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

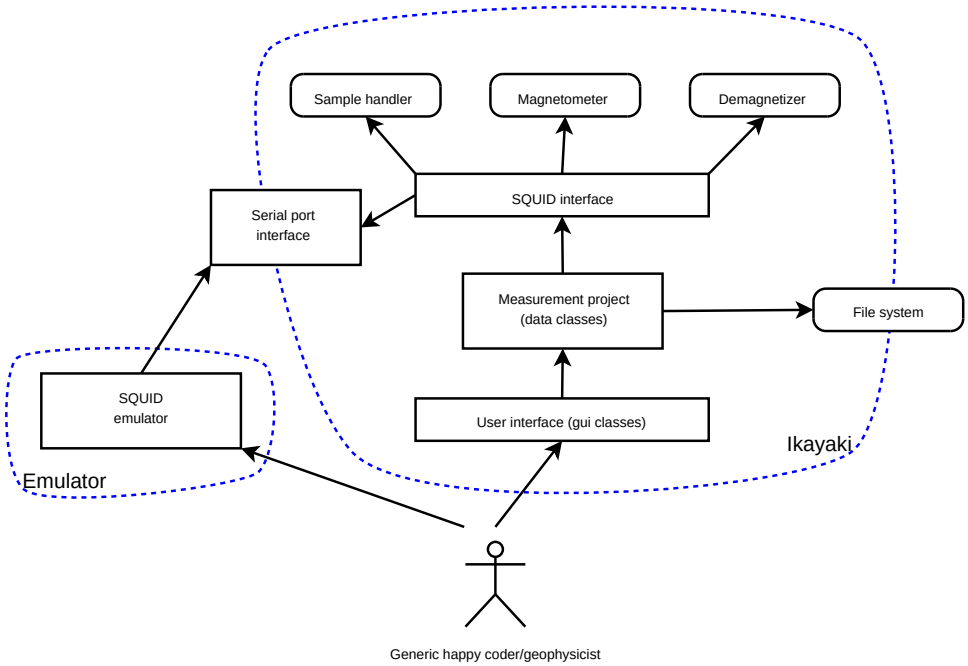

Figure 1: overview

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

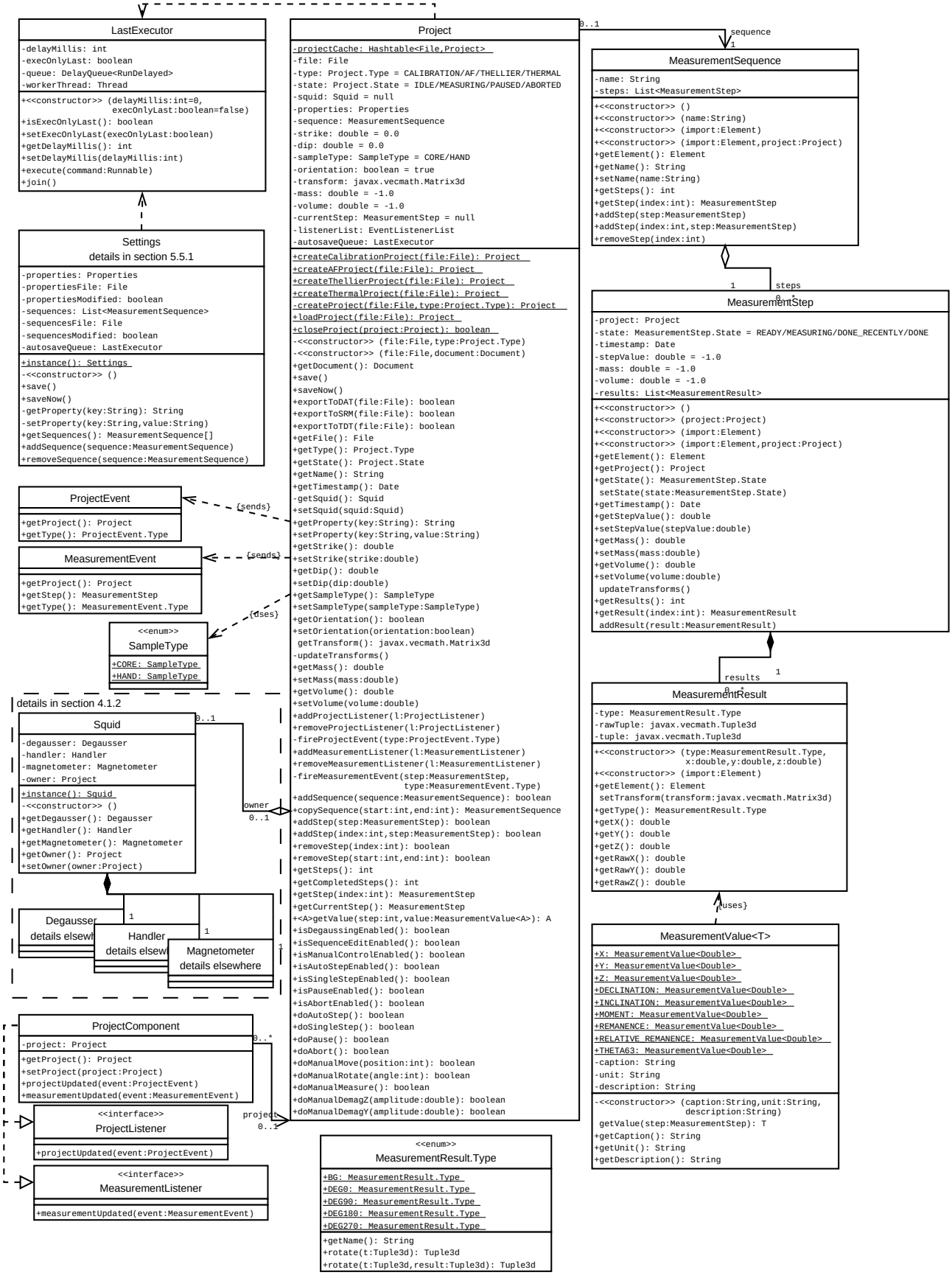

Figure 2: Squid class diagram: data classes

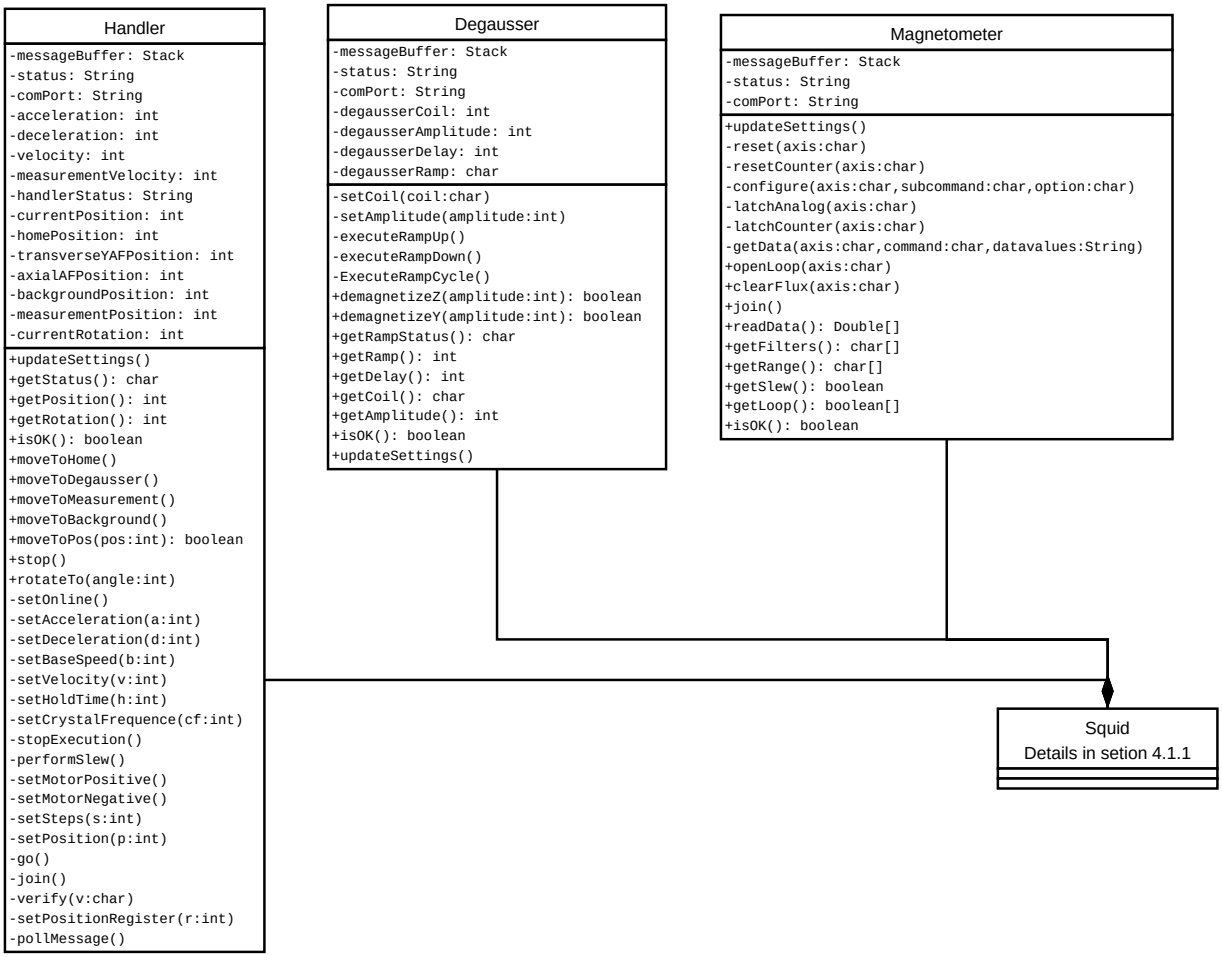

Figure 3: core-squid

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

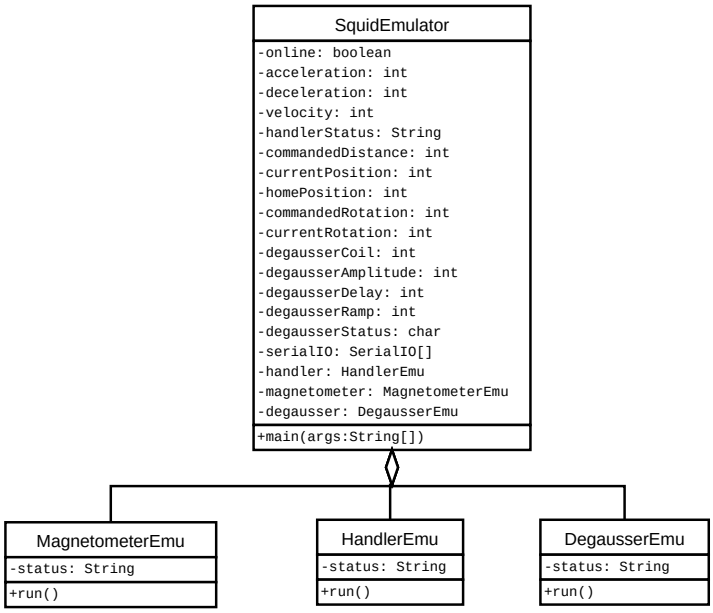

Figure 4: core-squidemu

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

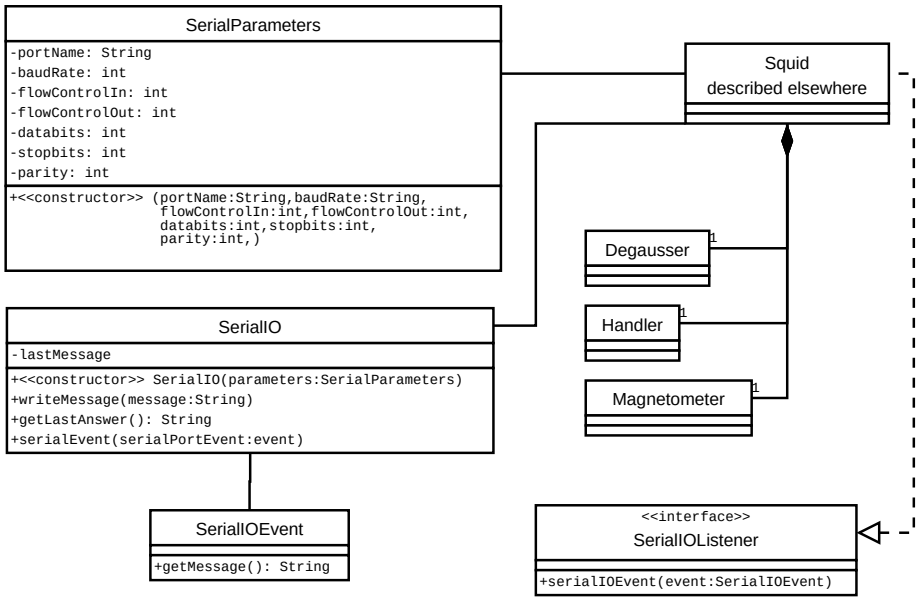

Figure 5: core-serial

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.

