

LIISim 3

Developer guide

Version 3.0.6

April 2018
(03.04.2018)

Authors:	Raphael Mansmann Philip Schmidt Tobias Terheiden
Website:	www.liisim.com www.uni-due.de/ivg/rf
Source code:	www.github.com/LIISim/LIISim3

LIISim is a project of:



Table of Contents

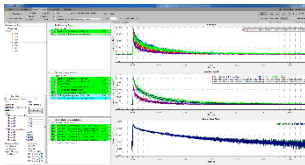
1	Introduction.....	1
1.1	Laser-induced incandescence.....	1
1.2	How to cite	2
1.3	Motivation	2
1.4	History	2
1.5	License	3
2	Needed libraries and software	4
2.1	Tested versions.....	4
2.2	Where to place	5
2.2.1	Boost.....	5
2.2.2	QWT	5
2.2.3	PicoScope SDK	6
2.2.4	NI DAQmx SDK.....	6
2.2.5	MATio and zlib	6
3	Directory structure	7
4	Basic calculation classes	8
4.1	Add your own heat-transfer model.....	8
5	Getting started	9
5.1	Compiler / development environment installation.....	9
5.1.1	Installing the Qt framework / Qt Creator	9
5.2	“Kits” / Configuring Qt Creator with multiple compilers	10
5.2.1	Adding another compiler / architecture manually.....	10
5.3	Open, configure and build LIISim in Qt Creator	13
5.4	Switching between lite and full version	15
6	Building a release	17
7	Class overview	20
7.1	Main.....	20
7.2	Core	20
7.2.1	SettingsBase	20
7.2.2	DatabaseManager	21
7.2.3	DataModel.....	21
7.2.4	SignalManager	21
7.2.5	HeatTransferModels.....	22
7.3	MasterWindow.....	22
7.3.1	DatabaseWindow	22

7.3.2	AnalysisTools	22
7.3.3	UI Elements	23
8	Troubleshooting	24
9	Frequently Asked Questions.....	24

1 Introduction

Welcome to LIISim3, a modular signal processing toolbox for time-resolved laser-induced incandescence measurements written in C++ with the Qt-Framework. This user guide describes the main functionalities of the software and how to use it. Please feel free to contact the authors if you have questions, find errors or have any other feedback.

Some screenshots of the software:

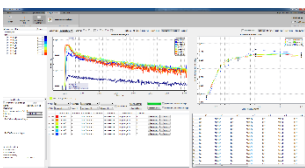
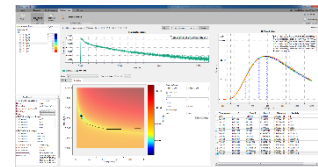


SignalProcessing

The modular signal processing toolbox allows processing of raw signals, absolute signals and temperature traces. Processing steps can be individually set and arranged. Intermediate processing results can be visualized and analyzed with various plot tools.

AnalysisTool: Temperature Fit

Visualization of spectral temperature fitting using Planck's law. Temperature traces that are calculated in the SignalProcessing module can be analyzed and all fitting iterations can be visualized.

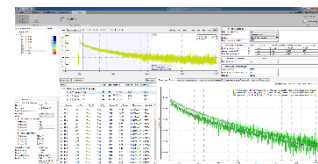


AnalysisTool: Parameter Analysis

Experimental data can be systematically compared for various parameters (i.e., laser fluence, LII peak temperature, PMT gain voltage, ...).

FitCreator

Select from a variety of heat transfer models and databases for simulation of LII signal traces and comparison with experimental data.



1.1 Laser-induced incandescence

Laser-induced incandescence (LII) is a non-intrusive method of measuring soot particle volume fraction and primary particle sizes in flames. Not only restricted to flames, this technique is recently used for characterization of car engine exhaust, atmospheric black carbon and synthetic nanoparticles.

These new applications require the same basic signal processing, but at the same time individual databases for material (soot or non-soot) and gas compositions as well as individual heat transfer models for the determination of particle sizes.

LIISim is a framework for basic signal processing and analysis experimental LII data to help researches across the world to compare their experimental data.

1.2 How to cite

If you use our software for your research, we would be grateful if you could cite the following paper:

R. Mansmann, T. Terheiden, P. Schmidt, J. Menser, T. Dreier, T. Endres and C. Schulz: "LIISim: a modular signal processing toolbox for laser-induced incandescence measurements" Appl. Phys. B, DOI 10.1007/s00340-018-6934-9 (2018)

View at publisher:

<https://doi.org/10.1007/s00340-018-6934-9>

1.3 Motivation

LIISim is designed to provide transparent and flexible tools, which allow individual setting of processing parameters, visualization of intermediate processing steps, and comparison of multiple experimental data sets.

