private boolean execOnlyLast = false;

/**
*  Prioritized FIFO queue for containing the RunDelayed items that have not expired. If execOnlyLast is true, then this queue should never contain more than one item.
*/
private DelayQueue<RunDelayed> queue = new DelayQueue<RunDelayed>();

/**
*  The worker thread that will run the inserted runnables. If the thread has no more work to do, it will set workerThread to null and terminate itself.
*/
private Thread workerThread = null;

/**
*  Creates an empty LastExecutor with a delay of 0 and execOnlyLast set to true.
*/
public LastExecutor(){
return null; // TODO
}

/**
```

```
*  Creates an empty LastExecutor with execOnlyLast set to true.
* @param delayMillis the length of execution delay in milliseconds; if less than 0, then 0 will be used.
*/
public LastExecutor(int delayMillis){
return null; // TODO
}

/**
*  Creates an empty LastExecutor with a delay of 0.
* @param execOnlyLast if true, only the last event will be executed after the delay; otherwise all are executed in order of appearance.
*/
public LastExecutor(boolean execOnlyLast){
return null; // TODO
}

/**
*  Creates an empty LastExecutor.
* @param delayMillis the length of execution delay in milliseconds; if less than 0, then 0 will be used.
* @param execOnlyLast if true, only the last event will be executed after the delay; otherwise all are executed in order of appearance.
*/
public LastExecutor(int delayMillis, boolean execOnlyLast){
return null; // TODO
}

/**
* @return true if only the last task will be executed after the delay; otherwise false.
*/
public synchronized boolean isExecOnlyLast(){
return null; // TODO
}

/**
* @param execOnlyLast if true, only the last task will be executed after the delay; otherwise all are executed in order of appearance.
*/
public synchronized void setExecOnlyLast(boolean execOnlyLast){
return null; // TODO
}
```

```
/**
* @return the delay in milliseconds.
*/
public synchronized int getDelayMillis(){
return null; // TODO
}

/**
* @param delayMillis delay in milliseconds; if less than 0, then the new value is ignored.
*/
public synchronized void setDelayMillis(int delayMillis){
return null; // TODO
}

/**
*  Inserts a Runnable object to the end of the queue.  It will remain there until it is
executed or another object replaces it.  If execOnlyLast is set to true, the queue will be
cleared before inserting this runnable to it.  If there is no worker thread running, a new
one will be spawned.
* @param command the runnable task to be executed after a pre-defined delay.
* @throws NullPointerException if command is null.
*/
public synchronized void execute(Runnable command){
return null; // TODO
}

/**
*  Waits for the queue to become empty.
* @throws InterruptedException if another thread has interrupted the current thread. The
interrupted status of the current thread is cleared when this exception is thrown.
*/
public synchronized void join(){
return null; // TODO
}
```

5.6.2 LastExecutor.LastExecutorThread

```
/*
* LastExecutor.LastExecutorThread.java
*
* Copyright (C) 2005 Project SQUID, http://www.cs.helsinki.fi/group/squid/
*
```

```
* This file is part of Ikayaki.
*
* Ikayaki is free software; you can redistribute it and/or modify
* it under the terms of the GNU General Public License as published by
* the Free Software Foundation; either version 2 of the License, or
* (at your option) any later version.
*
* Ikayaki is distributed in the hope that it will be useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
* GNU General Public License for more details.
*
* You should have received a copy of the GNU General Public License
* along with Ikayaki; if not, write to the Free Software
* Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
*/
package ikayaki.util;

/**
*  Keeps on checking the LastExecutor.queue to see if there are Runnables to be executed. If there is one, execute it and proceed to the next one. If an uncaught Throwable is thrown during the execution, prints an error message and stack trace to stderr.  If the queue is empty, this thread will set LastExecutor.workerThread to null and terminate itself.
*
* @author
*/
private class LastExecutorThread   extends Thread
/**
*/
public void run(){
return null; // TODO
}
```

5.6.3 LastExecutor.RunDelayed

```
/*
* LastExecutor.RunDelayed.java
*
* Copyright (C) 2005 Project SQUID, http://www.cs.helsinki.fi/group/squid/
*
* This file is part of Ikayaki.
*
* Ikayaki is free software; you can redistribute it and/or modify
* it under the terms of the GNU General Public License as published by
* the Free Software Foundation; either version 2 of the License, or
* (at your option) any later version.
*
* Ikayaki is distributed in the hope that it will be useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
* GNU General Public License for more details.
*
* You should have received a copy of the GNU General Public License
* along with Ikayaki; if not, write to the Free Software
* Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
*/
package ikayaki.util;

/**
*  Wraps a Runnable object and sets the delay after which it should be executed by a worker thread.
*
* @author
*/
private class RunDelayed   implements Delayed
/**
*  The point in time when this RunDelayed will expire.
*/
private long expires;

/**
*  Contained Runnable object to be run after this RunDelayed has expired.
*/
private Runnable runnable;

/**
*  Creates a new RunDelayed item that contains runnable.
* @param runnable the Runnable to be contained
* @param delayMillis delay in milliseconds
*/
public RunDelayed(Runnable runnable, int delayMillis){
return null; // TODO
}

/**
*  Returns the remaining delay associated with this object, always in milliseconds.
* @param unit ignored; always assumed TimeUnit.MILLISECONDS
* @return the remaining delay; zero or negative values indicate that the delay has already
elapsed
*/
public long getDelay(TimeUnit unit){
return null; // TODO
}

/**
*  Returns the contained Runnable.
* @return the Runnable given as constructor parameter
*/
public Runnable getRunnable(){
return null; // TODO
}

/**
*  Compares this object with the specified object for order.  Returns a negative integer,
zero, or a positive integer as this object is less than, equal to, or greater than the specified
object.
* @param delayed the Delayed to be compared.
* @return a negative integer, zero, or a positive integer as this delay is less than, equal to,
or greater than the specified delay.
*/
public int compareTo(Delayed delayed){
return null; // TODO
}
```

6 GUI classes and methods

6.1 Generic GUI components

6.1.1 ProjectComponent