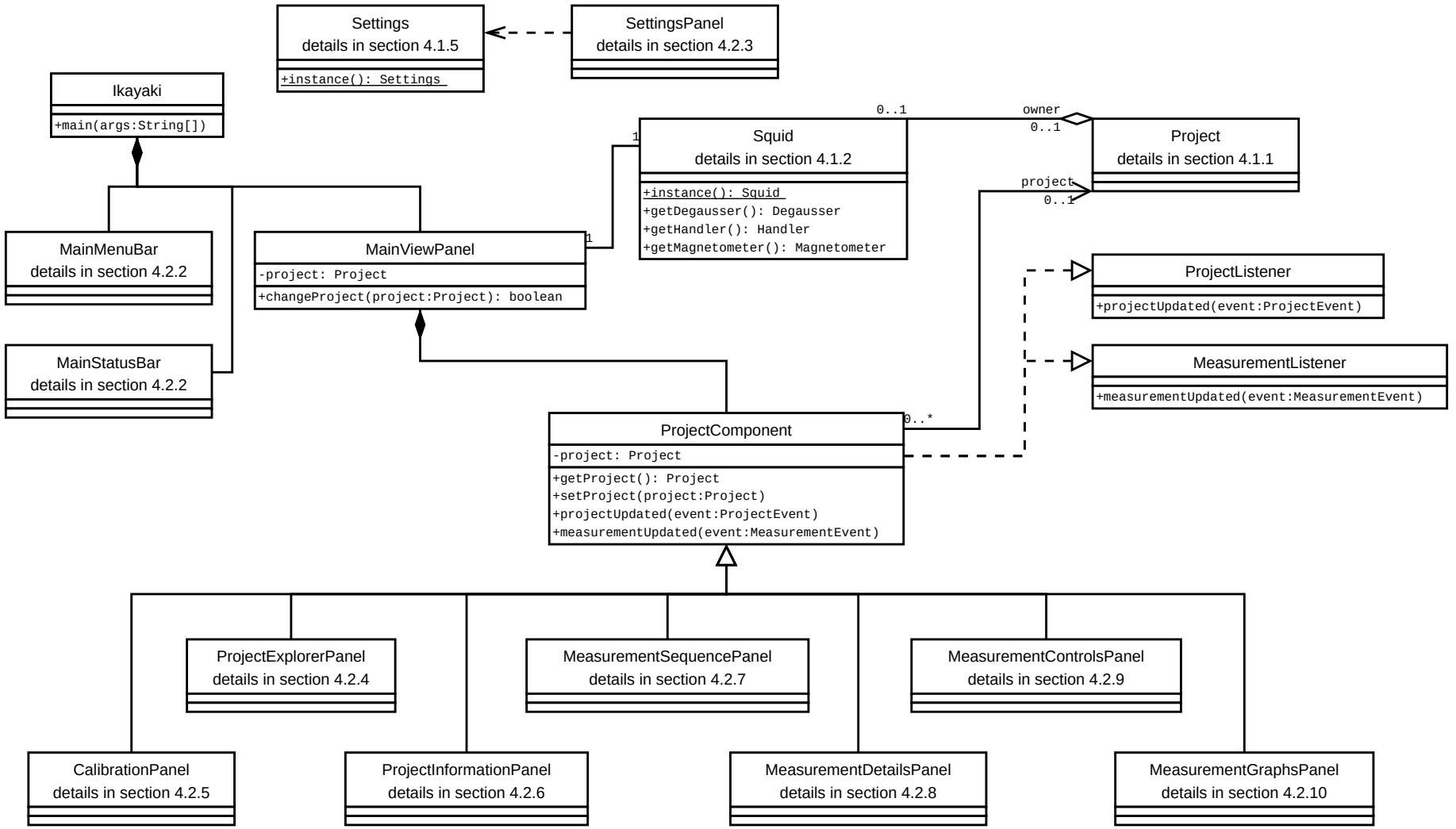

Figure 6: Squid class diagram: GUI classes

4.2.2 Main window

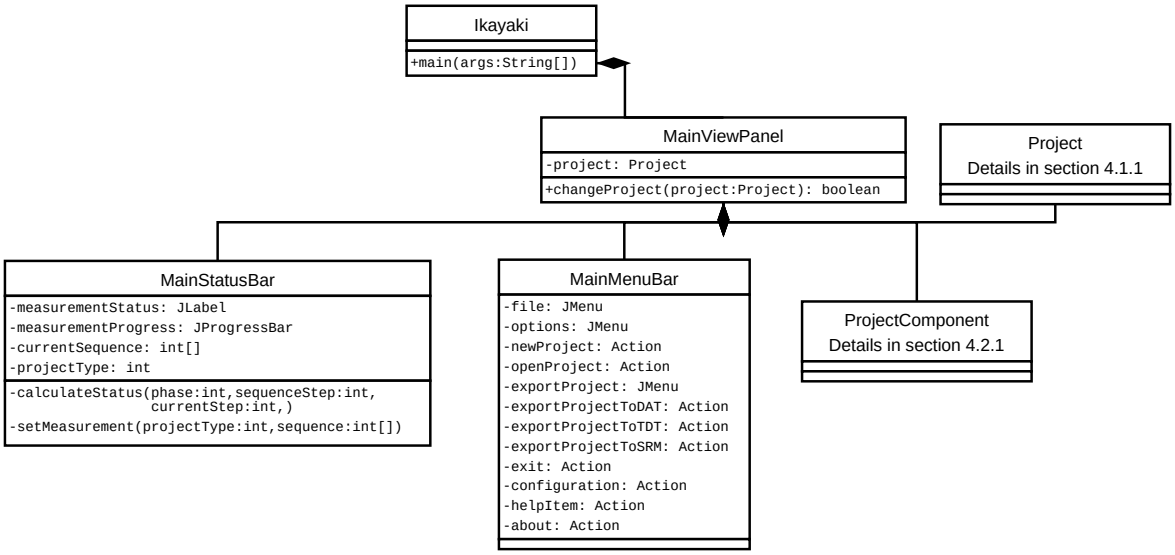

Figure 7: gui-main

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

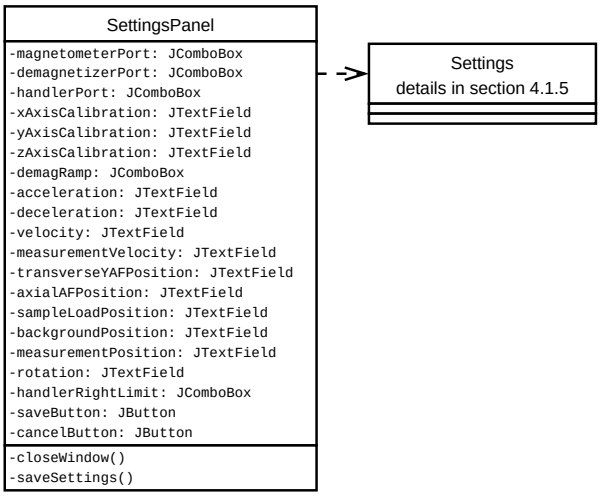

Figure 8: gui-configuration

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

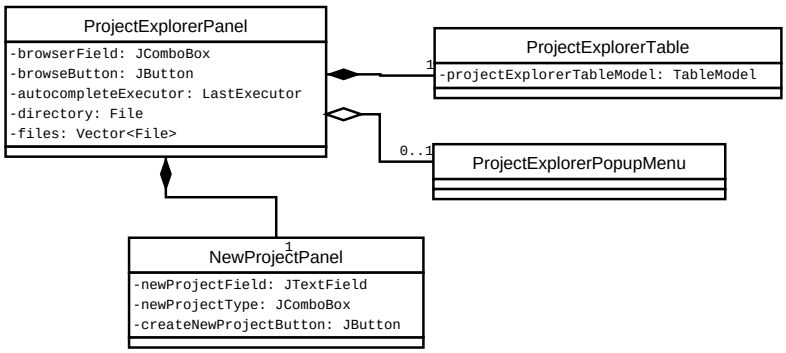

Figure 9: gui-explorer

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

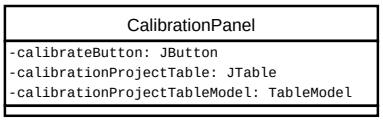

Figure 10: gui-calibration

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

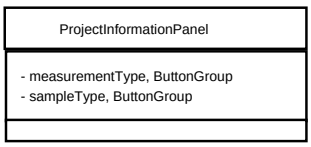

Figure 11: gui-projectinfo

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

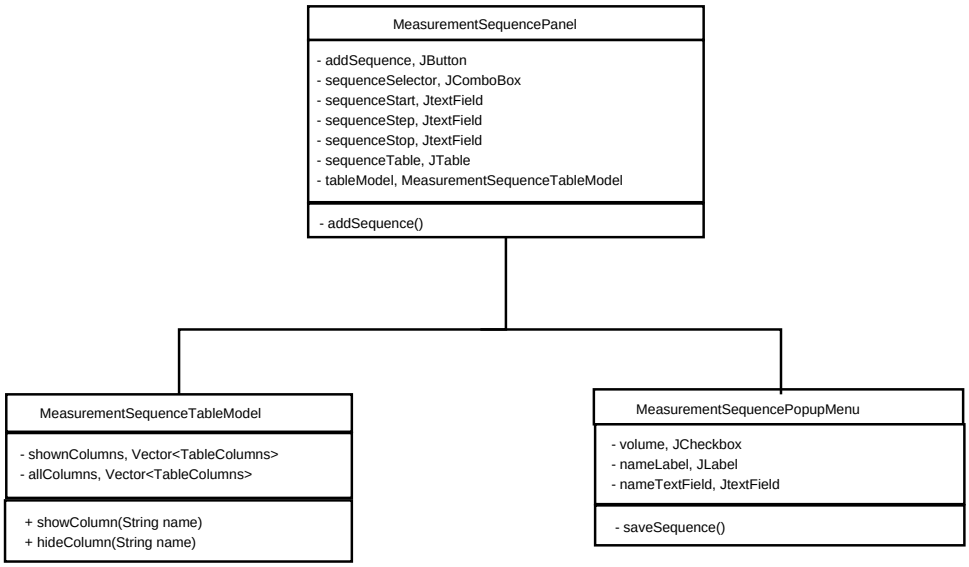

Figure 12: gui-sequence

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

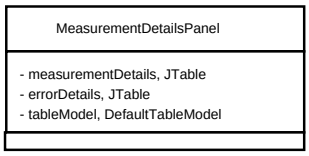

Figure 13: gui-mdetails

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

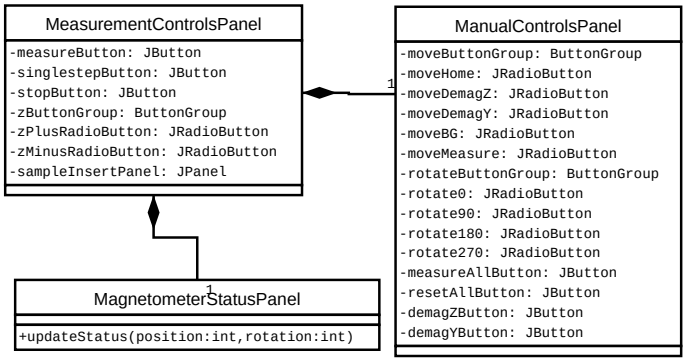

Figure 14: gui-mcontrols

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

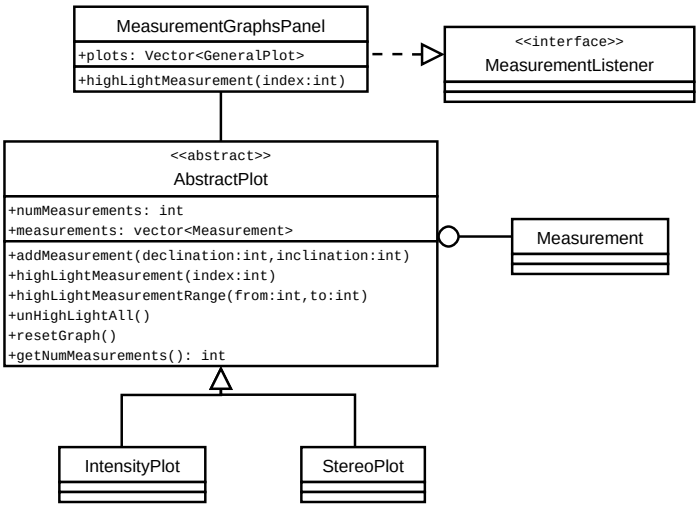

Figure 15: gui-graphs

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

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.

Package	ikayaki
Declaration	public class Project
Created by	ProjectExplorerPanel (6.4.1)
Uses 1	MeasurementSequence (5.1.4)
Uses 2	MeasurementStep (5.1.5)
Uses 3	MeasurementResult (5.1.7)
Uses 4	MeasurementValue (5.1.9)
Uses 5	Squid (5.2.1)
Uses 6	LastExecutor (5.6.1)
Uses 7	ProjectEvent (5.1.10)
Uses 8	MeasurementEvent (5.1.13)
Design patterns	Facade
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.

Fields of Project

private static Hashtable<File,Project> projectCache

Caches the created and loaded Project objects to make sure that no more than one object will be created for each physical file.

private File file

Location of the project file in the local file system. Autosaving will save the project to this file.

private Type type

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

private State state

Default value IDLE

Current state of the measurements. If no measurement is running, then state is IDLE. Only one measurement may be running at a time.

private Squid squid

Default value null

Pointer to the SQUID device interface, or null if this project is not its owner.

private Properties properties

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

private MeasurementSequence sequence

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 double strike

Default value 0.0

Strike of the sample. Will be used to create the transform matrix.

private double dip

Default value 0.0

Dip of the sample. Will be used to create the transform matrix.

private SampleType sampleType

Default value CORE

Type of the sample. Will be used to create the transform matrix.

private boolean orientation

Default value true

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 Matrix3d transform

Default value new Matrix3d().setIdentity()

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 double mass

Default value -1.0

Mass of the sample, or a negative value if no mass is defined.

private double volume

Default value -1.0

Volume of the sample, or a negative value if no volume is defined.

private MeasurementStep currentStep

Default value null

Current measurement step, or null if no measurement is running.

private EventListenerList listenerList

Default value new EventListenerList()

Listeners for this project.

private LastExecutor autosaveQueue

Default value new LastExecutor(500, true)

Scheduler for automatically writing the modified project to file after a short delay.

Methods of Project

public static Project createCalibrationProject(File file)

Creates a calibration project file.

Parameter 1 *file* - path for the new project file.

Returns the created project, or null if file was not writable.

Throws *NullPointerException* - if file is null.

public static Project createAFProject(File file)

Creates an AF project file.

Parameter 1 *file* - path for the new project file.

Returns the created project, or null if file was not writable.

Throws *NullPointerException* - if file is null.

public static Project createThellierProject(File file)

Creates a thellier project file.

Parameter 1 *file* - path for the new project file.

Returns the created project, or null if file was not writable.

Throws *NullPointerException* - if file is null.

public static Project createThermalProject(File file)

Creates a thermal project file.

Parameter 1 *file* - path for the new project file.

Returns the created project, or null if file was not writable.

Throws *NullPointerException* - if file is null.

private static Project createProject(File file, Type type)

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.

Parameter 1 *file* - path for the new project file.

Parameter 2 *type* - type of the project.

Returns the created project, or null if file was not writable.

Throws *NullPointerException* - if file is null.

public static Project loadProject(File file)

Loads a saved project file. If the file has already been loaded, will return a reference to the existing Project object.

Parameter 1 *file* - project file to be loaded.

Returns 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 boolean closeProject(Project project)

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.

Parameter 1 *project* - project to be closed.
Returns 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.

private Project(File file, Type type)

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.

Parameter 1 *file* - path for this project file. The file should exist (may be empty) and be writable, but this constructor will not check it.
Parameter 2 *type* - type of the project.
Returns the created project.
Throws *NullPointerException* - if any of the parameters is null.

private Project(File file, Document document)

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.

Parameter 1 *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.
Parameter 2 *document* - the document from which this project will be created.
Returns the created project.
Throws *NullPointerException* - if any of the parameters is null.
Throws *IllegalArgumentException* - if the document was not in the right format.

