Design document 0.45

SQUID

Helsinki 13th 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)

Contents

1 Introduction

1.1 Meaning and structure of the document

1.2 Glossary

2 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

See Figure 1 and Figure 2.

4 Architecture description

5 Data classes and methods

5.1 Project data

5.1.1 Project

Package	ikayaki
Declaration	public class Project
Created by	ProjectExplorerPanel
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 (5.5.1)
Uses 7	ProjectEvent (5.1.10)
Uses 8	MeasurementEvent (5.1.13)

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.

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.

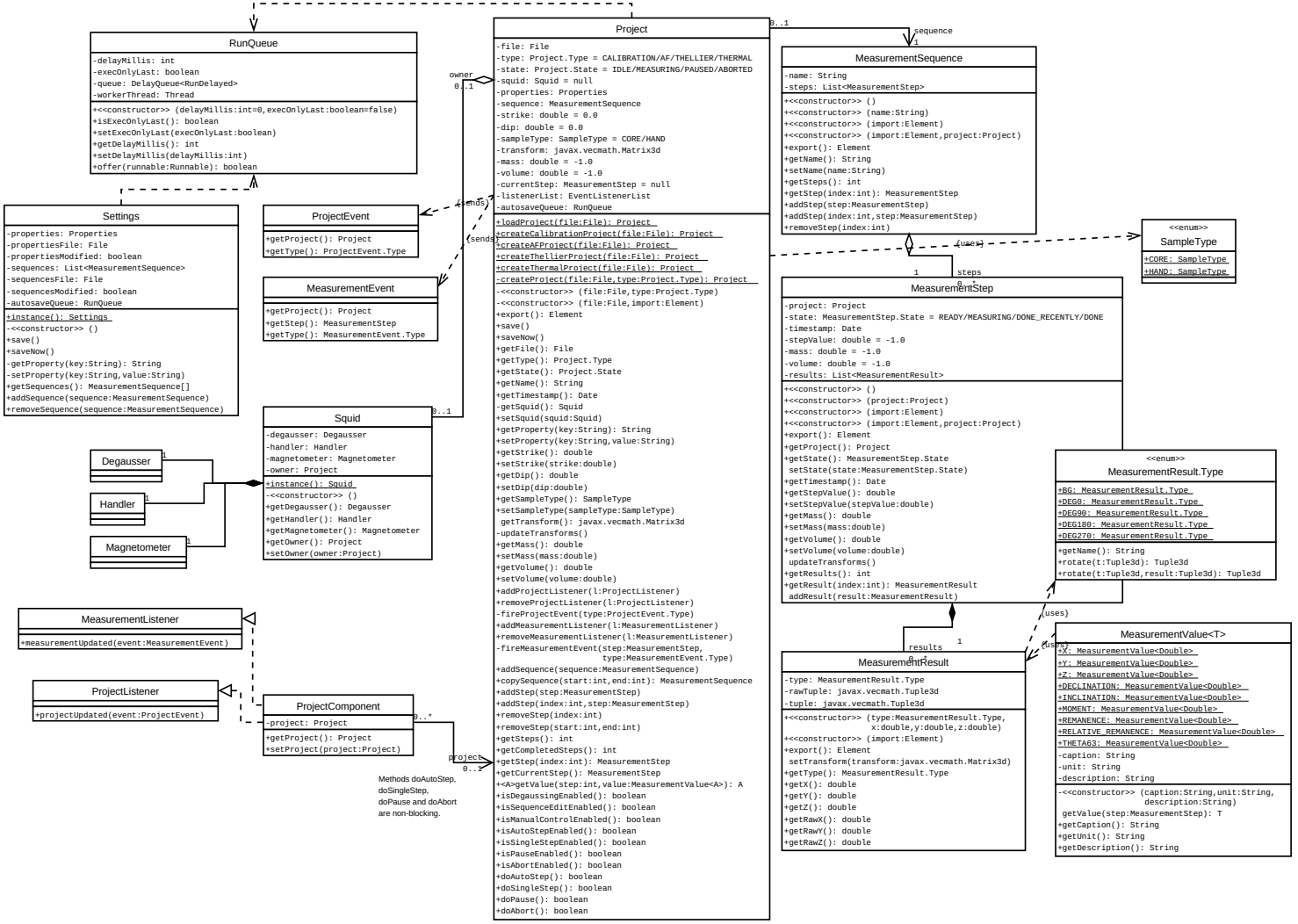
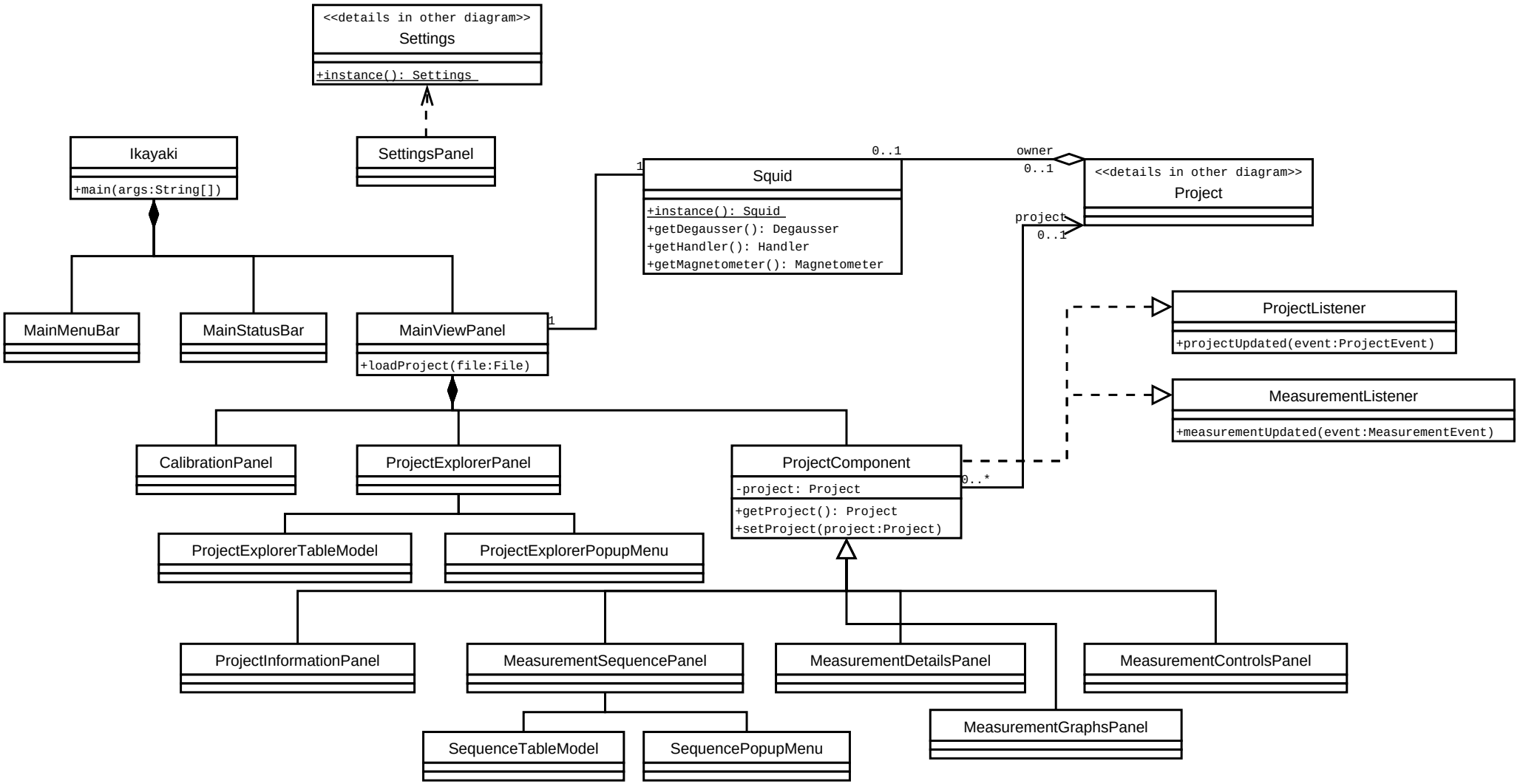