```
/*
* ProjectComponent.java
*
* Copyright (C) 2005 Project SQUID, http://www.cs.helsinki.fi/group/squid/
*
* This file is part of Ikayaki.
*
* Ikayaki is free software; you can redistribute it and/or modify
* it under the terms of the GNU General Public License as published by
* the Free Software Foundation; either version 2 of the License, or
* (at your option) any later version.
*
* Ikayaki is distributed in the hope that it will be useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
* GNU General Public License for more details.
*
* You should have received a copy of the GNU General Public License
* along with Ikayaki; if not, write to the Free Software
* Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
*/
package ikayaki.gui;

/**
*  Common superclass for components which use a Project and listen to MeasurementEvents
and ProjectEvents.
*
* @author
*/
public class ProjectComponent   extends JPanel
/*
Event A: On ProjectEvent - does nothing; subclasses may override this.
*/

/*
Event B: On MeasurementEvent - does nothing; subclasses may override this.
*/


/**
* The active project.
*/
private Project project;


/**
* Initializes this ProjectComponent with no project.
*/
public ProjectComponent(){
return null; // TODO
}


/**
*  Returns the active project, or null if no project is active.
```

```
*/
public Project getProject(){
return null; // TODO
}

/**
* Sets the project for this ProjectComponent. Unregisters MeasurementListener and
ProjectListener from the old project, and registers them to the new project.
* @param project new active project, or null to make no project active.
*/
public void setProject(Project project){
return null; // TODO
}

/**
* Does nothing; subclasses override this if they want to listen ProjectEvents.
* @param event ProjectEvent received.
*/
public void projectUpdated(ProjectEvent event){
return null; // TODO
}

/**
* Does nothing; subclasses override this if they want to listen MeasurementEvents.
* @param event MeasurementEvent received.
*/
public void measurementUpdated(MeasurementEvent event){
return null; // TODO
}
```

6.2 Main window

6.2.1 Ikayaki

```
/*
* Ikayaki.java
*
* Copyright (C) 2005 Project SQUID, http://www.cs.helsinki.fi/group/squid/
*
* This file is part of Ikayaki.
*
* Ikayaki is free software; you can redistribute it and/or modify
* it under the terms of the GNU General Public License as published by
* the Free Software Foundation; either version 2 of the License, or
* (at your option) any later version.
*
* Ikayaki is distributed in the hope that it will be useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
* GNU General Public License for more details.
*
* You should have received a copy of the GNU General Public License
* along with Ikayaki; if not, write to the Free Software
* Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
*/
package ikayaki.gui;

/**
*  Starts the program.  Lays out MainViewPanel, MainMenuBar and MainStatusBar in a JFrame.
*
* @author
*/
public class Ikayaki   extends JFrame
/*
Event A: On window close - checks that no measurement is running.  Saves all opened project files and settings. Closes the program, or notifies the user if the program may not be closed.
*/


/**
*  Starts the program with the provided command line parameters.  If the location of a project file is given as a parameter, the program will try to load it.
* @param args command line parameters.
*/
public static void main(String[] args){
return null; // TODO
}
```

6.2.2 MainViewPanel

```
/*
* MainViewPanel.java
*
* Copyright (C) 2005 Project SQUID, http://www.cs.helsinki.fi/group/squid/
```

```
*
* This file is part of Ikayaki.
*
* Ikayaki is free software; you can redistribute it and/or modify
* it under the terms of the GNU General Public License as published by
* the Free Software Foundation; either version 2 of the License, or
* (at your option) any later version.
*
* Ikayaki is distributed in the hope that it will be useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
* GNU General Public License for more details.
*
* You should have received a copy of the GNU General Public License
* along with Ikayaki; if not, write to the Free Software
* Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
*/
package ikayaki.gui;

/**
*  Creates the main view panels (split panels) and Squid and Project components. It also
tells everybody if the current project is changed.
*
* @author
*/
public class MainViewPanel   extends JPanel
private ProjectExplorerPanel projectExplorer;

private CalibrationPanel calibration;

private Squid squid;

/**
* Currently opened project.
*/
private Project project;

/**
* Project which has an ongoing measurement, or null if no measurement is running.
*/
private Project measuringProject;

private MainMenuBar menuBar;
private MainStatusBar statusBar;

private ProjectInformationPanel projectInformation;

private MeasurementSequencePanel measurementSequence;

private MeasurementControlsPanel measurementControls;

private MeasurementDetailsPanel measurementDetails;

private MeasurementGraphsPanel measurementGraphs;

/**
*  Loads default view and creates all components and panels.  Splitpanel between Calibration,Explorer,Inf
and rest.
*/
public MainViewPanel(){
return null; // TODO
}

/**
*  Looks for file with filename, if not exist creates new other wise opens it. Then updates
current project and tells Panels new project is opened.
*/
public boolean changeProject(Project project){
return null; // TODO
}
```

6.2.3 MainMenuBar

```
/*
* MainMenuBar.java
*
* Copyright (C) 2005 Project SQUID, http://www.cs.helsinki.fi/group/squid/
*
* This file is part of Ikayaki.
*
* Ikayaki is free software; you can redistribute it and/or modify
* it under the terms of the GNU General Public License as published by
* the Free Software Foundation; either version 2 of the License, or
* (at your option) any later version.
*
* Ikayaki is distributed in the hope that it will be useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
* GNU General Public License for more details.
*
* You should have received a copy of the GNU General Public License
* along with Ikayaki; if not, write to the Free Software
* Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
*/
package ikayaki.gui;

/**
* Creates Menu items for Menubar and makes action listeners for them
*
* @author
*/
public class MainMenuBar   extends JMenuBar
/*
Event A: On newProject Clicked - Opens File chooser and opens new file in selected
folder
*/

/*
Event B: On openProject Clicked - Opens File chooser and opens selected file
*/

/*
Event C: On exportToDAT Clicked - Opens File chooser and tells Project to export in
selected file
*/

/*
Event D: On exportToTDT Clicked - Opens File chooser and tells Project to export in
selected file
*/

/*
Event E: On exportToSRM Clicked - Opens File chooser and tells Project to export in
selected file
*/