public synchronized Document getDocument()

Exports this project to a DOM document.

public synchronized void save()

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 void saveNow()

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 boolean exportToDAT(File file)

Writes the project to a file in DAT format.

Parameter 1	*file* - the file to save to.
Returns	true if the file was successfully written, otherwise false.
Throws	*NullPointerException* - if file is null.

public boolean exportToSRM(File file)

Writes the project to a file in SRM format.

Parameter 1	*file* - the file to save to.
Returns	true if the file was successfully written, otherwise false.
Throws	*NullPointerException* - if file is null.

public boolean exportToTDT(File file)

Writes the project to a file in TDT format.

Parameter 1	*file* - the file to save to.
Returns	true if the file was successfully written, otherwise false.
Throws	*NullPointerException* - if file is null.

public synchronized File getFile()

Returns the project file of this project.

public synchronized Type getType()

Returns the type of this project.

public synchronized State getState()

Returns the current measurement state of this project.

public synchronized String getName()

Returns the name of this project. The name is equal to the name of the project file without the file extension.

public synchronized Date getTimestamp()

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.

private synchronized Squid getSquid()

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?)

public synchronized boolean setSquid(Squid squid)

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.

Parameter 1 *squid* - pointer to the SQUID interface, or null to detach this project from it.

Returns 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 String getProperty(String key)

Returns a project information property.

Parameter 1 *key* - the key which is associated with the property.

Returns the specified property, or an empty String if the property is not set.

public synchronized void setProperty(String key, String value)

Sets a project information property.

Parameter 1 *key* - the key which is associated with the property.

Parameter 2 *value* - new value for the property, or null to remove the property.

public synchronized double getStrike()

Returns the strike of the sample.

public synchronized void setStrike(double strike)

Sets the strike of the sample and calls updateTransforms().

public synchronized double getDip()

Returns the dip of the sample.

public synchronized void setDip(double dip)

Sets the dip of the sample and calls updateTransforms().

public synchronized SampleType getSampleType()

Returns the type of the sample.

public synchronized void setSampleType(SampleType sampleType)

Sets the type of the sample and calls updateTransforms().

Throws *NullPointerException* - if sampleType is null.

public synchronized boolean getOrientation()

Returns the orientation of the sample.

Returns true if the sample orientation is +Z, or false if it is -Z.

public synchronized void setOrientation(boolean orientation)

Sets the orientation of the sample and calls updateTransforms().

Parameter 1 *orientation* - true if the sample orientation is +Z, or false if it is -Z.

synchronized Matrix3d getTransform()

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.

Returns reference to the transformation matrix.

private synchronized void updateTransforms()

Recalculates the transformation matrix and updates all measurements. This method is called automatically by the setStrike(), setDip() and setSampleType() methods.

public synchronized double getMass()

Returns the mass of the sample.

Returns mass of the sample, or a negative number if no mass is specified.

public synchronized void setMass(double mass)

Sets the mass of the sample.

Parameter 1 *mass* - mass of the sample, or a negative number to clear it.

public synchronized double getVolume()

Returns the volume of the sample.

Returns volume of the sample, or a negative number if no volume is specified.

public synchronized void setVolume(double volume)

Sets the volume of the sample.

Parameter 1 *volume* - volume of the sample, or a negative number to clear it.

public synchronized void addProjectListener(ProjectListener l)

Adds a ProjectListener to the project.

Parameter 1 *l* - the listener to be added.

public synchronized void removeProjectListener(ProjectListener l)

Removes a ProjectListener from the project.

Parameter 1 *l* - the listener to be removed

private synchronized void fireProjectEvent(ProjectEvent.Type type)

Notifies all listeners that have registered for ProjectEvents.

Parameter 1 *type* - type of the event.

public synchronized void addMeasurementListener(MeasurementListener l)

Adds a MeasurementListener to the project.

Parameter 1 *l* - the listener to be added.

public synchronized void removeMeasurementListener(MeasurementListener l)

Removes a MeasurementListener from the project.

Parameter 1 *l* - the listener to be removed

private synchronized void fireMeasurementEvent(MeasurementStep step, MeasurementEvent.Type type)

Notifies all listeners that have registered for MeasurementEvents.

Parameter 1 *step* - the measurement step that has generated the event.

Parameter 2 *type* - the type of the event.

public synchronized boolean addSequence(MeasurementSequence sequence)

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.

Parameter 1 *sequence* - the measurement sequence to be added.

Returns true if the steps were added, or false if isSequenceEditEnabled() was false.

Throws *NullPointerException* - if sequence is null.

public synchronized MeasurementSequence copySequence(int start, int end)

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.

Parameter 1 *start* - index of the first step in the sequence.

Parameter 2 *end* - index of the last step in the sequence. If end < start, then an empty sequence will be returned.

Returns 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 boolean addStep(MeasurementStep step)

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.

Parameter 1 *step* - the measurement step to be added.

Returns true if the step was added, or false if isSequenceEditEnabled() was false.