Main features:

- Modular choice of user-defined material properties (soot, silicon, germanium, ...)
- Comparison of experimental data with different pre-implemented heat-transfer models
- Various analysis tools help visualizing dependencies between experimental data sets (plots, parameter comparison, statistical information)
- Easy-to-use import (TXT/CSV) and export (TXT/CSV/MATLAB) functionalities
- Copy/paste of signals and analysis results into spreadsheet software (MATLAB, Excel, Origin, ...)
- Software architecture allows processing and comparison of large data sets (> 4 GB)

1.4 History

LIISim was so far known as **console application (LIISim 1.5)** and **web interface (www.liisim.com)**, developed by Max Hofmann between 2001 and 2007. The console application was written in C and the web interface was based on Perl. Modeling settings and file names for experimental data could be defined in DAT files and after execution of the console application, DAT output files containing the modeling results are created.

Later the console application was extended by a C++ based graphical user interface (GUI) by Tobias Terheiden and Martin Leschowski, which allowed the visualization of the modeling results from the DAT files (**LIISim Desktop 1.02** and **LIISim Console 2.14**). While the console application was distributed among the community, the desktop version was not published and was only internally used.

In 2013 the development of **LIISim 3** a new framework with object-oriented structure based on C++ was started by Raphael Mansmann designed for the application on individual material systems (soot and non-soot) with a modular implementation of signal processing steps and choice of heat-transfer models.

1.5 License

LISim is free software: You can redistribute it and/or modify it under the terms of the **GNU General Public License** as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

LISim is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License (<http://www.gnu.org/licenses/>) for more details.

2 Needed libraries and software

LIISim is based on the Qt framework, hence before building, the Qt framework should be installed just as Visual Studio if you have chosen MSVC as compiler. See section 5 for further information on installing the Qt framework.

These are the libraries needed for building LIISim from source:

- Boost (<http://www.boost.org/>)
- QWT (<http://qwt.sourceforge.net/>; needs to be compiled from source)
- MATio (<https://github.com/tbeu/matio>) and zlib (<http://zlib.net/>) (have to be compiled into libraries before use)

If you like to compile the LIISim with the data acquisition module enabled, you additionally need the following package (and a PicoScope 6000-series oscilloscope):

- PicoScope SDK (<https://www.picotech.com/>)

If you like to use the NI DAQmx support inside the data acquisition module, you additionally need the following package:

- NI DAQmx SDK (<http://www.ni.com/>)

Note that the DAQmx support inside LIISim is limited, so it is likely that you need to modify the code according to your devices.

2.1 Tested versions

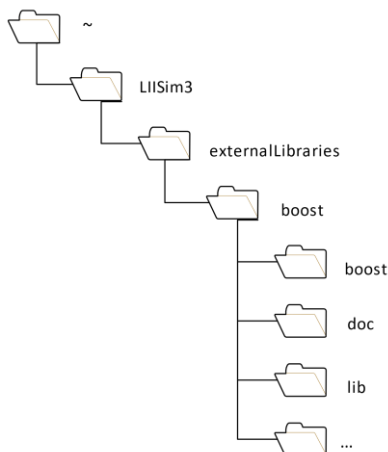
Library	Version
Qt	5.10.1
Boost	1.65.1
QWT	6.1.3
PicoScope SDK	10.6.10b
NI DAQmx SDK	15.5.0.31
MATio	1.5.11
zlib	1.2.11

2.2 Where to place

Note: If you want to change any standard path, edit the path settings in the LIISim3.pro under the “EXTERNAL LIBRARY PATHS” section.