/*
Event F: On configuration Clicked - Opens SettingsPanel (frame)
```

```
*/

/*
Event G: On helpItem Clicked - Opens Help dialog (own frame?)
*/

/*
Event H: On about Clicked - Opens dialog with credits and version number
*/

/*
Event I: On exit Clicked - closes program
*/

private JMenu file;

private JMenu options;

private JMenu help;

private Action newProject;

private Action openProject;

private JMenu exportProject;

private Action exportProjectToDAT;

private Action exportProjectToDTD;

private Action exportProjectToSRM;

private Action exit;

private Action configuration;

private Action helpItem;

private Action about;
```

```
/**
*  Creates all components and makes menu and sets ActionListeners.
*/
public MainMenuBar(){
return null; // TODO
}
```

6.2.4 MainStatusBar

```
/*
* MainStatusBar.java
*
* Copyright (C) 2005 Project SQUID, http://www.cs.helsinki.fi/group/squid/
*
* This file is part of Ikayaki.
*
* Ikayaki is free software; you can redistribute it and/or modify
* it under the terms of the GNU General Public License as published by
* the Free Software Foundation; either version 2 of the License, or
* (at your option) any later version.
*
* Ikayaki is distributed in the hope that it will be useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
* GNU General Public License for more details.
*
* You should have received a copy of the GNU General Public License
* along with Ikayaki; if not, write to the Free Software
* Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
*/
package ikayaki.gui;

/**
* Creates its components and listens project events on status change and calculates estimated time for measurement
*
* @author
*/
public class MainStatusBar   extends ProjectComponent
/*
Event A: On Measurement Event - recalculates progress and updates status for current measurement
*/
```

```
/**
* text comment of current status(moving,measurement,demagnetization)
*/
private JLabel measurementStatus;

/**
* progress of sequence/measurement as per cent of whole process
*/
private JProgressBar measurementProgress;

/**
* current projects sequence
*/
private int[] currentSequence;

/**
* current projects type (we know if we are doing demagnetization or not)
*/
private int projectType;

/**
*  Creates all components with default settings and sets Listener for MeasurementEvent.
*/
public MainStatusBar(){
return null; // TODO
}

/**
*  Recalculates current progress and updates status.
*/
private void calculateStatus(String phase, int sequenceStep, int currentStep){
return null; // TODO
}

/**
*  Formats status and creates new measurement status values.
*/
private void setMeasurement(int projectType, int[] sequence){
return null; // TODO
}
```

6.3 Configuration window

6.3.1 SettingsPanel

```
/*
* SettingsPanel.java
*
* Copyright (C) 2005 Project SQUID, http://www.cs.helsinki.fi/group/squid/
*
* This file is part of Ikayaki.
*
* Ikayaki is free software; you can redistribute it and/or modify
* it under the terms of the GNU General Public License as published by
* the Free Software Foundation; either version 2 of the License, or
* (at your option) any later version.
*
* Ikayaki is distributed in the hope that it will be useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
* GNU General Public License for more details.
*
* You should have received a copy of the GNU General Public License
* along with Ikayaki; if not, write to the Free Software
* Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
*/
package ikayaki.gui;

/**
* Creates its components and updats changes to Settings and saves them in Configuration
file
*
* @author
*/
public class SettingsPanel   extends JFrame
/*
Event A: On Save Clicked - saves current configuration to Settings-singleton and closes
window
*/

/*
Event B: On Cancel Clicked - closes window (discarding changes)
*/


/**
* COM port for magnetometer
```

```
*/
private JComboBox magnetometerPort;

/**
* COM port for demagnetizer, can be sharing same port with magnetometer
*/
private JComboBox demagnetizerPort;

/**
* COM port for sample handler
*/
private JComboBox handlerPort;

/**
* Calibration constants with polarization (factory set?)
*/
private JTextField xAxisCalibration;

/**
* Calibration constants with polarization (factory set?)
*/
private JTextField yAxisCalibration;

/**
* Calibration constants with polarization (factory set?)
*/
private JTextField zAxisCalibration;

/**
* how fast demagnetization goes
*/
private JComboBox demagRamp;

/**
* ?
*/
private JComboBox demagDelay;

/**
* Handler acceleration
*/
private JTextField acceleration;
```

```
/**
* Handler deceleration
*/
private JTextField deceleration;

/**
* Handler Max speed
*/
private JTextField velocity;

/**
* speed in measurement, should be small
*/
private JTextField measurementVelocity;

/**
* AF demag position for transverse
*/
private JTextField transverseYAFPosition;

/**
* axial AF demag position in steps, must be divisible by 10. Relative to Home.
*/
private JTextField axialAFPosition;

/**
* Position in steps, must be divisible by 10. Relative to Home. (same as Home?)
*/
private JTextField sampleLoadPosition;

/**
* Position in steps, must be divisible by 10. Relative to Home.
*/
private JTextField backgroundPosition;

/**
* Position in steps, must be divisible by 10. Relative to Home.
*/
private JTextField measurementPosition;

/**
```

```
* steps to perform full rotation, must be clockwise, determined by sign
*/
private JTextField rotation;

/**
* Refers to right limit switch on translation axis.  And usually sample holder motion
toward right limit is posivitive direction (default).
*/
private JComboBox handlerRightLimit;

private JButton saveButton;

private JButton cancelButton;

/**
*  Creates all components and puts them in right places. Labels are created only here (no
global fields). Creates ActionListeners for buttons.
*/
public SettingsPanel(){
return null; // TODO
}

/**
*  Closes window, no changes saved.
*/
public void closeWindow(){
return null; // TODO
}

/**
*  Saves all settings to Settings-singleton and calls closeWindow().
*/
public void saveSettings(){
return null; // TODO
}
```

6.4 Project Explorer

6.4.1 ProjectExplorerPanel

```
/*
* ProjectExplorerPanel.java
*
* Copyright (C) 2005 Project SQUID, http://www.cs.helsinki.fi/group/squid/
*
* This file is part of Ikayaki.
*
* Ikayaki is free software; you can redistribute it and/or modify
* it under the terms of the GNU General Public License as published by
* the Free Software Foundation; either version 2 of the License, or
* (at your option) any later version.
*
* Ikayaki is distributed in the hope that it will be useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
* GNU General Public License for more details.
*
* You should have received a copy of the GNU General Public License
* along with Ikayaki; if not, write to the Free Software
* Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
*/
package ikayaki.gui;