Figure 1: Squid class diagram: central classes


<<details in other diagram>>
Settings
+instance(): Settings
SettingsPanel
Ikayaki
+main(args:String[])
MainMenuBar
MainStatusBar
MainViewPanel
+loadProject(file:File)
1
1
0..1
owner
0..1
Squid
+instance(): Squid
+getDegausser(): Degausser
+getHandler(): Handler
+getMagnetometer(): Magnetometer
<<details in other diagram>>
Project
project
0..1
ProjectListener
+projectUpdated(event:ProjectEvent)
MeasurementListener
+measurementUpdated(event:MeasurementEvent)
CalibrationPanel
ProjectExplorerPanel
ProjectComponent
-project: Project
+getProject(): Project
+setProject(project:Project)
0..*
ProjectExplorerTableModel
ProjectExplorerPopupMenu
ProjectInformationPanel
MeasurementSequencePanel
MeasurementDetailsPanel
MeasurementControlsPanel
MeasurementGraphsPanel
SequenceTableModel
SequencePopupMenu

`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 allowed to access it.

`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()

Matrix for correcting the sample's orientation. The matrix will be updated whenever the strike, dip or sampleType is changed. After that 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.

`public static Project loadProject(File file)`
Loads a saved project file.
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 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.
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.

`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, Element import)`

Creates a new project from the specified import data. This constructor will assume that the specified file is the same from which the import data was read.

Parameter 1	*file* - path for this project file. The file should be the same from which import data was read and be writable, but this constructor will not check it.
Parameter 2	*import* - import data from which this project will be created.
Returns	the created project.
Throws	*IllegalArgumentException* - if the import data was not in the right format.

`public synchronized Element export()`

Exports this project to a DOM element.

`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.

`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, otherwise false (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().

`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. 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(MeasurementListene 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.

`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.

`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.

`public synchronized void addStep(int index, MeasurementStep step)`

Appends a step to this project's sequence to the specified index. 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 order 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()).

`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> value)`

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 *value* - the algorithm for calculating the desired value.

Returns the value returned by the algorithm.

`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()`
Begins an auto step measurement. Will do nothing if isAutoStepEnabled() is false. The measurement will run 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()`
Begins a single step measurement. Will do nothing if isSingleStepEnabled() is false. The measurement will run 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

Package ikayaki
Declaration public enum Type
The type of the project. Options are CALIBRATION, AF, THELLIER and THERMAL.

5.1.3 Project.State

Package ikayaki
Declaration public enum State
The state of the project's measurements. Options are IDLE, MEASURING, PAUSED, ABORTED.

5.1.4 MeasurementSequence

Package ikayaki
Declaration public class MeasurementSequence
Created by Project
Uses 1 MeasurementStep (5.1.5)
A list of measurement steps. Steps can be added or removed from the sequence. All operations are thread-safe.

`private String name`

Default value

`private List<MeasurementStep>`

Default value new ArrayList<MeasurementStep>()

`public MeasurementSequence()`

Returns
Throws -

```
public MeasurementSequence(String name)
```

Parameter 1 *name* -
Returns
Throws -

```
public MeasurementSequence(Element import)
```

Parameter 1 *import* -
Returns
Throws -

```
      public MeasurementSequence(Element import, Project
project)
```

Parameter 1 *import* -
Parameter 2 *project* -
Returns
Throws -

```
public synchronized Element export()
```

Returns
Throws -

```
public synchronized String getName()
```

Returns
Throws -

```
public synchronized void setName(String name)
```

Parameter 1 *name* -
Throws -

```
public synchronized int getSteps()
```

Returns
Throws -

```
public synchronized MeasurementStep getStep(int index)
```

Parameter 1 *index* -
Returns
Throws -

```
public synchronized void addStep(MeasurementStep step)
```