Throws *NullPointerException* - if step is null.

public synchronized boolean addStep(int index, MeasurementStep step)

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.

Parameter 1	*index* - the index to which the step will be added.
Parameter 2	*step* - the measurement step to be added.
Returns	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 removeStep(int index)

Removes a step from this project's sequence. Completed measurements can not be removed.
If isSequenceEditEnabled() is false, nothing will be done.

Parameter 1	*index* - the index of the step to be removed.
Returns	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 start, int end)

Removes a series of steps from this project's sequence. Completed measurements can not be removed.
If isSequenceEditEnabled() is false, nothing will be done.

Parameter 1	*start* - the first index to be removed.
Parameter 2	*end* - the last index to be removed. If end < start, no steps will be removed.
Returns	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 int getSteps()

Returns the number of steps in this project.

public synchronized int getCompletedSteps()

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 MeasurementStep getStep(int index)

Returns a step from the sequence.

Parameter 1	*index* - the index of the step.
Returns	the specified step.
Throws	*IndexOutOfBoundsException* - if the index is out of range (index < 0 || index >= getSteps()).

public synchronized MeasurementStep getCurrentStep()

Returns the step that is currently being measured.

Returns	the currently measured step, or null if no measurement is active.

public synchronized <A> A getValue(int step, MeasurementValue<A> algorithm)

Calculates and returns a value from a measurement step. The specified MeasurementValue's algorithm will be used and the results returned.

Parameter 1	*step* - the measurement step from which the value is calculated.
Parameter 2	*algorithm* - the algorithm for calculating the desired value.
Returns	the value returned by the algorithm, or null if it was not possible to calculate it.
Throws	*NullPointerException* - if algorithm is null.

public synchronized boolean isDegaussingEnabled()

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 isSequenceEditEnabled()

Tells whether it is allowed to edit the sequence. The returned value depends on the type and state of this project.

public synchronized boolean isManualControlEnabled()

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 isAutoStepEnabled()

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 isSingleStepEnabled()

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 isPauseEnabled()

Tells whether it is possible to pause the measurement. The returned value depends on the type and state of this project.

public synchronized boolean isAbortEnabled()

Tells whether it is possible to abort the measurement. The returned value depends on the type and state of this project.

public synchronized boolean doAutoStep()

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.

Returns true if the measurement was started, otherwise false.

public synchronized boolean doSingleStep()

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.

Returns true if the measurement was started, otherwise false.

public synchronized boolean doPause()

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.

Returns true if the measurement will pause, otherwise false.

public synchronized boolean doAbort()

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.

Returns true if the measurement will abort, otherwise false.

public synchronized boolean doManualMove(int position)

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.

Parameter 1 *position* - the position to move the handler to.

Returns true if the operation was started, otherwise false.

public synchronized boolean doManualRotate(int angle)

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.

Parameter 1 *angle* - the angle to rotate the handler to.

Returns true if the operation was started, otherwise false.

public synchronized boolean doManualMeasure()

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.

Returns true if the operation was started, otherwise false.

public synchronized boolean doManualDemagZ(double amplitude)

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.

Parameter 1 *amplitude* - the amplitude to demagnetize in mT.

Returns true if the operation was started, otherwise false.

public synchronized boolean doManualDemagY(double amplitude)

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.

Parameter 1 *amplitude* - the amplitude to demagnetize in mT.

Returns true if the operation was started, otherwise false.

5.1.2 Project.Type

The type of the project. Options are CALIBRATION, AF, THELLIER and THERMAL.

Package ikayaki

Declaration public enum Type

5.1.3 Project.State

The state of the project's measurements. Options are IDLE, MEASURING, PAUSED, ABORTED.

Package ikayaki

Declaration public enum State

5.1.4 MeasurementSequence

A list of measurement steps. Steps can be added or removed from the sequence. All operations are thread-safe.

Package ikayaki
Declaration public class MeasurementSequence
Created by Project (5.1.1)
Uses 1 MeasurementStep (5.1.5)

Fields of MeasurementSequence

private String name

Default value null

Name of the sequence or null if it has no name.

private List<MeasurementStep> steps

Default value new ArrayList<MeasurementStep>()

The measurement steps of this sequence.

Methods of MeasurementSequence

public MeasurementSequence()

Creates an empty sequence with no name.

Returns the created sequence.

public MeasurementSequence(String name)

Creates an empty sequence with the specified name.

Parameter 1 *name* - name of the sequence.
Returns the created sequence.

public MeasurementSequence(Element import)

Creates a sequence from the specified element.

Parameter 1	*import* - the element from which this sequence will be created.
Returns	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)

Creates a sequence from the specified element for a project.

Parameter 1	*import* - the element from which this sequence will be created.
Parameter 2	*project* - the project whose sequence this will be. Needed for importing the measurement steps correctly.
Returns	the created sequence.
Throws	*NullPointerException* - if import is null.
Throws	*IllegalArgumentException* - if the element was not in the right format.

public synchronized Element getElement()

Exports this sequence to a DOM element.

public synchronized String getName()

Returns the name of this sequence.

Returns	the name, or null if it has no name

public synchronized void setName(String name)

Sets the name of this sequence.

public synchronized int getSteps()

Returns the number of steps in this sequence.

public synchronized MeasurementStep getStep(int index)

Returns the specified step from this sequence.

Parameter 1	*index* - the index of the step.
Returns	the specified step.
Throws	*IndexOutOfBoundsException* - if the index is out of range (index < 0 || index >= getSteps()).

public synchronized void addStep(MeasurementStep step)

Appends a step to this sequence.

Parameter 1	*step* - the measurement step to be added.
Throws	*NullPointerException* - if step is null.

public synchronized void addStep(int index, MeasurementStep step)

Adds a step to the specified index of this sequence.

Parameter 1	*index* - the index to which the step will be added.
Parameter 2	*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 removeStep(int index)

Removes a step from this sequence.

Parameter 1	*index* - the index of the step to be removed.
Throws	*IndexOutOfBoundsException* - if the index is out of range (index < 0 || index >= getSteps()).

5.1.5 MeasurementStep

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.

Package	ikayaki
Declaration	public class MeasurementStep
Created by	Project (5.1.1)
Created by	MeasurementSequencePanel (6.7.1)
Uses 1	Project (5.1.1)
Uses 2	MeasurementResult (5.1.7)

Fields of MeasurementStep

private Project project

Default value null

The project that owns this step, or null if there is no owner.

private State state

Default value READY

Tells if this step has been completed or not, or if a measurement is still running.

private Date timestamp

Default value null

The time the measurements were completed, or null if that has not yet happened.

private double stepValue

Default value -1.0

The AF/Thermal value of this step, or a negative number if it has not been specified.

private double mass

Default value -1.0

The mass of this step's sample, or a negative number to use the project's default mass.

private double volume

Default value -1.0

The volume of this step's sample, or a negative number to use the project's default volume.

private List<MeasurementResult> results

Default value new ArrayList<MeasurementResult>()

The individual measurement results that are part of this measurement step.

Methods of MeasurementStep

public MeasurementStep()

Creates a blank measurement step.

Returns the created measurement step.

public MeasurementStep(Project project)

Creates a blank measurement step for a project.

Parameter 1 *project* - the project who is the owner of this step.

Returns the created measurement step.

public MeasurementStep(Element import)

Creates a measurement step from the specified element.

Parameter 1 *import* - the element from which this step will be created.

Returns 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)

Creates a measurement step from the specified element for a project.

Parameter 1 *import* - the element from which this step will be created.

Parameter 2 *project* - the project who is the owner of this step.

Returns the created measurement step.

Throws *NullPointerException* - if import is null.

Throws *IllegalArgumentException* - if the element was not in the right format.

public synchronized Element getElement()

Exports this step to a DOM element.

public synchronized Project getProject()

Returns the owner project of this step, or null if there is no owner.

public synchronized State getState()

Tells if this step has been completed or not, or if a measurement is still running.

void synchronized setState(State state)

Sets the completion status of this step. Only the owner project may set the state.

Throws *NullPointerException* - if state is null.

public synchronized Date getTimestamp()

Returns the time the measurements were completed, or null if that has not yet happened.

public synchronized double getStepValue()
Returns the AF/Thermal value of this step, or a negative number if it has not been specified.

public synchronized void setStepValue(double stepValue)
Sets the value of this step. A negative value will clear it.

public synchronized double getMass()
Returns the mass of this step's sample, or a negative number to use the project's default mass.

public synchronized void setMass(double mass)
Sets the mass of this step's sample. A negative value will clear it.

public synchronized double getVolume()
Returns the volume of this step's sample, or a negative number to use the project's default volume.

public synchronized void setVolume(double volume)
Sets the volume of this step's sample. A negative value will clear it.

synchronized void updateTransforms()
Updates all of the measurement results with the owner project's transformation matrix. If there is no owner, an identity matrix will be used.

public synchronized int getResults()
Returns the number of results in this step.

public synchronized MeasurementResult getResult(int index)
Returns the specified result from this step.

Parameter 1	*index* - the index of the result.
Returns	the specified result.
Throws	*IndexOutOfBoundsException* - if the index is out of range (index < 0 || index >= getResults()).

public synchronized void addResult(MeasurementResult result)
Appends a measurement result to this step. The transformation matrix of the result will be updated automatically.

Parameter 1	*result* - the result to be added.
Throws	*NullPointerException* - if result is null.

5.1.6 MeasurementStep.State

The state of a measurement step. Options are READY, MEASURING, DONE_RECENTLY and DONE.

Package	ikayaki
Declaration	public enum State

5.1.7 MeasurementResult

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.

Package ikayaki
Declaration public class MeasurementResult
Created by Magnetometer (5.2.4)

Fields of MeasurementResult

private Type type

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 Tuple3d rawTuple

Default value new Tuple3d()

The unmodified measurements recieved from the squid.

private Tuple3d tuple

Default value new Tuple3d()

The measurements with the rotation and transformation matrix applied.

Methods of MeasurementResult

public MeasurementResult(Type type, double x, double y, double z)

Creates a new measurement result.

Parameter 1 *type* - the type (background or rotation) of this result.
Parameter 2 *x* - the measured X coordinate value.
Parameter 3 *y* - the measured Y coordinate value.
Parameter 4 *z* - the measured Z coordinate value.
Returns the created measurement result.
Throws *NullPointerException* - if type is null.

public MeasurementResult(Element import)

Creates a measurement result from the specified element. This will not apply the transformation matrix.

Parameter 1	*import* - the element from which this result will be created.
Returns	the created measurement result.
Throws	*NullPointerException* - if import is null.
Throws	*IllegalArgumentException* - if the element was not in the right format.

public Element getElement()

Exports this result to a DOM element.

void setTransform(Matrix3d transform)

Applies a transformation matrix to this result.

Parameter 1	*transform* - the matrix to be applied. If null, will assume identity matrix.

public Type getType()

Returns the type of this result (background or rotation).

public double getX()

Returns the rotated and transformed X coordinate of this result.

public double getY()

Returns the rotated and transformed Y coordinate of this result.

public double getZ()

Returns the rotated and transformed Z coordinate of this result.

public double getRawX()

Returns the unmodified X coordinate of this result as recieved from the Squid.

public double getRawY()

Returns the unmodified Y coordinate of this result as recieved from the Squid.

public double getRawZ()

Returns the unmodified Z coordinate of this result as recieved from the Squid.

5.1.8 MeasurementResult.Type

The orientation of the sample when it was measured. Options are BG, DEG0, DEG90, DEG180 and DEG270.