/**
*  Creates a history/autocomplete field (browserField) for choosing the project directory, a
listing of project files in that directory (explorerTable) and in that listing a line for creating
new project, which has a textbox for project name, an AF/TH ComboBox and a "Create
new" button (createNewProjectButton) for actuating the creation.  Also has a right-click
popup menu for exporting project files.
*
* @author
*/
public class ProjectExplorerPanel   extends ProjectComponent
/*
Event A: On browserField change - send autocomplete-results-finder with browserField's
text to LastExecutor via lastExecutor.execute(Runnable), which schedules disk access and
displaying autocomplete results in browserField's popup window.
*/

/*
Event B: On browserField down-arrow-click - show directory history in browserField's
```

```
popup window.
*/

/*
Event C: On browserField popup window click - set clicked line as directory, update files
listing, update explorerTable and browserField.
*/

/*
Event D: On browseButton click - open a FileChooser dialog for choosing new directory,
set it to directory, update files listing, update explorerTable and browserField.
*/

/*
Event E: On ProjectEvent - hilight project whose measuring started, or unhilight one
whose measuring ended.
*/

/**
* Text field for writing directory to change to.  Autocomplete results appear to Combo
Box’ popup window, scheduled by LastExecutor.  Directory history appears to the same
popup window when the down-arrow right to text field is clicked.
*/
private JComboBox browserField;

private JButton browseButton;

private ProjectExplorerTable explorerTable;

private NewProjectPanel newProjectPanel;

/**
* LastExecutor for scheduling autocomplete results to separate thread (disk access and
displaying).
*/
private LastExecutor autocompleteExecutor = new LastExecutor(100, true);

/**
* Currently open directory.
*/
private File directory = null;
```

```
/**
* Project files in current directory.
*/
private Vector<File> files = new Vector<File>();

/**
* Creates all components, sets directory as the last open directory, initializes files with
files from that directory.
*/
public ProjectExplorerPanel(Project project){
return null; // TODO
}

/**
* Call super.setProject(project), hilight selected project, or unhilight unselected project.
*/
public void setProject(Project project){
return null; // TODO
}
```

6.4.2 NewProjectPanel

```
/*
* NewProjectPanel.java
*
* Copyright (C) 2005 Project SQUID, http://www.cs.helsinki.fi/group/squid/
*
* This file is part of Ikayaki.
*
* Ikayaki is free software; you can redistribute it and/or modify
* it under the terms of the GNU General Public License as published by
* the Free Software Foundation; either version 2 of the License, or
* (at your option) any later version.
*
* Ikayaki is distributed in the hope that it will be useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
* GNU General Public License for more details.
*
* You should have received a copy of the GNU General Public License
* along with Ikayaki; if not, write to the Free Software
* Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
*/
```

```
package ikayaki.gui;

/**
*  Panel with components for creating a new project. This Panel will be somewhere below
the project file listing...
*
* @author
*/
public class NewProjectPanel   extends JPanel
/*
Event A: On createNewProjectButton click - call Project.createXXXProject(File) with
filename from newProjectField; if returns null, show error message and do nothing. Otherwise,
update file listing, set new project active, tell explorerTable to reset newProjectField and
newProjectType and call MainViewPanel.changeProject(Project) with returned Project.
*/


private JTextField newProject;


private JComboBox newProjectType = AF/Thellier/Thermal;


private JButton createNewProjectButton;
```

6.4.3 ProjectExplorerTable

```
/*
* ProjectExplorerTable.java
*
* Copyright (C) 2005 Project SQUID, http://www.cs.helsinki.fi/group/squid/
*
* This file is part of Ikayaki.
*
* Ikayaki is free software; you can redistribute it and/or modify
* it under the terms of the GNU General Public License as published by
* the Free Software Foundation; either version 2 of the License, or
* (at your option) any later version.
*
* Ikayaki is distributed in the hope that it will be useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
* GNU General Public License for more details.
*
* You should have received a copy of the GNU General Public License
* along with Ikayaki; if not, write to the Free Software
* Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
*/
package ikayaki.gui;

/**
*  Creates a list of project files in directory. Handles loading selected projects and showing
export popup menu (ProjectExplorerPopupMenu). Inner class of ProjectExplorerPanel.
*
* @author
*/
public class ProjectExplorerTable   extends JTable
/*
Event A: On table click - call Project.loadProject(File) with clicked project file, call
MainViewPanel.changeProject(Project) with returned Project unless null, on which case
show error message and revert explorerTable selection to old project, if any.
*/

/*
Event B: On table mouse right-click - create a ProjectExplorerPopupMenu for right-
clicked project file.
*/


/**
* TableModel which handles data from files (in upper-class ProjectExplorerPanel). Unnamed
inner class.
*/
private TableModel projectExplorerTableModel;
```

6.4.4 ProjectExplorerPopupMenu

```
/*
* ProjectExplorerPopupMenu.java
*
* Copyright (C) 2005 Project SQUID, http://www.cs.helsinki.fi/group/squid/
*
* This file is part of Ikayaki.
*
* Ikayaki is free software; you can redistribute it and/or modify
* it under the terms of the GNU General Public License as published by
* the Free Software Foundation; either version 2 of the License, or
* (at your option) any later version.
*
```

```
* Ikayaki is distributed in the hope that it will be useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
* GNU General Public License for more details.
*
* You should have received a copy of the GNU General Public License
* along with Ikayaki; if not, write to the Free Software
* Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
*/
package ikayaki.gui;

/**
*  Shows popup menu with export choices: AF (.dat), Thellier (.tdt) and Thermal (.tdt), and for each, "to current directory", "to disk drive A:" and "...", which opens a standard file chooser for selecting dir and file to export to. Executes selected export command.
*
* @author
*/
public class ProjectExplorerPopupMenu   extends JPopupMenu
/*
Event A: On menu click - call project.exportToXXX(File) according to selected menu item; if false is returned, show error message.
*/
```

6.5 Calibration

6.5.1 CalibrationPanel