Parameter 1 *step* -
Throws -

```
public synchronized void addStep(int index,
MeasurementStep step)
```

Parameter 1 *index* -
Parameter 2 *step* -
Throws -

```
public synchronized void removeStep(int index)
```

Parameter 1 *index* -
Throws -

5.1.5 MeasurementStep

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

A single step in a measurement sequence. Each step can include multiple measurements for improved measurement 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.

```
private Project project
```

```
private State state
```
Default value READY

```
private Date timestamp
```

```
private double stepValue
```
Default value -1.0

```
private double mass
```
Default value -1.0

```
private double volume
```
Default value -1.0

```
private List<MeasurementResult> results
```
Default value new ArrayList<MeasurementResult>()

```
public MeasurementStep()
```

Returns
Throws -

```
public MeasurementStep(Project project)
```

Parameter 1 *project* -
Returns
Throws -

```
public MeasurementStep(Element import)
```

Parameter 1 *import* -
Returns
Throws -

```
public MeasurementStep(Element import, Project project)
```

Parameter 1 *import* -
Parameter 2 *project* -
Returns
Throws -

```
public synchronized Element export()
```

Returns
Throws -

```
public synchronized Project getProject()
```

Returns
Throws -

```
public synchronized State getState()
```

Returns
Throws -

```
void synchronized setState(State state)
```

Parameter 1 *state* -
Throws -

```
public synchronized Date getTimestamp()
```

Returns
Throws -

```
public synchronized double getStepValue()
```

Returns
Throws -

```
public synchronized void setStepValue(double stepValue)
```

Parameter 1 *stepValue* -
Throws -

```
public synchronized double getMass()
```

Returns
Throws -

```
public synchronized void setMass(double mass)
```

Parameter 1 *mass* -
Throws -

```
public synchronized double getVolume()
```

Returns
Throws -

```
public synchronized void setVolume(double volume)
```

Parameter 1 *volume* -
Throws -

```
synchronized void updateTransforms()
```

Throws -

```
public synchronized int getResults()
```

Returns
Throws -

```
public synchronized MeasurementResult getResult(int
index)
```

Parameter 1 *index* -
Returns
Throws -

```
public synchronized void addResult(MeasurementResult
result)
```

Parameter 1 *result* -
Throws -

5.1.6 MeasurementStep.State

Package ikayaki
Declaration public enum State
The state of a measurement step. Options are READY, MEASURING, DONE_RECENTLY and DONE.

5.1.7 MeasurementResult

Package ikayaki
Declaration public class MeasurementResult
Created by Magnetometer
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.

```
private Type type
```

```
private Tuple3d rawTuple
```
Default value new Tuple3d()

```
private Tuple3d tuple
```
Default value new Tuple3d()

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

Parameter 1 *type* -
Parameter 2 *x* -
Parameter 3 *y* -
Parameter 4 *z* -
Returns
Throws -

```
public MeasurementResult(Element import)
```

Parameter 1 *import* -
Returns
Throws -

```
public Element export()
```

Returns
Throws -

```
void setTransform(Matrix3d transform)
```

Parameter 1 *transform* -
Throws -

```
public Type getType()
```

Returns

```
public double getX()
```

Returns

```
public double getY()
```

Returns

```
public double getZ()
```

Returns

```
public double getRawX()
```

Returns

```
public double getRawY()
```

Returns

```
public double getRawZ()
```

Returns

5.1.8 MeasurementResult.Type

Package ikayaki
Declaration public enum Type
The orientation of the sample when it was measured. Options are BG, DEG0, DEG90, DEG180 and DEG270.

```
public String getName()
```
Returns "BG", "0", "90", "180" or "270"

```
public Tuple3d rotate(Tuple3d t)
```
Rotates the raw XYZ values 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 raw XYZ values 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

