Design document 0.55

SQUID

Helsinki 17th 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)
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)

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 planned package hierarchy.

Section 8 has bibliography references (which we wont be having any).

1.2 Glossary

Probably dumped again...

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

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 plitted 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.

See Figure 1 and Figure 2.

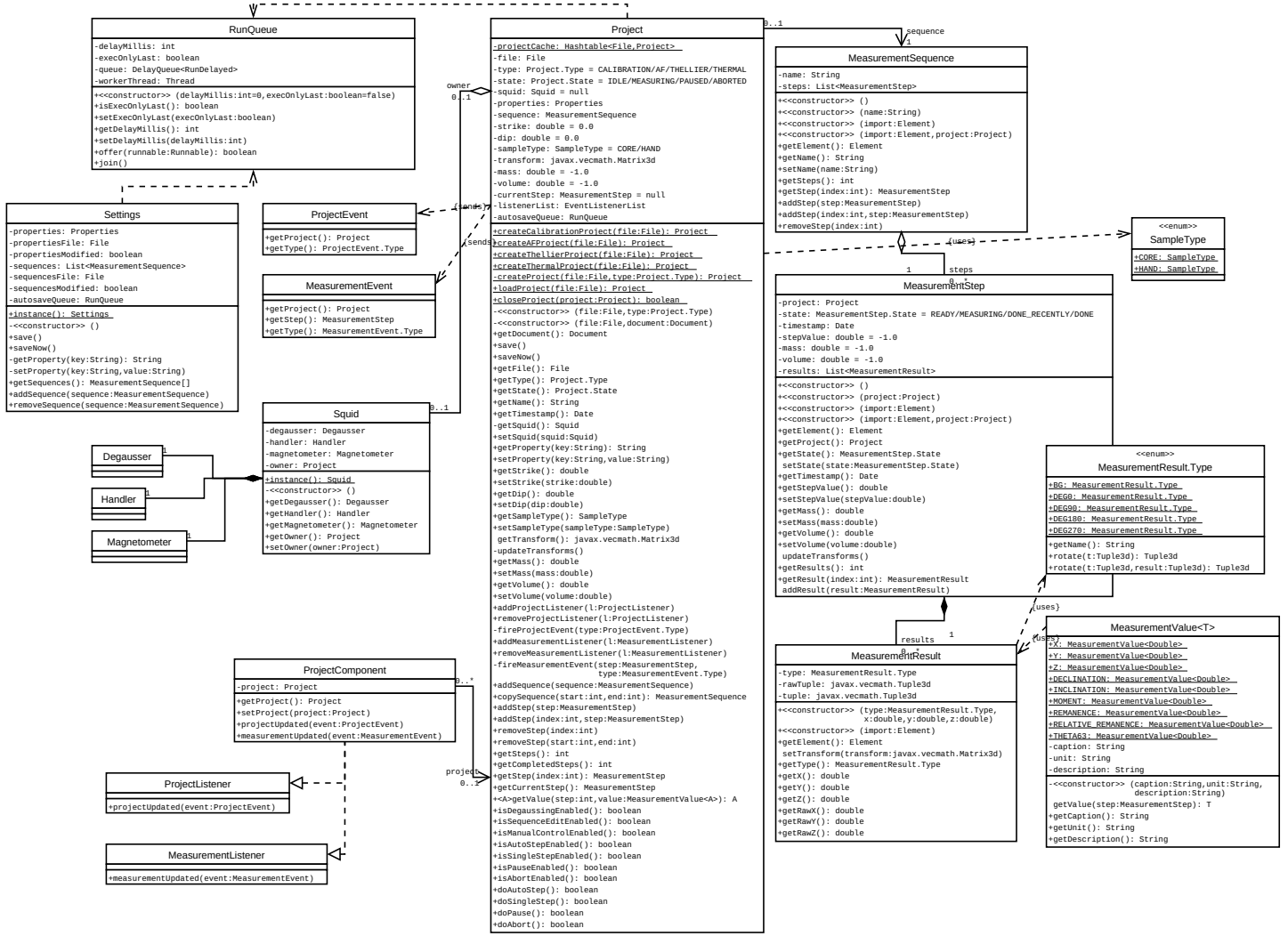

Figure 1: Squid class diagram: central classes

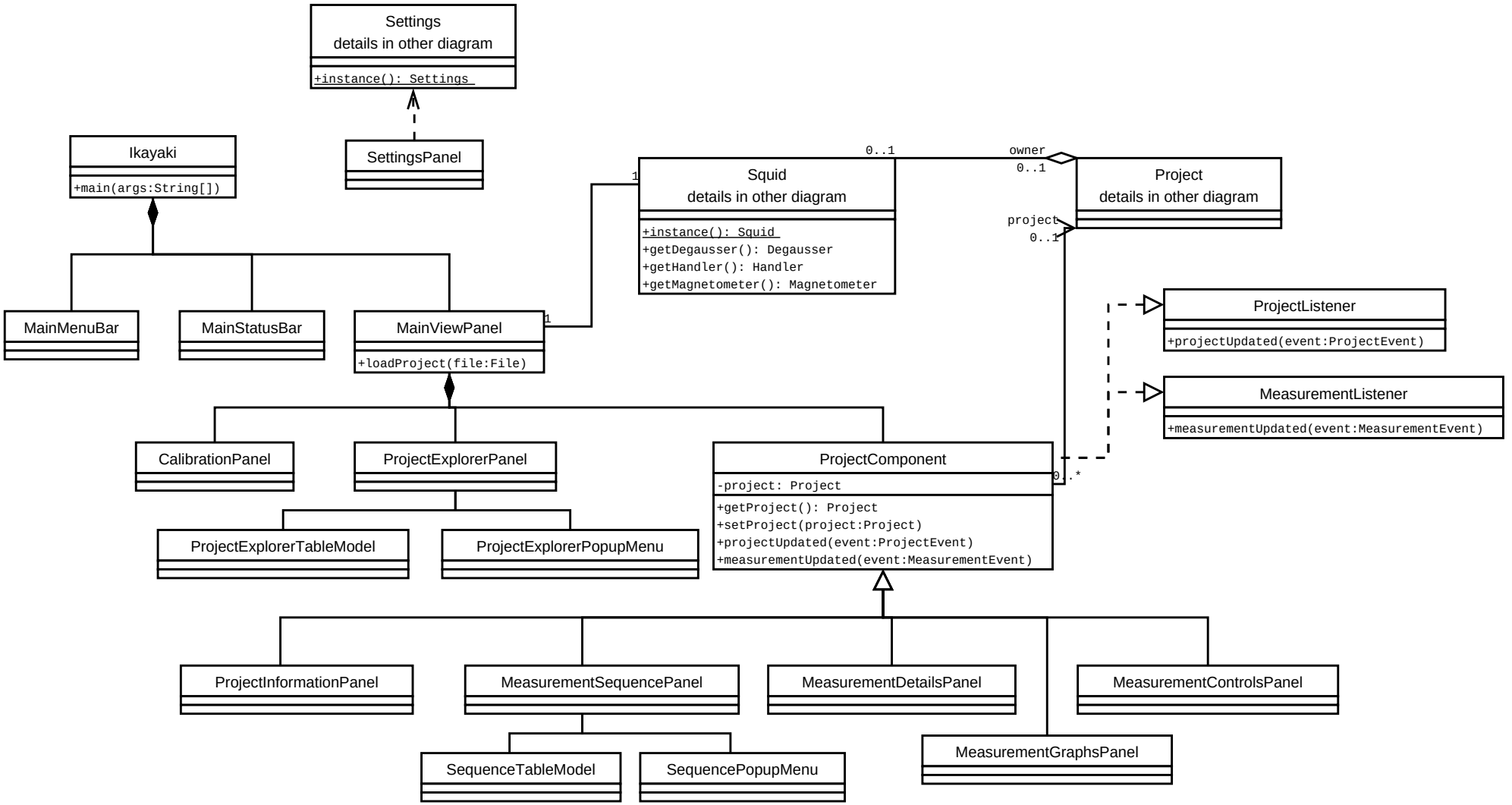

Figure 2: Squid class diagram: GUI classes

4 Architecture description

Is this needed? What would be here anyway? Perhaps some chainsaw-humour...

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	RunQueue (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 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 declination/inclination/volume changed* - The updated transformation matrix will be applied to all measurements and a ProjectEvent 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 Matrix3d transform

Default value new Matrix3d().setIdentity()

Matrix for correcting the sample's orientation. The matrix will be updated whenever the strike, dip or sampleType 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 RunQueue autosaveQueue
Default value new RunQueue(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.

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.

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.

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.

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.

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.

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.

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 *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 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.

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 void 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.

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 void addStep(MeasurementStep step)

Appends a step to this project's sequence. Only the stepValue will be copied from the specified step and added as new steps to this project.

If isSequenceEditEnabled() is false, nothing will be done.