```
/*
* CalibrationPanel.java
*
* Copyright (C) 2005 Project SQUID, http://www.cs.helsinki.fi/group/squid/
*
* This file is part of Ikayaki.
*
* Ikayaki is free software; you can redistribute it and/or modify
* it under the terms of the GNU General Public License as published by
* the Free Software Foundation; either version 2 of the License, or
* (at your option) any later version.
*
* Ikayaki is distributed in the hope that it will be useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
* GNU General Public License for more details.
*
* You should have received a copy of the GNU General Public License
* along with Ikayaki; if not, write to the Free Software
* Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
*/
package ikayaki.gui;

/**
*  Holds predefined "Holder noise" and "Standard sample" projects for calibration; they
are in a technically same table as Project explorer files.  Also has a "Calibrate" button,
which executes selected calibration project, similarly to clicking "Single step" in normal
projects.
*
* @author
*/
public class CalibrationPanel   extends ProjectComponent
/*
Event A: On calibrateButton click - call project.doSingleStep(); show error message if
false is returned.
*/

/*
Event B: On calibrationProjectTable click - call Project.loadProject(File) with clicked
project file (calibrationProjectTable row); call MainViewPanel.changeProject(Project) with
returned Project unless null, on which case show error message and revert calibrationProjectTable
selection to old project, if any.
*/

/*
Event C: On ProjectEvent - highlight calibration project whose measuring started, or
unhighlight one whose measuring ended; enable calibrateButton if measuring has ended,
or disable if measuring has started.
*/

private JButton calibrateButton;

/**
* Table for the two calibration projects; has "filename", "last modified" and "time" (time
since last modification) columns.
*/
private JTable calibrationProjectTable;

/**
```

```
* TableModel which holds the data for calibration projects. Unnamed inner class.
*/
private TableModel calibrationProjectTableModel;

/**
* Call super.setProject(project), highlight selected calibration project, or unhighlight unselected
calibration project.
*/
public void setProject(Project project){
return null; // TODO
}
```

6.6 Project information

6.6.1 ProjectInformationPanel

```
/*
* ProjectInformationPanel.java
*
* Copyright (C) 2005 Project SQUID, http://www.cs.helsinki.fi/group/squid/
*
* This file is part of Ikayaki.
*
* Ikayaki is free software; you can redistribute it and/or modify
* it under the terms of the GNU General Public License as published by
* the Free Software Foundation; either version 2 of the License, or
* (at your option) any later version.
*
* Ikayaki is distributed in the hope that it will be useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
* GNU General Public License for more details.
*
* You should have received a copy of the GNU General Public License
* along with Ikayaki; if not, write to the Free Software
* Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
*/
package ikayaki.gui;

/**
* Allows inserting and editing project information.
*
* @author
```

```
*/
public class ProjectInformationPanel   extends ProjectComponent
/*
Event A: On change of contest in textfield - Notify project about change in project information.
*/

/*
Event B: On project event - Update textfields to correspond new project information.
*/

private JLabel operatorLabel;

private JTextFiels operatorTextField;

private JLabel dateLabel;

private JTextFiels dateTextField;

/**
* Groups autoMeasurement and manualMeasurement radiobuttons.
*/
private ButtonGroup measurementType;

private JRadioButton autoMeasurement;

private JRadioButton manualMeasurement;

private JLabel rocktypeLabel;

private JTextFiels rocktypeTextField;

private JLabel siteLabel;

private JTextFiels siteTextField;

private JLabel commentLabel;

private JTextFiels commentTextField;

private JLabel latitudeLabel;
private JTextFiels latitudeTextField;

private JLabel longLabel;

private JTextFiels longTextField;

private JLabel strikeLabel;

private JTextFiels strikeTextField;

private JLabel dipLabel;

private JTextFiels dipTextField;

private JLabel volumeLabel;

private JTextFiels volumeTextField;

private JLabel massLabel;

private JTextFiels massTextField;

/**
* Groups coreSample and handSample radiobuttons.
*/
private ButtonGroup sampleType;

private JRadioButton coreSample;

private JRadioButton handSample;

/**
* Creates default ProjectInformationPanel.
*/
public ProjectInformationPanel(){
return null; // TODO
}
```

```
/**
* Calls super.setProject(project) and updates textfield with new projects data.
*/
private void setProject(Project project){
return null; // TODO
}
```

6.7 Sequence and measurement data

6.7.1 MeasurementSequencePanel

```
/*
* MeasurementSequencePanel.java
*
* Copyright (C) 2005 Project SQUID, http://www.cs.helsinki.fi/group/squid/
*
* This file is part of Ikayaki.
*
* Ikayaki is free software; you can redistribute it and/or modify
* it under the terms of the GNU General Public License as published by
* the Free Software Foundation; either version 2 of the License, or
* (at your option) any later version.
*
* Ikayaki is distributed in the hope that it will be useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
* GNU General Public License for more details.
*
* You should have received a copy of the GNU General Public License
* along with Ikayaki; if not, write to the Free Software
* Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
*/
package ikayaki.gui;

/**
* Allows creating, editing and removing measurement sequences. Shows measurement data. Right-click brings popup menu for hiding columns, and saving sequence. Left-click selects a row. Multiple rows can be selected by ctrl-clicking or shift-clicking. Allows dragging rows to different order if multiple rows are selected multiple rows are dragged. Has three textfields for inserting new sequences, first field for start value, second for step and third for stop value. Clicking Add sequence-button appends sequence into table. Saved sequences can be loaded from dropdown menu.
*
```

```
* @author
*/
public class MeasurementSequencePanel   extends ProjectComponent
/*
Event A: On SequenceTable mouse right-click - Create a MeasurementSequencePopupMenu.
*/

/*
Event B: On addSequence mouseclick - Add measurement sequence to project class and
tell MeasurementSequenceTableModel to update itself.
*/

/*
Event C: On sequenceSelector mouseclick - Bring dropdown menu for selecting premade
sequence.
*/

/*
Event D: On selecting sequence from dropdown menu - Add measurement sequence to
table and tell MeasurementSequenceTableModel to update itself.
*/

/*
Event E: On Project event - Update contest of table to correspond projects state.
*/