Package ikayaki
Declaration public abstract class MeasurementValue<T>
Uses 1 MeasurementStep (5.1.5)
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.
Design patterns Strategy

```
public static final MeasurementValue<Double> X
```

```
public static final MeasurementValue<Double> Y
```

```
public static final MeasurementValue<Double> Z
```

```
public static final MeasurementValue<Double> DECLINATION

public static final MeasurementValue<Double> INCLINATION

public static final MeasurementValue<Double> MOMENT

public static final MeasurementValue<Double> REMANENCE

            public static final MeasurementValue<Double>
 RELATIVE_REMANENCE

public static final MeasurementValue<Double> THETA63

private String caption

private String unit

private String description

    public MeasurementValue(String caption, String unit,
 String description)
```

Parameter 1 *caption* -
Parameter 2 *unit* -
Parameter 3 *description* -
Returns
Throws -

```
abstract T getValue(MeasurementStep step)
```

Parameter 1 *step* -
Returns
Throws -

```
public String getCaption()
```

Returns
Throws -

```
public String getUnit()
```

Returns
Throws -

```
public String getDescription()
```

Returns
Throws -

5.1.10 ProjectEvent

Package ikayaki
Declaration public class ProjectEvent
Extends EventObject
Created by Project
ProjectEvent is used to notify others about the state change of a project.

```
private Project project
```

```
private Type type
```

```
public ProjectEvent(Project project, Type type)
```

Parameter 1 *project* -
Parameter 2 *type* -
Returns
Throws -

```
public Project getProject()
```

Returns
Throws -

```
public Type getType()
```

Returns
Throws -

5.1.11 ProjectEvent.Type

Package ikayaki
Declaration public enum Type
The type of a project event. Options are ...

5.1.12 ProjectListener

Package ikayaki
Declaration public interface ProjectListener
Extends EventListener
Defines a listener for project events.

```
public void projectUpdated(ProjectEvent event)
```

Parameter 1 *event* -

5.1.13 MeasurementEvent

Package ikayaki
Declaration public class MeasurementEvent
Extends EventObject
Created by Project
MeasurementEvent is used to notify listeners about the stages of an ongoing measurement.

```
private Project project


private MeasurementStep step


private Type type


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

Parameter 1 *project* -
Parameter 2 *step* -
Parameter 3 *type* -
Returns
Throws -

```
public Project getProject()
```

Returns
Throws -

```
public MeasurementStep getStep()
```

Returns
Throws -

```
public Type getType()
```

Returns
Throws -

5.1.14 MeasurementEvent.Type

Package ikayaki
Declaration public enum Type
The type of a measurement event. Options are ...

5.1.15 MeasurementListener

Package ikayaki
Declaration public interface MeasurementListener
Extends EventListener
Defines a listener for measurement events.

```
public void measurementUpdated(MeasurementEvent event)
```

Parameter 1 *event* -

5.2 Squid interface

5.2.1 Squid

Package ikayaki.squid
Declaration public class Squid
Extends
Implements
Created by MainViewPanel
Uses 1 Settings (**??**)
Subclass 1 Degausser (5.2.2)
Subclass 2 Handler (5.2.3)
Subclass 3 Magnetometer (5.2.4)
offers Squid interface for project-class that controls SQUID-system. It reads Settings-class for settings and creates classes degausser, handler and magnetometer and offers handlers for them.
Design patterns This class is singleton, there will be always only one instance of it and its created when class is needed first time.