Package	ikayaki
Declaration	public enum Type

Methods of MeasurementResult.Type

public String getName()

Returns "BG", "0", "90", "180" or "270"

public Tuple3d rotate(Tuple3d t)

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.

Parameter 1 *t* - old values that need to be rotated.

Returns a new object with the rotated values.

public Tuple3d rotate(Tuple3d t, Tuple3d result)

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.

Parameter 1 *t* - old values that need to be rotated.

Parameter 2 *result* - where the new values will be saved.

Returns the same as the result parameter, or a new object if it was null.

5.1.9 MeasurementValue

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.

Package ikayaki

Declaration public abstract class MeasurementValue<T>

Uses 1 MeasurementStep (5.1.5)

Design patterns Strategy

Fields of MeasurementValue

public static final MeasurementValue<Double> X

Calculates the average of all X components.

public static final MeasurementValue<Double> Y

Calculates the average of all Y components.

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

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

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

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

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

public static final MeasurementValue<Double> RELATIVE_REMANENCE
Calculates the remanence relative to the first measurement's remanence.

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

private String caption
A short name for the value.

private String unit
The unit of the value.

private String description
A long description of the value.

Methods of MeasurementValue

public MeasurementValue(String caption, String unit, String description)
Creates a new measurement value.

Parameter 1	*caption* - a short name for the value.
Parameter 2	*unit* - the unit of the value.
Parameter 3	*description* - a long description of the value.
Returns	the created measurement value.
Throws	*NullPointerException* - if any of the arguments is null.

abstract T getValue(MeasurementStep step)
Calculates a specific value from a measurement step.

Parameter 1	*step* - the step from which the value will be calculated.
Returns	the calculated value, or null if it was not possible to calculate it.
Throws	*NullPointerException* - if step is null.

public String getCaption()
Returns a short name for the value.

public String getUnit()
Returns the unit of the value.

public String getDescription()
Returns a long description of the value.

5.1.10 ProjectEvent

ProjectEvent is used to notify others about the state change of a project.

Package	ikayaki
Declaration	public class ProjectEvent
Extends	EventObject
Created by	Project (5.1.1)

Method	Page
ProjectEvent(Project,Type)	37
getProject()	37
getType()	37

Fields of ProjectEvent

private Project project
The project that sent this event.

private Type type
The type of event this is.

Methods of ProjectEvent

public ProjectEvent(Project project, Type type)
Creates a new project event.

Parameter 1	*project* - the project that sends this event.
Parameter 2	*type* - the type of the event.
Returns	the created event.
Throws	*NullPointerException* - if any of the arguments is null.

public Project getProject()
Returns the project that sent this event.

public Type getType()
Returns the type of this event.

5.1.11 ProjectEvent.Type

The type of a project event. Options are STATE_CHANGED, DATA_CHANGED, FILE_SAVED.

Package	ikayaki
Declaration	public enum Type

5.1.12 ProjectListener

Defines a listener for project events.

Package	ikayaki
Declaration	public interface ProjectListener
Extends	EventListener

Method	Page
projectUpdated(ProjectEvent)	38

Methods of ProjectListener

public void projectUpdated(ProjectEvent event)

Will be invoked whenever a project event happens.

Parameter 1	*event* - the event that happened.

5.1.13 MeasurementEvent

MeasurementEvent is used to notify listeners about the stages of an ongoing measurement.

Package	ikayaki
Declaration	public class MeasurementEvent
Extends	EventObject
Created by	Project (5.1.1)

Method	Page
MeasurementEvent(Project,MeasurementStep,Type)	38
getProject()	39
getStep()	39
getType()	39

Fields of MeasurementEvent

private Project project

The project whose measurement sent this event.

private MeasurementStep step

The measurement that sent this event.

private Type type

The type of event this is.

Methods of MeasurementEvent

public MeasurementEvent(Project project, MeasurementStep step, Type type)

Creates a new measurement event.

Parameter 1	*project* - the project whose measurement sent this event.
Parameter 2	*step* - the measurement that sent this event.
Parameter 3	*type* - the type of event this is.
Returns	the created event.
Throws	*NullPointerException* - if any of the arguments is null.

public Project getProject()

Returns the project whose measurement sent this event.

public MeasurementStep getStep()

Returns the measurement that sent this event.

public Type getType()

Returns the type of event this is.

5.1.14 MeasurementEvent.Type

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.

Package	ikayaki
Declaration	public enum Type

5.1.15 MeasurementListener

Defines a listener for measurement events.

Package	ikayaki
Declaration	public interface MeasurementListener
Extends	EventListener

Method	**Page**
measurementUpdated(MeasurementEvent)	39

Methods of MeasurementListener

public void measurementUpdated(MeasurementEvent event)

Will be invoked whenever a measurement event happens.

Parameter 1	*event* - the event that happened.

5.2 Squid interface

5.2.1 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.

Package	ikayaki.squid
Declaration	public class Squid
Created by	MainViewPanel (6.2.2)
Uses 1	Settings (5.5.1)
Uses 2	Degausser (5.2.2)
Uses 3	Handler (5.2.3)
Uses 4	Magnetometer (5.2.4)
Design patterns	Singleton

Fields of Squid

private static final Squid instance
Instance of the Squid interface.

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

private final Degausser degausser
Instance of the degausser interface.

private final Handler handler
Instance of the handler interface.

private final Magnetometer magnetometer
Instance of the magnetometer interface.

Methods of Squid

public static synchronized Squid instance()
Returns a reference to the Squid. If it has not yet been created, will create one.

private Squid()

Initializes the Squid interface. Creates instances of Degausser, Handler and Magnetometer.

public Degausser getDegausser()

Returns an interface for controlling the degausser.

Returns Handler for Degausser if available

public Handler getHandler()

Returns an interface for controlling the handler.

Returns Handler for Handler if available

public Magnetometer getMagnetometer()

Returns an interface for controlling the magnetometer.

Returns Handler for Magnetometer if available

public synchronized void updateSettings()

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 boolean isOK()

Checks whether all devices are working correctly.

Returns true if everything is correct, otherwise false.

public synchronized boolean setOwner(Project owner)

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.

Parameter 1 *owner* - the project that will have exclusive access to the Squid.

Returns true if successful, false if the existing owner had a running measurement.

public synchronized Project getOwner()

Returns project that is currently using the Squid.

Returns the project, or null if none is using the Squid.

5.2.2 Degausser

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.

Package	ikayaki.squid
Declaration	public class Degausser
Implements	SerialIOListener
Created by	Squid (5.2.1)
Uses 1	Settings (5.5.1)
Uses 2	SerialIO (5.4.1)
Event A	*On SerialIOEvent* - reads the message and puts it in a buffer

Fields of Degausser

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

private String status
Degaussers current status

private SerialIO serialIO
COM port for communication.

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

private int degausserAmplitude
0->3000 default amp 0

private int degausserDelay
1-9 seconds default delay 1 second

private int degausserRamp
(3, 5, 7, 9) default 3

private char degausserRamp
Z=Zero, T=Tracking, ?=Unknown

Methods of Degausser

public Degausser()
Creates a new degausser interface. Opens connection to degausser COM port (if not open yet) and reads settings from the Setting class.

public void updateSettings()
Checks which settings have changed and updates the degausser interface. This method will be called by the Squid class.

private void setCoil(char coil)
Sets coil X,Y,Z.
Parameter 1 *coil* - coil to set on.

private void setAmplitude(int amplitude)
Sets amplitude to ramp, range 0 to 3000.
Parameter 1 *amplitude* - amplitude to demag.

private void executeRampUp()
Performs Ramp up.

private void executeRampDown()
Brings Ramp down.

private void executeRampCycle()
Performs Ramp up and down.

public boolean demagnetizeZ(int amplitude)
Performs full sequence to demagnetize Z coil with the given amplitude. Blocking method.
Parameter 1 *amplitude* - amplitude to demag.
Returns true if process was sended succesfully, otherwise false.

public boolean demagnetizeY(int amplitude)
Performs full sequence to demagnetize Y (and X) coil with the given amplitude. Blocking method.
Parameter 1 *amplitude* - amplitude to demag.
Returns true if process was sended succesfully, otherwise false.

public char getRampStatus()
Sends status query to degausser and returns answer. Blocking.
Returns Z=Zero, T=Tracking, ?=Unknown

public int getRamp()
Sends ramp query to degausser and returns answer. Blocking.
Returns 3, 5, 7 or 9

public int getDelay()
Sends delay query to degausser and returns answer. Blocking.
Returns 1 to 9 as seconds

public char getCoil()
Sends coil query to degausser and returns answer. Blocking.
Returns X=X Axis, Y=Y Axis, Z=Z Axis, ?=Unknown

public int getAmplitude()
Sends amplitude query to degausser and returns answer. Blocking.
Returns 0 to 3000

public boolean isOK()
Checks if connection is ok.
Returns true if ok.

5.2.3 Handler

Offers an interface for controlling the sample handler.

Package	ikayaki.squid
Declaration	public class Handler
Implements	SerialIOListener
Created by	Squid (5.2.1)
Uses 1	Settings (5.5.1)
Uses 2	SerialIO (5.4.1)
Event A	*On SerialIOEvent* - reads message and puts it in a buffer

Fields of Handler

private Stack messageBuffer
Buffer for incoming messages, readed when needed.

private String status
Handlers current status.

private SerialIO serialIO
COM port for communication.

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

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

private int velocity
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 measurementVelocity
Speed in measurement, should be small.

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

private int currentPosition
Value between 1 and 16,777,215.

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

private int transverseYAFPosition
AF demag position for transverse.

private int axialAFPosition
Axial AF demag 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
Position in steps, must be divisible by 10. Relative to Home.

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

Methods of Handler

public Handler()
Creates a new handler interface. Opens connection to handler COM port and reads settings from the Settings class.

public void updateSettings()

Checks which settings have changed and updates the handler interface. This method will be called by the Squid class.

public char getStatus()

Returns current status on Sample Handler.

Returns 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 int getPosition()

Returns current known position.

Returns Value between 1 and 16,777,215

public int getRotation()

Returns current known rotation.

Returns Value between 0 and 2000

public boolean isOK()

checks if connection is ok.

Returns True if ok

public void moveToHome()

Commands the holder to move to home position. Only starts movement, needs to poll with join() when movement is finished.

public void moveToDegausser()

Commands the holder to move to degauss position. Only starts movement, needs to poll with join() when movement is finished.

public void moveToDegausser()

Commands the holder to move to degauss position. Only starts movement, needs to poll with join() when movement is finished.

public void moveToMeasurement()

Commands the holder to move to measure position. Only starts movement, needs to poll with join() when movement is finished.

public void moveToBackground()

Commands the holder to move to background position. Only starts movement, needs to poll with join() when movement is finished.

public boolean moveToPos(int pos)

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.

Parameter 1 *pos* - the position where the handler will move to.

Returns true if given position was ok, otherwise false.

public void stop()

Commands the handler to stop its current job.

public void rotateTo(int angle)

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.

Parameter 1 *angle* - the angle in degrees to rotate the handler to.

private void setOnline()

Sends message to handler go online (@0).

private void setAcceleration(int a)

Sends message to handler to set acceleration (Aa).

Parameter 1 *a* - Acceleration is a number from 0 to 127

private void setDeceleration(int d)

Sends message to handler to set deceleration (Dd).