/*
Event F: On Measurement event - If measurement step is finished, get measurement
data from project class and if row being measured was selected select next row unless
measurement sequence ended.
*/

/*
Event G: On Drag event - Change measurement sequences row order in project class
and tell MeasurementSequenceTableModel to update itself to correspond new row order.
Order of rows with measurement data cannot be changed.
*/

private JButton addSequence;

private JComboBox sequenceSelector;

private JTextField sequenceStart;
private JTextField sequenceStep;

private JTextField sequenceStop;

private JTable sequenceTable;

private MeasurementSequenceTableModel tableModel;

/**
* Creates default MeasurementSequencePanel.
*/
public MeasurementSequencePanel(){
return null; // TODO
}

/**
* Adds sequence determined by textfields to end of table.
*/
private void addSequence(){
return null; // TODO
}

/**
* Calls super.setProject(project), clears table and calculates shown data from project’s
measurement data.
*/
private void setProject(Project project){
return null; // TODO
}
```

6.7.2 MeasurementSequenceTableModel

```
/*
* MeasurementSequenceTableModel.java
*
* Copyright (C) 2005 Project SQUID, http://www.cs.helsinki.fi/group/squid/
*
* This file is part of Ikayaki.
*
* Ikayaki is free software; you can redistribute it and/or modify
* it under the terms of the GNU General Public License as published by
* the Free Software Foundation; either version 2 of the License, or
* (at your option) any later version.
*
* Ikayaki is distributed in the hope that it will be useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
* GNU General Public License for more details.
*
* You should have received a copy of the GNU General Public License
* along with Ikayaki; if not, write to the Free Software
* Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
*/
package ikayaki.gui;

/**
* Handles data in table.
*
* @author
*/
public class MeasurementSequenceTableModel   extends AbstractTableModel
/**
* Tells if volume is shown in table.
*/
private boolean volume;

/**
* Creates SequenceTableModel
*/
public MeasurementSequenceTableModel(){
return null; // TODO
}

/**
* Shows named column.
* @param name name of the column to be shown. possible values VOLUME=0
*/
public void showColumn(int name){
return null; // TODO
}

/**
* Hides named column.
* @param name name of the column to be hidden. possible values VOLUME=0
```

```
*/
public void hideColumn(int name){
return null; // TODO
}
```

6.7.3 MeasurementSequencePopupMenu

```
/*
* MeasurementSequencePopupMenu.java
*
* Copyright (C) 2005 Project SQUID, http://www.cs.helsinki.fi/group/squid/
*
* This file is part of Ikayaki.
*
* Ikayaki is free software; you can redistribute it and/or modify
* it under the terms of the GNU General Public License as published by
* the Free Software Foundation; either version 2 of the License, or
* (at your option) any later version.
*
* Ikayaki is distributed in the hope that it will be useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
* GNU General Public License for more details.
*
* You should have received a copy of the GNU General Public License
* along with Ikayaki; if not, write to the Free Software
* Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
*/
package ikayaki.gui;

/**
* Allows selection if volume is shown in table and saving sequence.  Pops up when measurement sequence table is right-clicked.  Allows saving selected sequence or whole sequence.
*
* @author
*/
public class MeasurementSequencePopupMenu   extends JPopupMenu
/**
* If checked volume is shown in measurement sequence table.
*/
private JCheckBox volume;
private JLabel nameLabel;

/**
* Name of the sequence to be saved.
*/
private JCheckBox nameTextField;

/**
* Creates SequencePopupMenu.
*/
public MeasurementSequencePopupMenu(){
return null; // TODO
}

/**
* Saves whole sequence into dropdown menu.
*/
private void saveSequence(){
return null; // TODO
}

/**
* Saves selected sequence into dropdown menu.
*/
private void saveSequence(){
return null; // TODO
}

/**
* Removes selected rows. Rows with measurement data cannot be removed.
*/
private void removeRows(){
return null; // TODO
}
```

6.8 Measurement details

6.8.1 MeasurementDetailsPanel

```
/*
* MeasurementDetailsPanel.java
```

```
*
* Copyright (C) 2005 Project SQUID, http://www.cs.helsinki.fi/group/squid/
*
* This file is part of Ikayaki.
*
* Ikayaki is free software; you can redistribute it and/or modify
* it under the terms of the GNU General Public License as published by
* the Free Software Foundation; either version 2 of the License, or
* (at your option) any later version.
*
* Ikayaki is distributed in the hope that it will be useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
* GNU General Public License for more details.
*
* You should have received a copy of the GNU General Public License
* along with Ikayaki; if not, write to the Free Software
* Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
*/
package ikayaki.gui;

/**
* Shows details of measurement selected in MeasurementSequencePanel.
*
* @author
*/
public class MeasurementDetailsPanel   extends ProjectComponent
/*
Event A: On project event - Update tables to correspond projects new state.
*/

/*
Event B: On change of selected row in MeasurementSequencePanel - Change tables to
correspond selected row.
*/

/*
Event C: On measurement event - If row corresponding to ongoing measurement is selected
in MeasurementSequencePanel update tables with new measurement data.
*/


/**
* X, Y and Z components of BG1, 0, 90, 180, 270, BG2
*/
```

```
private JTable measurementDetails;

/**
* S/D, S/H and S/N of error
*/
private JTable errorDetails;

private DefaultTableModel tableMoled;

/**
* Creates default MeasurementDetailsPanel.
*/
public MeasurementDetails(){
return null; // TODO
}

/**
* Calls super.setProject(project), clears tables and shows new projects measurement details.
*/
private void setProject(Project project){
return null; // TODO
}
```

6.9 Measurement controls

6.9.1 MeasurementControlsPanel

```
/*
* MeasurementControlsPanel.java
*
* Copyright (C) 2005 Project SQUID, http://www.cs.helsinki.fi/group/squid/
*
* This file is part of Ikayaki.
*
* Ikayaki is free software; you can redistribute it and/or modify
* it under the terms of the GNU General Public License as published by
* the Free Software Foundation; either version 2 of the License, or
* (at your option) any later version.
*
* Ikayaki is distributed in the hope that it will be useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
* GNU General Public License for more details.
*
* You should have received a copy of the GNU General Public License
* along with Ikayaki; if not, write to the Free Software
* Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
*/
package ikayaki.gui;