```
private Degausser degausser
```
class for commanding degausser

`private Handler handler`
class for commanding handler

`private Magnetometer magnetometer`
class for commanding magnetometer

`public Degausser getDegausser()`
Gives handler for Degausser.

`public Handler getHandler()`
Gives handler for Degausser.

`public Magnetometer getMagnetometer()`
Gives handler for Magnetometer

`public void updateSettings()`
When settings are saved, update all settings in subclasses.

`private Squid()`
Creates instances of Degausser, Handler and Magnetometer.

`public Settings instance()`
creates Squid, if not yet created, and return it.

5.2.2 Degausser

Package	ikayaki.squid
Declaration	public class Degausser
Created by	Squid
Uses 1	Settings (**??**)
Uses 2	SerialIO (**??**)

Controls Degausser (demagnetizer). Sets it up and offers Interface to control it. 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. Suitable wait loops will have to be used in the external computer code to prevent unreliable communications.

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

`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

`public void setCoil(char coil)`
Sets coil X,Y,Z.

`public void setAmplitude(int amplitude)`
Sets amplitude to ramp, range 0 to 3000.

`public void executeRampUp()`
Performs Ramp up.

`public void executeRampDown()`
Brings Ramp down.

`public void executeRampCycle()`
Performs Ramp up and down.

`public String getStatus()`
Gives configuration and ramp status.

`public void updateSettings()`
Squid tells if settings are changed, update all settings.

`public Degausser()`
Opens connection to Degausser COM port (if not open yet) and read settings to fields from Setting-class.

5.2.3 Handler

Package	ikayaki.squid
Declaration	public class Handler
Created by	Squid
Uses 1	Settings (**??**)
Uses 2	SerialIO (**??**)

Controls Sample Handler and sets it up, offering interface for it.

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

`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)

`public void updateSettings()`
Squid tells if settings are changed, update all settings.

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

`public void moveToHome()`
Send command to sample holder to move home

`public void moveToHome()`
Send handler to Degaus position

`public void moveToMeasurement()`
Send handler to Measure position

`public void moveToBackground()`
Send handler to Background position

`public bool moveToPos(int pos)`
Value must be between 1 and 16,777,215. return true if good pos-value and moves handler there.

`public void stop()`
Tells handler to stop its curren job.

`public void rotateTo(int angler)`
Value is in degrees, remainder of divided by 360. Rotates handler that much.

`public Handler()`
Opens connection to Handler COM port and read settings to fields from Setting-class.

5.2.4 Magnetometer

Package	ikayaki.squid
Declaration	public class Magnetometer
Created by	Squid
Uses 1	Settings (**??**)
Uses 2	SerialIO (**??**)

Controls Magnetometer and sets it up, offering interface for it. Commands are at most five characters in length including a carriage return <CR>. The syntax is as follows: "<device><command><subcommand><data><CR>"

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

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

`private String status`
Magnetometers current status

`private String comPort`
COM port for communication

`public void updateSettings()`
Squid tells if settings are changed, update all settings.

`public String reset(char axis)`
axis is x,y,x or a (all).

`public String resetCounter(char axis)`
axis is x,y,x or a (all).

`public String configure(char axis, char subcommand, char option)`
axis is x,y,x or a (all). 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)

`public void latchAnalog(char axis)`
axis is x,y,x or a (all).

`public void latchCounter(char axis)`
axis is x,y,x or a (all).

`public String getData(char axis, char command, String datavalues)`
axis is x,y or x. 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. 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 Magnetometer()`
Opens connection to Magnetometer COM port (if its not open already) and read settings to fields from Setting-class.

5.3 Squid emulator

5.3.1 SquidEmulator

Package	ikayaki.squid
Declaration	public class SquidEmulation
Extends	Thread
Created by	
Uses 1	SerialIO (**??**)

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.

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

`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

`public void getSequences()`
reads message and commits it.

`public void writeMessage(String,int))`
send message to SerialIO to be sented.

`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 Utilities

5.5.1 RunQueue

Package ikayaki.util
Declaration public class RunQueue
Uses 1 RunQueue.RunQueueThread (5.5.2)
Uses 2 RunQueue.RunDelayed (5.5.3)
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.
Design patterns Command

`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.

`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

5.5.2 RunQueue.RunQueueThread

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

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.