Parameter 1 *a* - Deceleration is a number from 0 to 127

private void setBaseSpeed(int b)

Sends message to handler to set base speed. Base rate is the speed at which the motor motion starts and stops. (Bb).

Parameter 1 *b* - Base Speed is pulses per second and has a range of 50 to 5000.

private void setVelocity(int v)

Sends message to handler to set maximum velocity (Mv).

Parameter 1 *v* - Velocity range is 50 to 20,000

private void setHoldTime(int h)

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).

Parameter 1 *h* - value from 0 to 127 representing the number of clock ticks to leave power on the motor after a move.

private void setCrystalFrequence(int cf)

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).

Parameter 1 *cf* - frequence range is 4,000,000 to 8,000.000

private void stopExecution()

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 performSlew()

Slew the motor up to maximum speed and continue until reaching a hard limit switch or receiving a quit (Q) command. (S).

private void setMotorPositive()

Set the motor direction of movement to positive. (+).

private void setMotorNegative()

Set the motor direction of movement to negative. (-).

private void setSteps(int s)

Set the number of steps to move for the G command. (N s).

Parameter 1 *s* - steps range is 0 to 16,777,215

private void setPosition(int p)

Set absolute position to move for the G command. (P p).

Parameter 1 *p* - position range is 0 to 16,777,215

private void go()

Send handler on move (G).

private void join()

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 String verify(char v)

Gives result for wanted registery. (V v).

Parameter 1 *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

Returns returns registery as string

private void setPositionRegister(int r)

Set the position register. This command sets the internal absolute position counter to the value of r. (Z r).

Parameter 1 *r* - position range is 0 to 16,777,215

private char pollMessage()

Poll the device for any waiting messages such as errors or end of move. (

Returns 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

5.2.4 Magnetometer

Offers an interface for controlling the magnetometer."

Package	ikayaki.squid
Declaration	public class Magnetometer
Implements	SerialIOListener
Created by	Squid (5.2.1)
Uses 1	Settings (5.5.1)
Uses 2	SerialIO (5.4.1)
Event A	*On SerialIOEvent* - reads the message and puts it in a buffer

Fields of Magnetometer

private Stack messageBuffer

Buffer for incoming messages, readed when needed.

private String status

Magnetometer’s current status.

private SerialIO serialIO

COM port for communication.

Methods of Magnetometer

public Magnetometer()

Creates a new magnetometer interface. Opens connection to Magnetometer COM port (if its not open already) and reads settings from the Setting class.

public void updateSettings()

Checks which settings have changed and updates the magnetometer interface. This method will be called by the Squid class.

private String reset(char axis)

Reset settings for axis.

Parameter 1 *axis* - x,y,z or a (all). In lower case.

private void resetCounter(char axis)

Reset counter for axis.

Parameter 1 *axis* - x,y,z or a (all). In lower case.

private void configure(char axis, char subcommand, char option)

Used for configuring Magnetometer parameters. See subcommand for usages.

Parameter 1 *axis* - x,y,z or a (all)

Parameter 2 *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)

Parameter 3 *option* - see data values from subcommands.

private void latchAnalog(char axis)

axis is x,y,x or a (all).

Parameter 1 *axis* - x,y,z or a (all). In lower case.

private void latchCounter(char axis)

axis is x,y,x or a (all).

Parameter 1 *axis* - x,y,z or a (all). In lower case.

private String getData(char axis, char command, String datavalues)

Generic send message sender, use with caution and knowledge. Checks if commands are good.

Parameter 1 *axis* - x,y,z. In lower case.

Parameter 2 *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.

Parameter 3 *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?

Returns Returns data wanted, see command and datavalue

public void openLoop(char axis)

Pulse reset and opens feedback loop for axis. Need to be done to all axes before measuring.

Parameter 1 *axis* - x,y,z or a (all). In lower case.

public void clearFlux(char axis)

Clears flux counter for axis. Need to be done to all axes before measuring.

Parameter 1 *axis* - x,y,z or a (all). In lower case.

public void join()

Waits for magnetometer to settle down. Blocking.

public Double[] readData()

Calls first openLoop(a) and clearFlux(a). Latches axes, reads counters and analog. Calculates data from them and returns them.

Returns Returns 3 double values in following order: (x,y,z)

public char[] getFilters()

Returns filter configurations for all axis. Blocking.

Returns 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[] getRange()

Returns range configurations for all axis. Blocking.

Returns	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 boolean getSlew()

Returns Fast Slew options value. Blocking.

Returns	true if Fast Slew is on, false if not

public boolean[] getLoop()

Returns if Loops have been opened on axes. Blocking.

Returns	return Loop status for all axis in order (x,y,z). Values true = on, false = off.

public boolean isOK()

Checks if connection is ok.

Returns	true if ok.

5.3 Squid emulator

5.3.1 SquidEmulator

This class tries to emulate behavior of real squid-system. It starts 3 threads (handler,magnetometer,degausser), 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.

Package	ikayaki.squid
Declaration	public class SquidEmulation
Uses 1	SerialIO (5.4.1)
Event A	*On New IO Message* - reads message and puts it in Buffer

Method	**Page**
writeMessage(String,int)	56
main(String[])	56

Fields of SquidEmulator

private bool online
indicates if system have been started

private File logFile
log file we are using read or write

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

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

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

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

private int commandedDistance
value between 1 and 16,777,215

private int currentPosition
value between 1 and 16,777,215

private int homePosition
value between 1 and 16,777,215

private int commandedRotation
angles are between 0 (0) and 2000 (360)

private int currentRotation
angles are between 0 (0) and 2000 (360)

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

private int degausserAmplitude
0->3000 default amp 0

private int degausserDelay
1-9 seconds default delay 1 second

private int degausserRamp
(3, 5, 7, 9) default 3

private char degausserRamp
Z=Zero, T=Tracking, ?=Unknown

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

private HandlerEmu handler
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 MagnetometerEmu magnetometer
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 DegausserEmu degausser
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.

Methods of SquidEmulator

public void writeMessage(String message ,int port))
send message to SerialIO to be sented.
Parameter 1 *message* - any message reply we are sending back
Parameter 2 *port* - port number to be sent

public static void main(String[] args)
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.

5.4 Serial communication

5.4.1 SerialIO

This class represents hardware layer to serial port communications.

Package	ikayaki.squid
Declaration	public class SerialIO
Implements	SerialPortEventListener
Created by	Squid (5.2.1)
Event A	*On new SerialPortEvent* - generates new SerialMessageArrivedEvent if a data message from serial port is received.

Fields of SerialIO

private String lastMessge

Default value null

contains last received message from the serial port that this SerialIO represents.

Methods of SerialIO

public void SerialIO(SerialParameters parameters)

Creates an instance of SerialIO which represents one serial port.

Parameter 1	*parameters* - parameters for the serial port being opened.
Returns	SerialIO object.
Throws	*NoSuchPortException* - if no such port is found.
Throws	*PortInUseException* - if the serial port is already in use.

public void writeMessage(String message)

Writes an ASCII format message to serial port.

Parameter 1	*message* - message to be send
Throws	*NoSuchPortException* - if no such port is found.
Throws	*PortInUseException* - if serial port is already in use.

public String getLastAnswer()

Writes an ASCII format message to serial port. SerialIO sends and SerialPortEvent if it gets answer to this message.

Returns	last answer received from serial port or null if no last message is available.

private void serialEvent(SerialPortEvent event)

This method is run when a serial message is received from serial port. It generates a new SerialIOEvent.

5.4.2 SerialParameters

Contains all the serial communication parameters which SerialIO uses when opening the port.

Package	ikayaki.squid
Declaration	public class SerialParameters
Created by	Squid (5.2.1)

Method	**Page**
SerialParameters(String,int,int,int,int,int,int)	58

Fields of SerialParameters

private String portName
The name of the serial port.

private int baudRate
The baud rate.

private int flowControlIn
Type of flow control for receiving.

private int flowControlOut
Type of flow control for sending.

private int databits
The number of data bits.

private int stopbits
The number of stop bits.

private int parity
The type of parity.

Methods of SerialParameters

public SerialParameters(String portName, int baudRate, int flowControlIn, int flowControlOut, int databits, int stopbits, int parity)
Creates a SerialParameter object containing settings for serial port communication.

Returns	Parameter settings object.
Parameter 1	*portName* - The name of the serial port.
Parameter 2	*baudRate* - The baud rate.
Parameter 3	*flowControlIn* - Type of flow control for receiving.
Parameter 4	*flowControlOut* - Type of flow control for sending.
Parameter 5	*databits* - The number of data bits.
Parameter 6	*stopbits* - The number of stop bits.
Parameter 7	*parity* - The type of parity.

5.4.3 SerialIOEvent

An event that is generated when SerialIO receives data from serial port.

Package	ikayaki.squid
Declaration	public class SerialIOEvent extends java.util.EventObject
Created by	SerialIO (5.4.1)

Method	**Page**
getMessage()	59

Fields of SerialIOEvent

private String message

ASCII message recieved from serial port.

Methods of SerialIOEvent

public String getMessage()

Returns received serial message.

Returns	The message in ASCII form that was received from serial port.

5.4.4 SerialIOListener

If a class wants to receive SerialIOEvents it must implement this interface.

Package	ikayaki.squid
Declaration	public interface SerialIOListener extends java.util.EventListener

Method	**Page**
serialIOEvent(SerialIOEvent)	59

Methods of SerialIOListener

public void serialIOEvent(SerialIOEvent event)

Propagates serial port message event.

Parameter 1	*event* - the event that happened.

5.5 Global settings

5.5.1 Settings

Singleton class for holding all global settings. All changes are automatically written to file after a short delay.

Package	ikayaki
Declaration	public class Settings
Design patterns	Singleton

Fields of Settings

private Properties properties

Default value new Properties()

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), measurementVelocity(int), transverseYAFPosition(int), axialAFPosition(int), sampleLoadPosition(int), backgroundPosition(int), measurementPosition(int), rotation(int), handlerRightLimit(boolean)

private File propertiesFile

File where the properties will be saved in XML format

private boolean propertiesModified

true if the properties have been modified, otherwise false

private List<MeasurementSequence> sequences

Default value new ArrayList<MeasurementSequence>()

All saved sequences

private File sequencesFile

File where the sequences will be saved in XML format

private bool sequencesModified

true if the sequences have been modified, otherwise false

private LastExecutor autosaveQueue

Queue for scheduling save operations after properties/sequences have been changed

Methods of Settings

public static Settings instance()

Returns the global Settings object. If not yet created, will first create one.

private Settings()
Creates a new Settings instance. Loads settings from the configuration files.

public void save()
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 saveNow()
Saves the settings and keeps waiting until its done. If no settings have been modified, will do nothing.

private String getProperty(String key)
Returns the value that maps to the specified key.
Parameter 1 *key* - key whose associated value is to be returned.
Returns Value associated with key, or an empty string if none exists.

private void setProperty(String key, String value)
Associates the specified value with the specified key. Will invoke autosaving.
Parameter 1 *key* - key with which the specified value is to be associated.
Parameter 2 *value* - value to be associated with the specified key.

public Type getXXX()
Generic accessor for all properties. Returns the value from Properties in appropriate type.
Returns Value associated with key

public boolean setXXX(Type value)
Generic accessor for all properties. Checks whether the value is ok and sets it. Will invoke autosaving.
Returns true if value was correct, otherwise false.

public MeasurementSequence[] getSequences()
Returns all saved Sequences.

public void addSequence(MeasurementSequence sequence)
Adds a sequence to the sequence list.

public void removeSequence(MeasurementSequence sequence)
Removes a sequence from the sequence list. If the specified sequence is not in the list, it will be ignored.