/**
*  Has "Measure"/"Pause", "Single step" and "Stop now!" buttons for controlling measurements;
"+z/-z" radiobuttons for changing sample orientation used in calculations, help picture for
inserting sample, picture of current magnetometer status, and, manual controls.
Listens MeasurementEvents and ProjectEvents, and updates buttons and magnetometer
status accordingly.
*
* @author
*/
public class MeasurementControlsPanel   extends ProjectComponent
/*
Event A: On measureButton click - call project.doAutoStep() or project.doPause(), depending
on current button status. Show error message if false is returned.
*/

/*
Event B: On singlestepButton click - call project.doSingleStep(); show error message if
false is returned.
*/

/*
Event C: On stopButton click - call project.doAbort(); show critical error message if false
is returned.
*/

/*
Event D: On zPlus,MinusRadioButton click - call project.setOrientation(boolean) where
Plus is true and Minus is false.
*/

/*
Event E: On ProjectEvent - update buttons and manual controls according to project.isXXXEnabled().
*/

/*
Event F: On MeasurementEvent - call magnetometerStatusPanel.updateStatus(int, int)
```

```
with the right values from MeasurementEvent.
*/

/**
* Measure/pause -button; "Measure" when no measuring is being done, "Pause" when
there is ongoing measuring sequence.
*/
private JButton measureButton;

private JButton singlestepButton;

private JButton stopButton;

/**
* Groups together +z and -z RadioButtons.
*/
private ButtonGroup zButtonGroup;

/**
* Changes sample orientation to +Z.
*/
private JRadioButton zPlusRadioButton;

/**
* Changes sample orientation to -Z.
*/
private JRadioButton zMinusRadioButton;

/**
* Draws a help image and text for sample inserting: "Put sample in holder arrow up."
*/
private JPanel sampleInsertPanel;

private MagnetometerStatusPanel magnetometerStatusPanel;

private ManualControlsPanel manualControlsPanel;
```

6.9.2 MagnetometerStatusPanel

```
/*
* MagnetometerStatusPanel.java
*
* Copyright (C) 2005 Project SQUID, http://www.cs.helsinki.fi/group/squid/
*
* This file is part of Ikayaki.
*
* Ikayaki is free software; you can redistribute it and/or modify
* it under the terms of the GNU General Public License as published by
* the Free Software Foundation; either version 2 of the License, or
* (at your option) any later version.
*
* Ikayaki is distributed in the hope that it will be useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
* GNU General Public License for more details.
*
* You should have received a copy of the GNU General Public License
* along with Ikayaki; if not, write to the Free Software
* Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
*/
package ikayaki.gui;

/**
* Picture of current magnetometer status, with sample holder position and rotation. Status
is updated according to MeasurementEvents received by MeasurementControlsPanel.
*
* @author
*/
public class MagnetometerStatusPanel   extends JPanel
/**
* Sets magnetometer status to current position.
*/
public MagnetometerStatusPanel(){
return null; // TODO
}


/**
* Updates magnetometer status picture; called by MeasurementControlsPanel when it
receives MeasurementEvent.
* @param position sample holder position, from 1 to 16777215.
* @param rotation sample holder rotation, from 0 (angle 0) to 2000 (angle 360).
```

```
*/
public void updateStatus(int position, int rotation){
return null; // TODO
}
```

6.9.3 ManualControlsPanel

```
/*
* ManualControlsPanel.java
*
* Copyright (C) 2005 Project SQUID, http://www.cs.helsinki.fi/group/squid/
*
* This file is part of Ikayaki.
*
* Ikayaki is free software; you can redistribute it and/or modify
* it under the terms of the GNU General Public License as published by
* the Free Software Foundation; either version 2 of the License, or
* (at your option) any later version.
*
* Ikayaki is distributed in the hope that it will be useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
* GNU General Public License for more details.
*
* You should have received a copy of the GNU General Public License
* along with Ikayaki; if not, write to the Free Software
* Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
*/
package ikayaki.gui;

/**
* Magnetometer manual controls. MeasurementControlsPanel disables these whenever a
normal measurement step is going.
*
* @author
*/
public class ManualControlsPanel   extends JPanel
/*
Event A: On moveXXX click - call project.doManualMove(int) with clicked position.
If false is returned, show small error message. Position values are found from Settings;
demagZ is Settings.instance().getAxialAFPosition() and demagY is Settings.instance().getTransverseYA
*/
```

```
/*
Event B: On rotateXXX click - call project.doManualRotate(int) with clicked angle.  If
false is returned, show small error message.
*/

/*
Event C: On measureAllButton click - call project.doManualMeasure(). If false is returned,
show small error message.
*/

/*
Event D: On resetAllButton click - call project.doManualReset()?  If false is returned,
show small error message.
*/

/*
Event E: On DemagZButton click - call project.doManualDemagZ(double) with value
from demagAmplitudeField. If false is returned, show small error message.
*/

/*
Event F: On DemagYButton click - call project.doManualDemagY(double) with value
from demagAmplitudeField. If false is returned, show small error message.
*/


/**
* Groups together all sample holder moving RadioButtons (moveXXX).
*/
private ButtonGroup moveButtonGroup;

/**
* Moves sample holder to home position.
*/
private JRadioButton moveHome;

/**
* Moves sample holder to demagnetize-Z position.
*/
private JRadioButton moveDemagZ;

/**
* Moves sample holder to demagnetize-Y position.
*/
```

```
private JRadioButton moveDemagY;

/**
* Moves sample holder to background position.
*/
private JRadioButton moveBG;

/**
* Moves sample holder to measurement position.
*/
private JRadioButton moveMeasure;

/**
* Groups together all sample holder rotating RadioButtons (rotateXXX).
*/
private ButtonGroup rotateButtonGroup;

/**
* Rotates sample holder to angle 0.
*/
private JRadioButton rotate0;

/**
* Rotates sample holder to angle 90.
*/
private JRadioButton rotate90;

/**
* Rotates sample holder to angle 180.
*/
private JRadioButton rotate180;

/**
* Rotates sample holder to angle 270.
*/
private JRadioButton rotate270;