```
public void run()
```

5.5.3 RunQueue.RunDelayed

Package ikayaki.util
Declaration private class RunDelayed
Implements Delayed
Created by RunQueue

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

```
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.

```
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

Package	ikayaki.gui
Declaration	public class ProjectComponent
Extends	JPanel
Created by	MainViewPanel
Uses 1	Project (5.1.1)
Subclass 1	ProjectInformationPanel (**??**)
Subclass 2	MeasurementSequencePanel (**??**)
Subclass 3	MeasurementDetailsPanel (**??**)
Subclass 4	MeasurementControlsPanel (6.9.1)
Subclass 5	MeasurementGraphsPanel (**??**)
Subclass 6	ProjectExplorerPanel (6.4.1)
Subclass 7	CalibrationPanel (6.5.1)

Generic gui component which uses Project and listens MeasurementEvents and ProjectEvents.

`private Project project`

Active project.

`public ProjectComponent()`

Initializes this ProjectComponent with no Project (one probably arrives shortly with setProject(Project)).

`public Project getProject()`

Returns	this.project.

`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.

6.2 Main view and menu

6.2.1 MainViewPanel

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

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

```
private ProjectExplorerPanel projectExplorer
```

```
private CalibrationPanel calibration
```

```
private Squid squid
```

```
private ProjectComponent project
```

currently active project

```
private MainMenuBar menuBar
```

```
private MainStatusBar statusBar
```

```
private ProjectInformationPanel projectInformation
```

```
private MeasurementSequencePanel measurementSequence
```

```
private MeasurementControlsPanel measurementControls
```

```
private MeasurementDetailsPanel measurementDetails
```

`private MeasurementGraphsPanel measurementGraphs`

`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.

6.3 Configuration window

6.3.1 SettingsPanel

Package	ikayaki.gui
Declaration	public class SettingsPanel
Extends	JFrame
Created by	MainStatusBar
Uses 1	Settings (**??**)
Uses 2	Squid (5.2.1)

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

Event A	*On Save Clicked* - saves current configuration to Settings-singleton and closes window
Event B	*On Cancel Clicked* - closes window (discarding changes)

`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`

`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

Package	ikayaki.gui
Declaration	public class ProjectExplorerPanel
Extends	ProjectComponent
Created by	MainViewPanel
Uses 1	MainViewPanel (6.2.1)
Uses 2	ProjectExplorerTableModel (6.4.3)
Uses 3	ProjectExplorerPopupMenu (6.4.5)
Uses 4	NewProjectPanel (6.4.4)

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.

Event A	*On browserField change* - send browserField's text to RunQueue which schedules disk access and autocomplete.
Event B	*On browserHistoryButton click* - show browserAutocompletePanel with directory history.
Event C	*On browseButton click* - open a FileChooser dialog for choosing new directory and tell explorerTable and browserField to change to that directory.
Event D	*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. (This could be in ProjectExplorerTableModel class?)
Event E	*On explorerTable mouse right-click* - create a ProjectExplorerPopupMenu for right-clicked project file. (This could be in ProjectExplorerTableModel class?)
Event F	*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.
Event G	*On ProjectEvent* - hilight project whose measuring started, or unhilight one whose measuring ended.

`private BrowserFieldPanel browserFieldPanel`

`private JButton browseButton`

`private ProjectExplorerTableModel explorerTable`

`private NewProjectPanel newProjectPanel`

`public ProjectExplorerPanel(Project project)`
Creates all components, sets the last open project folder as current project folder.

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

6.4.2 BrowserFieldPanel

Package	ikayaki.gui
Declaration	public class BrowserFieldPanel
Extends	JPanel
Created by	ProjectExplorerPanel

Panel with browser text field and history-down-arrow-button.

`private JTextField browserField`
Text field for writing directory to change to.

`private JButton browserHistoryButton`
Small down-arrow which shows directory history when clicked.

`private JPanel browserAutocompletePanel`
Panel whichs appears when autocomplete results are to be shown, or, when history-downarrow is clicked.

`private JList browserAutocompleteList`
List which holds autocomplete/history results.

6.4.3 ProjectExplorerTableModel

Package ikayaki.gui
Declaration public class ProjectExplorerTableModel
Extends AbstractTableModel
Created by ProjectExplorerPanel
Uses 1 MainViewPanel (6.2.1)
Creates a list of project files in directory. Handles loading selected projects and executing export choice? What the hell anyway, I'm confused.. *whacks some stupid monsters with a chainsaw* NNNNnnnnnnnnrrnnnnnnnn-splat-grah-blurts-blorp-stuff-guttering

```
private File directory
```
Default value null
Currently opened directory

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

```
public ProjectExplorerTableModel()
```
Reads the contents of the default directory and initializes the file list.