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 project's sequence. Only the stepValue will be copied from the specified step and added as new steps 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.
Throws *IndexOutOfBoundsException* - if the index is out of range (index < getCompletedSteps() || index > getSteps()).
Throws *NullPointerException* - if step is null.

public synchronized void 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.
Throws *IndexOutOfBoundsException* - if the index is out of range (index < getCompletedSteps() || index >= getSteps()).

public synchronized void 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.
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, leave the handler where it was and enable 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.

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>

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)

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.

Package	ikayaki
Declaration	public enum Type

5.1.12 ProjectListener

Defines a listener for project events.

Package	ikayaki
Declaration	public interface ProjectListener
Extends	EventListener

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)

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

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 interfaces and offers handlers 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 Settings 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.

public Handler getHandler()
Returns an interface for controlling the handler.

public Magnetometer getMagnetometer()
Returns an interface for controlling the magnetometer.

public synchronized void updateSettings()
Reloads all settings and initializes all the device interfaces. This method should be called when changes are made to the device parameters.

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 would calls are not made 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 (**??**)
Event A	*On new IO message* - 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 String comPort
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()
Reloads all settings and initializes the degausser interface. This method should be called when changes are made to the device parameters.

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

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

public void executeRampUp()
Performs Ramp up.

public void executeRampDown()
Brings Ramp down.

public void executeRampCycle()
Performs Ramp up and down.

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

public String getStatus()
Gives configuration and ramp status.

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 (**??**)
Event A	*On New IO Message* - reads message and puts it in Buffer

Fields of Handler

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

private String status
Handlers current status

private String comPort
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 Setting class.

public void updateSettings()

Reloads all settings and initializes the handler interface. This method should be called when changes are made to the device parameters.

public String getStatus()

Returns current status on Sample Handler.

public boolean isOK()

checks if connection is ok.

Returns True if ok

public void moveToHome()

Send command to sample holder to move home

public void moveToDegausser()

Send handler to Degauss position

public void moveToMeasurement()

Send handler to Measure position

public void moveToBackground()

Send handler to Background position

public boolean moveToPos(int pos)

Value must be between 1 and 16,777,215. return true if good pos-value and moves handler there.

Parameter 1 *pos* - Position where handler are sent

Returns true if given position was ok, otherwise false.

public void stop()

Tells handler to stop its curren job.

public void rotateTo(int angle)

Rotates the handler to the specified angle.

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

5.2.4 Magnetometer

Offers an interface for controlling the magnetometer.
Commands are at most five characters in length including a carriage return <CR>.
The syntax is as follows: "<device><command><subcommand><data><CR>"

Package	ikayaki.squid
Declaration	public class Magnetometer
Implements	SerialIOListener
Created by	Squid (5.2.1)
Uses 1	Settings (5.5.1)
Uses 2	SerialIO (**??**)
Event A	*On New IO Message* - reads message and puts it in Buffer

Fields of Magnetometer

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

private String status
Magnetometers current status

private String comPort
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()
Reloads all settings and initializes the magnetometer interface. This method should be called when changes are made to the device parameters.

public String reset(char axis)
Reset settings on axis
Parameter 1 *axis* - x,y,x or a (all)

public String resetCounter(char axis)
Reset counter for axis.
Parameter 1 *axis* - x,y,x or a (all)

public String configure(char axis, char subcommand, char option)

Parameter 1 *axis* - x,y,x 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.

public void latchAnalog(char axis)

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

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

public void latchCounter(char axis)

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

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

public void measure(char axis)

Performs full sequence to measure axis.

Parameter 1 *axis* - x,y,x.

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

Generic send message sender, use with cautios and knowledge. Check if commands are good.

Parameter 1 *axis* - x,y,x

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 *axis* - 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?

public String getStatus()
Returns current status on Sample Handler.

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 generates random data 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.. where x is 0 or 1 and indicates if Magnetometer and Demagnetizer are on same COM port. z... values are COM ports.

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

Fields of SquidEmulator

private Stack messageBuffer
buffer for incoming messages

private bool online
indicates if system have been started

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[] messageReader
starts Threads which reads messages from selected COM port

Methods of SquidEmulator

public void getSequences()
reads message and commits it.

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 void run())
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

5.5 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), transverseYPosition(int), axialPosition(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 RunQueue 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 RunQueue

Executes Runnable objects in a private worker thread after a pre-defined delay. The worker thread will terminate automatically when there are no runnables to be executed. Optionally executes only the last inserted runnable. 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 RunQueue
Uses 1	RunQueue.RunQueueThread (5.6.2)
Uses 2	RunQueue.RunDelayed (5.6.3)
Design patterns	Command

Fields of RunQueue

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 RunQueue

public RunQueue()

Creates an empty RunQueue with a delay of 0 and execOnlyLast set to false.

public RunQueue(int delayMillis)

Creates an empty RunQueue with execOnlyLast set to false.