/**
* Measures X, Y and Z (at current sample holder position) by calling project.doManualMeasure().
*/
private JButton measureAllButton;
```

```
/**
* Resets X, Y and Z by calling project.doManualReset()? Does what?
*/
private JButton resetAllButton;

/**
* Demagnetization amplitude in mT, used when demagZ,YButton is clicked.
*/
private JTextField demagAmplitudeField;

/**
* Demagnetizes in Z (at current sample holder position) by calling project.doManualDemagZ(double).
*/
private JButton demagZButton;

/**
* Demagnetizes in Y (at current sample holder position) by calling project.doManualDemagY(double).
*/
private JButton demagYButton;
```

6.10 Graphs

6.10.1 MeasurementGraphsPanel

```
/*
* MeasurementGraphsPanel.java
*
* Copyright (C) 2005 Project SQUID, http://www.cs.helsinki.fi/group/squid/
*
* This file is part of Ikayaki.
*
* Ikayaki is free software; you can redistribute it and/or modify
* it under the terms of the GNU General Public License as published by
* the Free Software Foundation; either version 2 of the License, or
* (at your option) any later version.
*
* Ikayaki is distributed in the hope that it will be useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
* GNU General Public License for more details.
*
* You should have received a copy of the GNU General Public License
* along with Ikayaki; if not, write to the Free Software
* Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
*/
package ikayaki.gui;

/**
*
*
* @author
*/
public class MeasurementGraphsPanel extends ProjectComponent implements MeasurementListener
```

6.10.2 AbstractPlot

```
/*
* AbstractPlot.java
*
* Copyright (C) 2005 Project SQUID, http://www.cs.helsinki.fi/group/squid/
*
* This file is part of Ikayaki.
*
* Ikayaki is free software; you can redistribute it and/or modify
* it under the terms of the GNU General Public License as published by
* the Free Software Foundation; either version 2 of the License, or
* (at your option) any later version.
*
* Ikayaki is distributed in the hope that it will be useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
* GNU General Public License for more details.
*
* You should have received a copy of the GNU General Public License
* along with Ikayaki; if not, write to the Free Software
* Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
*/
package ikayaki.gui;

/**
* Abstract class that implements general construction of a graphical plot.
*
* @author
*/
public abstract AbstractPlot extends JPanel
```

```
/**
* Contains all the data that is shown in this graph.
*/
private Vector<Measurement> measurement = null;

/**
* Adds new measurement data to plot.
* @param declination Declination coordinate of the measurement.
* @param inclination Inclination coordinate of the measurement.
*/
public void addMeasurement(int declination, int inclination){
return null; // TODO
}

/**
* High lights measurement in plot in the given index.
* @param index Index of measurement to be highlighted.
* @throws IndexOutOfBoundsException If no such measurement existed.
*/
public void highlightMeasurement(int index){
return null; // TODO
}

/**
* Highlights a set of measurements in given range.
* @param from Starting index of highlighted measurements.
* @param to End index of highlighted meeasurements.
* @throws IndexOutOfBoundsException If one or both of the indices are out of bounds.
*/
public void highlightMeasurementRange(int from, int to){
return null; // TODO
}

/**
* dehighlights all values in this graph.
*/
public void unHighlightAll(){
return null; // TODO
}

/**
* Removes all measurements from the graph.
*/
```

```
public void resetGraph(){
return null; // TODO
}

/**
* Returns the number of measurements in this graph.
* @return Number of measurements.
*/
public int getNumMeasurements(){
return null; // TODO
}
```

6.10.3 IntensityPlot

```
/*
* IntensityPlot.java
*
* Copyright (C) 2005 Project SQUID, http://www.cs.helsinki.fi/group/squid/
*
* This file is part of Ikayaki.
*
* Ikayaki is free software; you can redistribute it and/or modify
* it under the terms of the GNU General Public License as published by
* the Free Software Foundation; either version 2 of the License, or
* (at your option) any later version.
*
* Ikayaki is distributed in the hope that it will be useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
* GNU General Public License for more details.
*
* You should have received a copy of the GNU General Public License
* along with Ikayaki; if not, write to the Free Software
* Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
*/
package ikayaki.gui;

/**
* Implements intensity graph plot.
*
* @author
*/
public class IntensityPlot   extends AbstractPlot
```

6.10.4 StereoPlot

```
/*
* StereoPlot.java
*
* Copyright (C) 2005 Project SQUID, http://www.cs.helsinki.fi/group/squid/
*
* This file is part of Ikayaki.
*
* Ikayaki is free software; you can redistribute it and/or modify
* it under the terms of the GNU General Public License as published by
* the Free Software Foundation; either version 2 of the License, or
* (at your option) any later version.
*
* Ikayaki is distributed in the hope that it will be useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
* GNU General Public License for more details.
*
* You should have received a copy of the GNU General Public License
* along with Ikayaki; if not, write to the Free Software
* Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
*/
package ikayaki.gui;

/**
* Implements stereo graph plot.
*
* @author
*/
public class IntensityPlot   extends AbstractPlot
```

7 Testing

Because program will be used to control a magnetometer, testing will be more important than in normal software engineering student projects. We will do unit testing for each class, integrate testing to program and use separate squid emulator to test squid interface system.

In unit testing each class is tested independently. Unit testing will be done by using JUnit. Every programmer will test his own classes. Class should be tested when it is finished and corrected before integration test begins.

Integration testing tests interfaces between classes. It will be done by going through all user interface protos and checking that all sections in requirements document can be done.

Some critical sequences which are done many times with program should be done too.

Squid interface integration testing is done simulating real system with emulator. It will be done using Squid-emulator before testing it with real magnetometer. Squid-emulator runs in different machine and is connected by few (2-3) Serial I/O cables. Squid-emulator will be tested with old program (2G) same way before testing Ikayaki-system so that it will have all same tested properties which old program have and both systems have same results with squid emulator.

To verify that old program and new program works same way, we will do critical measurement with old program and emulator, save emulators log file and then use emulator with that log file and do same critical measurement with new program and see that both have same results.

If Rita testing utility is easy enough to use it will be used in testing. Tests will be constructed in such way that every line of code is visited at least once.