```
public ProjectExplorerTableModel(String directory)
```
Reads the contents of the given directory and initializes the file list.
Parameter 1 *directory* - path to the directory to be opened

```
public void loadDirectory(String directory)
```
Update list to show given directory.
Parameter 1 *directory* - path to the directory to be opened

```
public void loadProject()
```
Gets the selected line from the file list and sends the file to MainViewPanel.

6.4.4 NewProjectPanel

Package ikayaki.gui
Declaration public class NewProjectPanel
Extends JPanel
Created by ProjectExplorerPanel
Panel with components for creating a new project. This Panel will be somewhere below the project file listing...

```
private JTextField newProject
```

```
private JComboBox newProjectType
```
Default value AF/Thellier/Thermal

```
private JButton createNewProjectButton
```

6.4.5 ProjectExplorerPopupMenu

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

Shows choices to export: AF, Thellier, Thermal and executes selected

Event A	*On selectItem mouseEvent* - tells selected Project to create selected file from it's self

```
public ProjectExplorerTableModel()
```

Builds the menu items

6.5 Calibration

6.5.1 CalibrationPanel

Package	ikayaki.gui
Declaration	public class CalibrationPanel
Extends	ProjectComponent
Created by	MainViewPanel
Uses 1	MainViewPanel (6.2.1)
Uses 2	Project (5.1.1)

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.

Event A	*On calibrateButton click* - call project.doSingleStep().
Event B	*On calibrationProjectTable 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 calibrationProjectTable selection to old project, if any.
Event C	*On ProjectEvent* - hilight calibration project whose measuring started, or unhilight one whose measuring ended; enable/disable calibrateButton similarly.

```
private JButton calibrateButton
```

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

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

6.6 Project information

6.7 Sequence and measurement data

6.8 Measurement details

6.9 Measurement controls

6.9.1 MeasurementControlsPanel

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

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.

Event A	*On measureButton click* - call project.doAutoStep() or project.doPause(), depending on current button status.
Event B	*On singlestepButton click* - call project.doSingleStep().
Event C	*On stopButton click* - call project.doAbort().
Event D	*On MeasurementEvent* - call magnetometerStatusPanel.updateStatus(int, int).
Event E	*On ProjectEvent* - update buttons and manual controls according to project.isXXXEnabled().

`private JButton measureButton`

`private JButton singlestepButton`

```
private JButton stopButton

private JRadioButton zPlusRadioButton

private JRadioButton zMinusRadioButton

private JPanel sampleInsertPanel
```
Draws the help image for sample inserting.

```
private MagnetometerStatusPanel magnetometerStatusPanel

private ManualControlsPanel manualControlsPanel
```

6.9.2 MagnetometerStatusPanel

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

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

```
public MagnetometerStatusPanel()
```
Sets magnetometer status to current position.

```
public updateStatus(int position, int rotation)
```
Updates magnetometer status picture; called by MeasurementControlsPanel.

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

Package	ikayaki.gui
Declaration	public class ManualControlsPanel
Extends	JPanel
Created by	MeasurementControlsPanel

Magnetometer manual control radiobuttons.

Uses 1	Project (5.1.1)
Event A	*On xxxN click* - call project.xxxN().

```
private JRadioButton demagX
```

```
private JRadioButton demagY

private JRadioButton demagZ

private JRadioButton measureX

private JRadioButton measureY

private JRadioButton measureZ
```

6.10 Status bar

6.10.1 MainStatusBar

Package	ikayaki.gui
Declaration	public class MainStatusBar
Extends	ProjectComponent
Created by	MainViewPanel

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

Event A	*On Measurement Event* - recalculates progress and updates status for current measurement

`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)

`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.

6.11 Graphs

7 Package structure

8 Bibliography