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

public RunQueue(boolean execOnlyLast)

Creates an empty RunQueue 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 RunQueue(int delayMillis, boolean execOnlyLast)

Creates an empty RunQueue.

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 event will be executed after the delay; otherwise false.

public synchronized void setExecOnlyLast(boolean execOnlyLast)

Parameter 1 *execOnlyLast* - if true, only the last event 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 boolean offer(Runnable runnable)

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 *runnable* - the Runnable to be run after a pre-defined delay

Returns true

Throws *NullPointerException* - if runnable 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 RunQueue.RunQueueThread

Keeps on checking the RunQueue.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 RunDelayed.workerThread to null and terminate itself.

Package	ikayaki.util
Declaration	private class RunQueueThread
Extends	Thread
Created by	RunQueue (5.6.1)

Methods of RunQueue.RunQueueThread

public void run()

5.6.3 RunQueue.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	RunQueue (5.6.1)

Fields of RunQueue.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 RunQueue.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.

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 exportToDTDT 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

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

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	RunQueue (5.6.1)
Event A	*On browserField change* - send autocomplete-results-finder with browserField's text to RunQueue via runQueue.offer(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.

Fields of ProjectExplorerPanel

private JComboBox browserField

Text field for writing directory to change to. Autocomplete results appear to Combo Box' popup window, scheduled by RunQueue. 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 RunQueue autocompleteQueue

Default value new RunQueue(100, true)

RunQueue 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 explorerTable 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 explorerTable 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 executes selected.

Package	ikayaki.gui
Declaration	public class ProjectExplorerPopupMenu
Extends	JPopupMenu
Created by	ProjectExplorerPanel (6.4.1)
Event A	*On selectItem mouseEvent* - call project.exportXXX(); 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* - hilight calibration project whose measuring started, or unhilight one whose measuring ended; enable calibrateButton if measuring has ended, or disable if measuring has started.

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), hilight selected calibration project, or unhilight 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.

public ProjectInformationPanel(Project project)
Creates ProjectInformationPanel with information taken from project.

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 painting or ctrl-clicking. Allows dragging rows to different order. 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 end of table 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, add measurement data to appropriate row and if row being measured was selected select next row unless measurement sequence ended.

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.

public MeasurementSequencePanel(Project project)
Creates MeasurementSequencePanel and calculates shown data from project's measurement data. Higlights right row.

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. Higlights right row.

6.7.2 MeasurementSequenceTableModel

Handles data in table.

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

Fields of MeasurementSequenceTableModel

private Vector<TableColumns> shownColumns
Currently shown columns.

private Vector<TableColumns> allColumns
All possible columns.

Methods of MeasurementSequenceTableModel

public MeasurementSequenceTableModel()
Creates SequenceTableModel

public void showColumn(String name)
Shows named column.
Parameter 1 *name* - name of the column to be shown

public void hideColumn(String name)
Hides named column.
Parameter 1 *name* - name of the column to be hidden

6.7.3 MeasurementSequencePopupMenu

Allows selection if volume is shown in table and saving sequence.

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

Fields of MeasurementSequencePopupMenu

private JCheckBox volume

private JLabel nameLabel

private JCheckBox nameTextField

Methods of MeasurementSequencePopupMenu

public MeasurementSequencePopupMenu()
Creates SequencePopupMenu.

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

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.

public MeasurementDetails(Project project)
Creates MeasurementDetailsPanel with measurement details taken from project.

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, 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 ProjectEvent* - update buttons and manual controls according to project.isXXXEnabled().
Event E	*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 JRadioButton zPlusRadioButton

private JRadioButton zMinusRadioButton

private JPanel sampleInsertPanel
Draws a help image for sample inserting.

private MagnetometerStatusPanel magnetometerStatusPanel

private ManualControlsPanel manualControlsPanel

6.9.2 MagnetometerStatusPanel

Picture of current magnetometer status, with sample holder position and rotation.

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 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 control radiobuttons.

Package	ikayaki.gui
Declaration	public class ManualControlsPanel
Extends	JPanel
Created by	MeasurementControlsPanel (6.9.1)
Uses 1	Project (5.1.1)
Event A	*On xxxN click* - call project.xxxN(). Or something like that. I shouldn't care.

Fields of ManualControlsPanel

private JRadioButton demagX

private JRadioButton demagY

private JRadioButton demagZ

private JRadioButton measureX

private JRadioButton measureY

private JRadioButton measureZ

6.10 Graphs

6.10.1 MeasurementGraphsPanel

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

7 Package structure

8 Bibliography