5.6 Utilities

5.6.1 LastExecutor

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.

Package	ikayaki.util
Declaration	public class LastExecutor
Implements	Executor
Uses 1	LastExecutor.LastExecutorThread (5.6.2)
Uses 2	LastExecutor.RunDelayed (5.6.3)
Design patterns	Command

Method	**Page**
LastExecutor()	63
LastExecutor(int)	63
LastExecutor(boolean)	63
LastExecutor(int,boolean)	63
isExecOnlyLast()	63
setExecOnlyLast(boolean)	63
getDelayMillis()	63
setDelayMillis(int)	63
execute(Runnable)	63
join()	64

Fields of LastExecutor

private int delayMillis

Default value 0

Defines how long is the delay in milliseconds, after which the events need to be run.

private boolean execOnlyLast

Default value false

Defines if only the last event should be executed. If false, then all of the events are executed in the order of appearance.

private DelayQueue<RunDelayed> queue

Default value new DelayQueue<RunDelayed>()

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 Thread workerThread

Default value null

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.

Methods of LastExecutor

public LastExecutor()

Creates an empty LastExecutor with a delay of 0 and execOnlyLast set to true.

public LastExecutor(int delayMillis)

Creates an empty LastExecutor with execOnlyLast set to true.

Parameter 1 *delayMillis* - the length of execution delay in milliseconds; if less than 0, then 0 will be used.

public LastExecutor(boolean execOnlyLast)

Creates an empty LastExecutor with a delay of 0.

Parameter 1 *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)

Creates an empty LastExecutor.

Parameter 1 *delayMillis* - the length of execution delay in milliseconds; if less than 0, then 0 will be used.

Parameter 2 *execOnlyLast* - if true, only the last event will be executed after the delay; otherwise all are executed in order of appearance.

public synchronized boolean isExecOnlyLast()

Returns true if only the last task will be executed after the delay; otherwise false.

public synchronized void setExecOnlyLast(boolean execOnlyLast)

Parameter 1 *execOnlyLast* - if true, only the last task will be executed after the delay; otherwise all are executed in order of appearance.

public synchronized int getDelayMillis()

Returns the delay in milliseconds.

public synchronized void setDelayMillis(int delayMillis)

Parameter 1 *delayMillis* - delay in milliseconds; if less than 0, then the new value is ignored.

public synchronized void execute(Runnable command)

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.

Parameter 1 *command* - the runnable task to be executed after a pre-defined delay.

Throws *NullPointerException* - if command is null.

public synchronized void join()

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.

5.6.2 LastExecutor.LastExecutorThread

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.

Package ikayaki.util
Declaration private class LastExecutorThread
Extends Thread
Created by LastExecutor (5.6.1)

Method	Page
run()	64

Methods of LastExecutor.LastExecutorThread

public void run()

5.6.3 LastExecutor.RunDelayed

Wraps a Runnable object and sets the delay after which it should be executed by a worker thread.

Package ikayaki.util
Declaration private class RunDelayed
Implements Delayed
Created by LastExecutor (5.6.1)

Method	Page
RunDelayed(Runnable,int)	65
getDelay(TimeUnit)	65
getRunnable()	65
compareTo(Delayed)	65

Fields of LastExecutor.RunDelayed

private long expires

The point in time when this RunDelayed will expire.

private Runnable runnable

Contained Runnable object to be run after this RunDelayed has expired.

Methods of LastExecutor.RunDelayed

public RunDelayed(Runnable runnable, int delayMillis)

Creates a new RunDelayed item that contains runnable.

Parameter 1	*runnable* - the Runnable to be contained
Parameter 2	*delayMillis* - delay in milliseconds

public long getDelay(TimeUnit unit)

Returns the remaining delay associated with this object, always in milliseconds.

Parameter 1	*unit* - ignored; always assumed TimeUnit.MILLISECONDS
Returns	the remaining delay; zero or negative values indicate that the delay has already elapsed

public Runnable getRunnable()

Returns the contained Runnable.

Returns	the Runnable given as constructor parameter

public int compareTo(Delayed delayed)

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.

Parameter 1	*delayed* - the Delayed to be compared.
Returns	a negative integer, zero, or a positive integer as this delay is less than, equal to, or greater than the specified delay.

6 GUI classes and methods

6.1 Generic GUI components

6.1.1 ProjectComponent

Common superclass for components which use a Project and listen to MeasurementEvents and ProjectEvents.

Package	ikayaki.gui
Declaration	public class ProjectComponent
Extends	JPanel
Created by	MainViewPanel (6.2.2)
Uses 1	Project (5.1.1)
Subclass 1	ProjectInformationPanel (6.6.1)
Subclass 2	MeasurementSequencePanel (6.7.1)
Subclass 3	MeasurementDetailsPanel (6.8.1)
Subclass 4	MeasurementControlsPanel (6.9.1)
Subclass 5	MeasurementGraphsPanel (6.10.1)
Subclass 6	ProjectExplorerPanel (6.4.1)
Subclass 7	CalibrationPanel (6.5.1)
Event A	*On ProjectEvent* - does nothing; subclasses may override this.
Event B	*On MeasurementEvent* - does nothing; subclasses may override this.

Fields of ProjectComponent

private Project project

The active project.

Methods of ProjectComponent

public ProjectComponent()

Initializes this ProjectComponent with no project.

public Project getProject()

Returns the active project, or null if no project is active.

public void setProject(Project project)

Sets the project for this ProjectComponent. Unregisters MeasurementListener and ProjectListener from the old project, and registers them to the new project.

Parameter 1 *project* - new active project, or null to make no project active.

public void projectUpdated(ProjectEvent event)

Does nothing; subclasses override this if they want to listen ProjectEvents.

Parameter 1 *event* - ProjectEvent received.

public void measurementUpdated(MeasurementEvent event)

Does nothing; subclasses override this if they want to listen MeasurementEvents.

Parameter 1 *event* - MeasurementEvent received.

6.2 Main window

6.2.1 Ikayaki

Starts the program. Lays out MainViewPanel, MainMenuBar and MainStatusBar in a JFrame.

Package	ikayaki.gui
Declaration	public class Ikayaki
Extends	JFrame
Uses 1	MainViewPanel (6.2.2)
Uses 2	MainMenuBar (6.2.3)
Uses 3	MainStatusBar (6.2.4)
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.

Method	**Page**
main(String[])	67

Methods of Ikayaki

public static void main(String[] args)

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.

Parameter 1 *args* - command line parameters.

6.2.2 MainViewPanel

Creates the main view panels (split panels) and Squid and Project components. It also tells everybody if the current project is changed.

Package	ikayaki.gui
Declaration	public class MainViewPanel
Extends	JPanel
Created by	Ikayaki (6.2.1)
Uses 1	ProjectExplorerPanel (6.4.1)
Uses 2	CalibrationPanel (6.5.1)
Uses 3	Squid (5.2.1)
Uses 4	MainMenuBar (6.2.3)
Uses 5	MainStatusBar (6.2.4)
Uses 6	ProjectInformationPanel (6.6.1)
Uses 7	MeasurementSequencePanel (6.7.1)
Uses 8	MeasurementDetailsPanel (6.8.1)
Uses 9	MeasurementControlsPanel (6.9.1)
Uses 10	MeasurementGraphsPanel (6.10.1)

Fields of MainViewPanel

private ProjectExplorerPanel projectExplorer

private CalibrationPanel calibration

private Squid squid

private Project project
Currently opened project.

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

private MainMenuBar menuBar

private MainStatusBar statusBar

private ProjectInformationPanel projectInformation

private MeasurementSequencePanel measurementSequence

private MeasurementControlsPanel measurementControls

private MeasurementDetailsPanel measurementDetails

private MeasurementGraphsPanel measurementGraphs

Methods of MainViewPanel

public MainViewPanel()

Loads default view and creates all components and panels. Splitpanel between Calibration,Explorer,Information and rest.

public boolean changeProject(Project project)

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.

6.2.3 MainMenuBar

Creates Menu items for Menubar and makes action listeners for them

Package	ikayaki.gui
Declaration	public class MainMenuBar
Extends	JMenuBar
Created by	MainViewPanel (6.2.2)
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

Method	**Page**
MainMenuBar()	70

Fields of MainMenuBar

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

Methods of MainMenuBar

public MainMenuBar()

Creates all components and makes menu and sets ActionListeners.

6.2.4 MainStatusBar

Creates its components and listens project events on status change and calculates estimated time for measurement

Package	ikayaki.gui
Declaration	public class MainStatusBar
Extends	ProjectComponent
Created by	MainViewPanel (6.2.2)
Event A	*On Measurement Event* - recalculates progress and updates status for current measurement

Method	Page
MainStatusBar()	71
calculateStatus(int,int,int)	71
setMeasurement(int,int[])	71

Fields of MainStatusBar

private JLabel measurementStatus
text comment of current status(moving,measurement,demagnetization)

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

private int[] currentSequence
current projects sequence

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

Methods of MainStatusBar

public MainStatusBar()
Creates all components with default settings and sets Listener for MeasurementEvent.

private void calculateStatus(String phase, int sequenceStep, int currentStep)
Recalculates current progress and updates status.

private void setMeasurement(int projectType, int[] sequence)
Formats status and creates new measurement status values.

6.3 Configuration window

6.3.1 SettingsPanel

Creates its components and updats changes to Settings and saves them in Configuration file

Package	ikayaki.gui
Declaration	public class SettingsPanel
Extends	JFrame
Created by	MainStatusBar (6.2.4)
Uses 1	Settings (5.5.1)
Uses 2	Squid (5.2.1)
Event A	*On Save Clicked* - saves current configuration to Settings-singleton and closes window
Event B	*On Cancel Clicked* - closes window (discarding changes)

Fields of SettingsPanel

private JComboBox magnetometerPort
COM port for magnetometer

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

private JComboBox handlerPort
COM port for sample handler

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

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

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

private JComboBox demagRamp
how fast demagnetization goes

private JComboBox demagDelay
?

private JTextField acceleration
Handler acceleration

private JTextField deceleration
Handler deceleration

private JTextField velocity
Handler Max speed

private JTextField measurementVelocity
speed in measurement, should be small

private JTextField transverseYAFPosition
AF demag position for transverse

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

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

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

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

private JTextField rotation
steps to perform full rotation, must be clockwise, determined by sign

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

private JButton saveButton

private JButton cancelButton

Methods of SettingsPanel

public SettingsPanel()
Creates all components and puts them in right places. Labels are created only here (no global fields). Creates ActionListeners for buttons.

public void closeWindow()
Closes window, no changes saved.

public void saveSettings()
Saves all settings to Settings-singleton and calls closeWindow().

6.4 Project Explorer

6.4.1 ProjectExplorerPanel

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.

Package	ikayaki.gui
Declaration	public class ProjectExplorerPanel
Extends	ProjectComponent
Created by	MainViewPanel (6.2.2)
Uses 1	MainViewPanel (6.2.2)
Uses 2	ProjectExplorerTable (6.4.3)
Uses 3	ProjectExplorerPopupMenu (6.4.4)
Uses 4	NewProjectPanel (6.4.2)
Uses 5	LastExecutor (5.6.1)
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.

Method	**Page**
ProjectExplorerPanel(Project)	75
setProject(Project)	75

Fields of ProjectExplorerPanel

private JComboBox browserField

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 JButton browseButton

private ProjectExplorerTable explorerTable

private NewProjectPanel newProjectPanel

private LastExecutor autocompleteExecutor
Default value new LastExecutor(100, true)
LastExecutor for scheduling autocomplete results to separate thread (disk access and displaying).

private File directory
Default value null
Currently open directory.

private Vector<File> files
Default value new Vector<File>()
Project files in current directory.

Methods of ProjectExplorerPanel

public ProjectExplorerPanel(Project project)
Creates all components, sets *directory* as the last open directory, initializes *files* with files from that directory.

public void setProject(Project project)
Call super.setProject(project), hilight selected project, or unhilight unselected project.

6.4.2 NewProjectPanel

Panel with components for creating a new project. This Panel will be somewhere below the project file listing...

Package	ikayaki.gui
Declaration	public class NewProjectPanel
Extends	JPanel
Created by	ProjectExplorerPanel (6.4.1)
Uses 1	MainViewPanel (6.2.2)
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.

Fields of NewProjectPanel

private JTextField newProject

private JComboBox newProjectType

Default value AF/Thellier/Thermal

private JButton createNewProjectButton

6.4.3 ProjectExplorerTable

Creates a list of project files in *directory*. Handles loading selected projects and showing export popup menu (ProjectExplorerPopupMenu). Inner class of ProjectExplorerPanel.

Package	ikayaki.gui
Declaration	public class ProjectExplorerTable
Extends	JTable
Created by	ProjectExplorerPanel (6.4.1)
Uses 1	MainViewPanel (6.2.2)
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.

Fields of ProjectExplorerTable

private TableModel projectExplorerTableModel

TableModel which handles data from *files* (in upper-class ProjectExplorerPanel). Unnamed inner class.

6.4.4 ProjectExplorerPopupMenu

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.

Package	ikayaki.gui
Declaration	public class ProjectExplorerPopupMenu
Extends	JPopupMenu
Created by	ProjectExplorerPanel (6.4.1)
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

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.

Package	ikayaki.gui
Declaration	public class CalibrationPanel
Extends	ProjectComponent
Created by	MainViewPanel (6.2.2)
Uses 1	MainViewPanel (6.2.2)
Uses 2	Project (5.1.1)
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.

Method	**Page**
setProject(Project)	77

Fields of CalibrationPanel

private JButton calibrateButton

private JTable calibrationProjectTable

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

private TableModel calibrationProjectTableModel

TableModel which holds the data for calibration projects. Unnamed inner class.

Methods of CalibrationPanel

public void setProject(Project project)

Call super.setProject(project), highlight selected calibration project, or unhighlight unselected calibration project.

6.6 Project information

6.6.1 ProjectInformationPanel

Allows inserting and editing project information.

Package	ikayaki.gui
Declaration	public class ProjectInformationPanel
Extends	ProjectComponent
Created by	MainViewPanel (6.2.2)
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.

Fields of ProjectInformationPanel

private JLabel operatorLabel

private JTextFiels operatorTextField

private JLabel dateLabel

private JTextFiels dateTextField

private ButtonGroup measurementType

Groups autoMeasurement and manualMeasurement radiobuttons.

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

private ButtonGroup sampleType
Groups coreSample and handSample radiobuttons.

private JRadioButton coreSample

private JRadioButton handSample

Methods of ProjectInformationPanel

public ProjectInformationPanel()
Creates default ProjectInformationPanel.

private void setProject(Project project)
Calls super.setProject(project) and updates textfield with new projects data.

6.7 Sequence and measurement data

6.7.1 MeasurementSequencePanel

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.

Package	ikayaki.gui
Declaration	public class MeasurementSequencePanel
Extends	ProjectComponent
Created by	MainViewPanel (6.2.2)
Uses 1	MeasurementSequenceTableModel (6.7.2)
Uses 2	MeasurementSequencePopupMenu (6.7.3)
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.

Method	**Page**
MeasurementSequencePanel()	81
addSequence()	81
setProject(Project)	81

Fields of MeasurementSequencePanel

private JButton addSequence

private JComboBox sequenceSelector

private JTextField sequenceStart

private JTextField sequenceStep

private JTextField sequenceStop

private JTable sequenceTable

private MeasurementSequenceTableModel tableModel

Methods of MeasurementSequencePanel

public MeasurementSequencePanel()
Creates default MeasurementSequencePanel.

private void addSequence()
Adds sequence determined by textfields to end of table.

private void setProject(Project project)
Calls super.setProject(project), clears table and calculates shown data from project's measurement data.

6.7.2 MeasurementSequenceTableModel

Handles data in table.

Package	ikayaki.gui
Declaration	public class MeasurementSequenceTableModel
Extends	AbstractTableModel
Created by	MeasurementSequencePanel (6.7.1)

Method	**Page**
MeasurementSequenceTableModel()	82
showColumn(String)	**??**
hideColumn(String)	**??**

Fields of MeasurementSequenceTableModel

private boolean volume
Tells if volume is shown in table.

Methods of MeasurementSequenceTableModel

public MeasurementSequenceTableModel()
Creates SequenceTableModel

public void showColumn(int name)
Shows named column.
Parameter 1 *name* - name of the column to be shown. possible values VOLUME=0

public void hideColumn(int name)
Hides named column.
Parameter 1 *name* - name of the column to be hidden. possible values VOLUME=0

6.7.3 MeasurementSequencePopupMenu

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.

Package	ikayaki.gui
Declaration	public class MeasurementSequencePopupMenu
Extends	JPopupMenu
Created by	MeasurementSequencePanel (6.7.1)

Fields of MeasurementSequencePopupMenu

private JCheckBox volume
If checked volume is shown in measurement sequence table.

private JLabel nameLabel

private JCheckBox nameTextField
Name of the sequence to be saved.

Methods of MeasurementSequencePopupMenu

public MeasurementSequencePopupMenu()
Creates SequencePopupMenu.

private void saveSequence()
Saves whole sequence into dropdown menu.

private void saveSequence()
Saves selected sequence into dropdown menu.

private void removeRows()
Removes selected rows. Rows with measurement data cannot be removed.

6.8 Measurement details

6.8.1 MeasurementDetailsPanel

Shows details of measurement selected in MeasurementSequencePanel.

Package	ikayaki.gui
Declaration	public class MeasurementDetailsPanel
Extends	ProjectComponent
Created by	MainViewPanel (6.2.2)
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.

Fields of MeasurementDetailsPanel

private JTable measurementDetails
X, Y and Z components of BG1, 0, 90, 180, 270, BG2

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

private DefaultTableModel tableMoled

Methods of MeasurementDetailsPanel

public MeasurementDetails()
Creates default MeasurementDetailsPanel.

private void setProject(Project project)
Calls super.setProject(project), clears tables and shows new projects measurement details.

6.9 Measurement controls

6.9.1 MeasurementControlsPanel

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.

Package	ikayaki.gui
Declaration	public class MeasurementControlsPanel
Extends	ProjectComponent
Created by	MainViewPanel (6.2.2)
Uses 1	Project (5.1.1)
Uses 2	MagnetometerStatusPanel (6.9.2)
Uses 3	ManualControlsPanel (6.9.3)
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.

Fields of MeasurementControlsPanel

private JButton measureButton

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

private JButton singlestepButton

private JButton stopButton

private ButtonGroup zButtonGroup

Groups together +z and -z RadioButtons.

private JRadioButton zPlusRadioButton
Changes sample orientation to +Z.

private JRadioButton zMinusRadioButton
Changes sample orientation to -Z.

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

private MagnetometerStatusPanel magnetometerStatusPanel

private ManualControlsPanel manualControlsPanel

6.9.2 MagnetometerStatusPanel

Picture of current magnetometer status, with sample holder position and rotation. Status is updated according to MeasurementEvents received by MeasurementControlsPanel.

Package	ikayaki.gui
Declaration	public class MagnetometerStatusPanel
Extends	JPanel
Created by	MeasurementControlsPanel (6.9.1)

Methods of MagnetometerStatusPanel

public MagnetometerStatusPanel()
Sets magnetometer status to current position.

public void updateStatus(int position, int rotation)
Updates magnetometer status picture; called by MeasurementControlsPanel when it receives MeasurementEvent.

Parameter 1	*position* - sample holder position, from 1 to 16777215.
Parameter 2	*rotation* - sample holder rotation, from 0 (angle 0) to 2000 (angle 360).

6.9.3 ManualControlsPanel

Magnetometer manual controls. MeasurementControlsPanel disables these whenever a normal measurement step is going.

Package	ikayaki.gui
Declaration	public class ManualControlsPanel
Extends	JPanel
Created by	MeasurementControlsPanel (6.9.1)
Uses 1	Project (5.1.1)
Uses 2	Settings (5.5.1)
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().getTransverseYAFPosition().
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.

Fields of ManualControlsPanel

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

private JRadioButton moveHome
Moves sample holder to home position.

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

private JRadioButton moveDemagY
Moves sample holder to demagnetize-Y position.

private JRadioButton moveBG
Moves sample holder to background position.

private JRadioButton moveMeasure
Moves sample holder to measurement position.

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

private JRadioButton rotate0
Rotates sample holder to angle 0.

private JRadioButton rotate90
Rotates sample holder to angle 90.

private JRadioButton rotate180
Rotates sample holder to angle 180.

private JRadioButton rotate270
Rotates sample holder to angle 270.

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

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

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

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

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

6.10 Graphs

6.10.1 MeasurementGraphsPanel

Package	ikayaki.gui
Declaration	public class MeasurementGraphsPanel
Extends	ProjectComponent
Implements	MeasurementListener
Created by	MainViewPanel (6.2.2)
Uses 1	MeasurementStep (5.1.5)

6.10.2 AbstractPlot

Abstract class that implements general construction of a graphical plot.

Package	ikayaki.gui
Declaration	public abstract AbstractPlot
Extends	JPanel
Uses 1	Measurement (**??**)
Subclass 1	StereoPlot (6.10.4)
Subclass 2	IntensityPlot (6.10.3)

Fields of AbstractPlot

private Vector<Measurement> measurement

Default value null

Contains all the data that is shown in this graph.

Methods of AbstractPlot

public void addMeasurement(int declination, int inclination)

Adds new measurement data to plot.

Parameter 1	*declination* - Declination coordinate of the measurement.
Parameter 2	*inclination* - Inclination coordinate of the measurement.

public void highlightMeasurement(int index)

High lights measurement in plot in the given index.

Parameter 1	*index* - Index of measurement to be highlighted.
Throws	*IndexOutOfBoundsException* - If no such measurement existed.

public void highlightMeasurementRange(int from, int to)

Highlights a set of measurements in given range.

Parameter 1	*from* - Starting index of highlighted measurements.
Parameter 2	*to* - End index of highlighted meeasurements.
Throws	*IndexOutOfBoundsException* - If one or both of the indices are out of bounds.

public void unHighlightAll()

dehighlights all values in this graph.

public void resetGraph()

Removes all measurements from the graph.

public int getNumMeasurements()

Returns the number of measurements in this graph.

Returns Number of measurements.

6.10.3 IntensityPlot

Implements intensity graph plot.

Package ikayaki.gui
Declaration public class IntensityPlot
Extends AbstractPlot
Created by MeasurementGraphsPanel (6.10.1)

6.10.4 StereoPlot

Implements stereo graph plot.

Package ikayaki.gui
Declaration public class IntensityPlot
Extends AbstractPlot
Created by MeasurementGraphsPanel (6.10.1)

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.