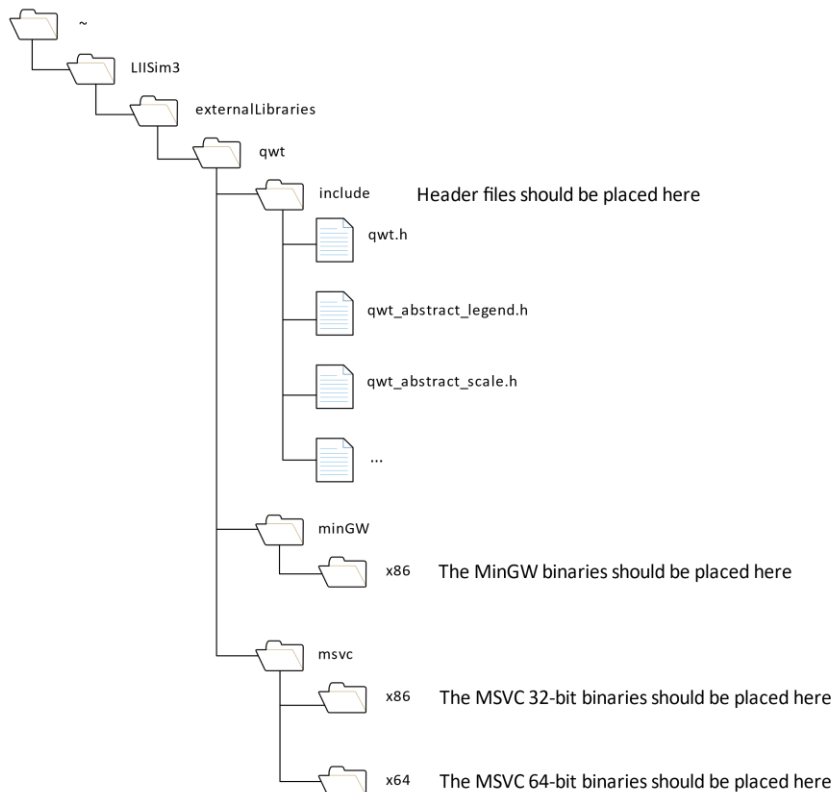
2.2.1 Boost

The extracted library folder (remove any version naming from the folder name) should be placed in the “externalLibraries” folder. The folder structure should look something like this:



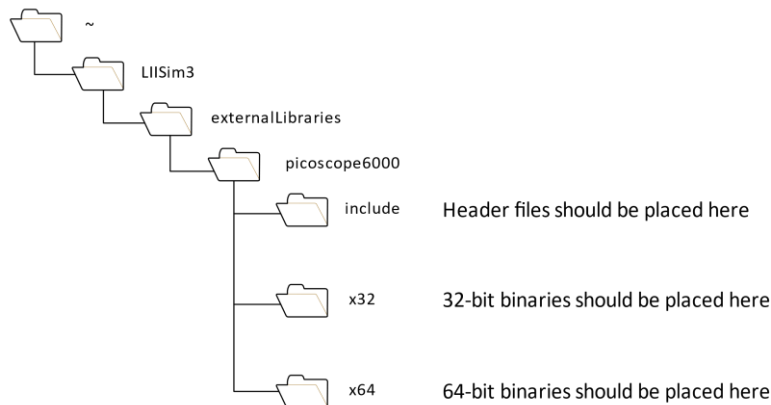
2.2.2 QWT

The compiled binaries and headers should be placed in the following folder structure:



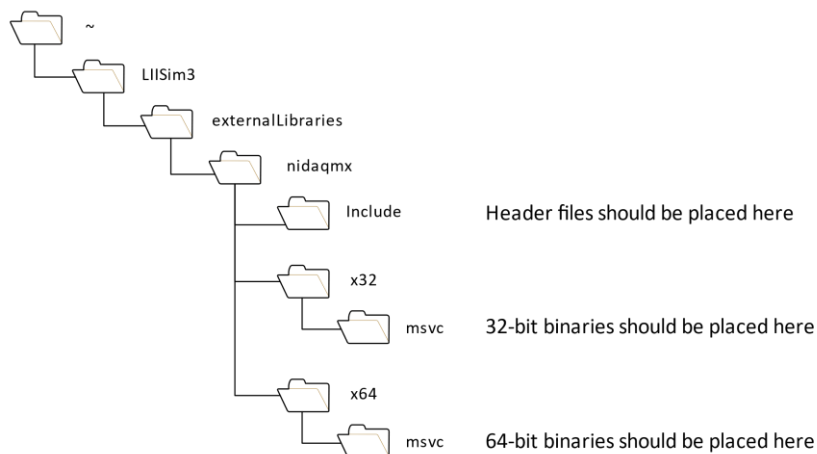
2.2.3 PicoScope SDK

Copy the libraries and headers from the installed SDK into the following folder structure:



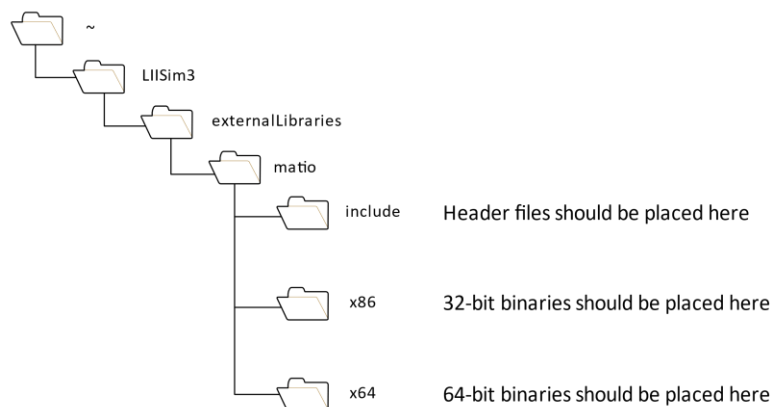
2.2.4 NI DAQmx SDK

Copy the libraries and headers from the installed SDK into the following folder structure:



2.2.5 MATio and zlib

Copy the compiled libraries and header into the following structure:



3 Directory structure

The following list explains the directory structure of the source code. Root directory is LIISim.

calculations/		Classes needed for different calculations
	fit/	Fitting-related classes
	models/	Heat transfer model classes
data/	database/	Database directory where Gases, Materials, LIISettings etc. reside
database/		Database base classes, database manager
	structure/	Classes for different database elements
exampleData/		Directory containing example data files
externalLibraries/		Libraries needed for compilation
general/		Commonly used classes which do not fit in any special category
gui/		UI-related classes
	analysisTools/	Analysis tools and related UI elements
	dataAcquisition/	Classes for the data acquisition window
	databaseEditor/	Editor for database elements
	dataItemViews/	Treeview UI element
	deviceManager/	Manager used for the DAQmx devices in the data acquisition
	fitTools/	UI classes related to the fitting tools
	settingsWidgets/	UI widgets for settings
	signalEditor/	UI classes for the signal editor
	utils/	Commonly used UI elements
io/		Classes for loading/saving files and accessing devices
logging/		Logging related classes
models/		Data model classes
resources/		External resources, e.g. icons, equation-files, stylesheets
settings/		Classes managing settings
signal/		Classes handling signal data
	processing/	Processing base classes

4 Basic calculation classes

The following classes build the foundation for all calculations:

constants.cpp	This class contains numeric constants used in calculations.
heattransfermodels.cpp	This class is the base class for all heat transfer models, present in the 'models' directory.
numeric.cpp	Contains the numerical calculation routines.
temperature.cpp	Functions used in the temperature calculation.

4.1 Add your own heat-transfer model

The implemented heat transfer models can be found in the 'models' directory. If you want to implement your own model, copy the .cpp and .h files of one of the other models and modify the following locations:

a) LIISim3.pro

add the class files:

```
SOURCES += \
calculations/models/htm_<NEWNAME>.cpp \
```

and

```
HEADERS += \
calculations/models/htm_<NEWNAME>.h \
```

b) core.cpp

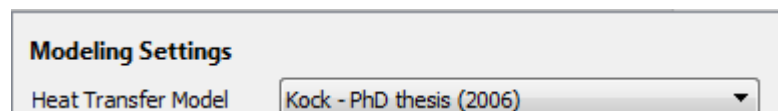
```
#include "calculations/models/htm_melton.h"
#include "calculations/models/htm_kock_soot.h"
#include "calculations/models/htm_liu.h"
#include "calculations/models/htm_<NEWNAME>.h"
```

and

```
// setup heat transfer model list
HTM_Melton      *h1 = new HTM_Melton;
HTM_KockSoot    *h2 = new HTM_KockSoot;
HTM_Liu         *h3 = new HTM_Liu;
HTM_<NEWNAME>   *h4 = new HTM_<NEWNAME>;

heatTransferModels.push_back( h1 );
heatTransferModels.push_back( h2 );
heatTransferModels.push_back( h3 );
heatTransferModels.push_back( h4 );
```

After compilation, the new heat transfer model will be listed in the modeling settings of the FitCreator:



5 Getting started

5.1 Compiler / development environment installation

The Qt Creator comes in different flavors, either with MinGW as compiler combined (32-bit only), or for use together with Visual Studio (MSVC) in 32- and 64-bit versions. The Visual Studio versions are recommended, as they are overall faster in compilation and execution time, come in a 64-bit version, but Visual Studio (at least version 2013 and upwards) must **be installed prior to using Qt Creator**.

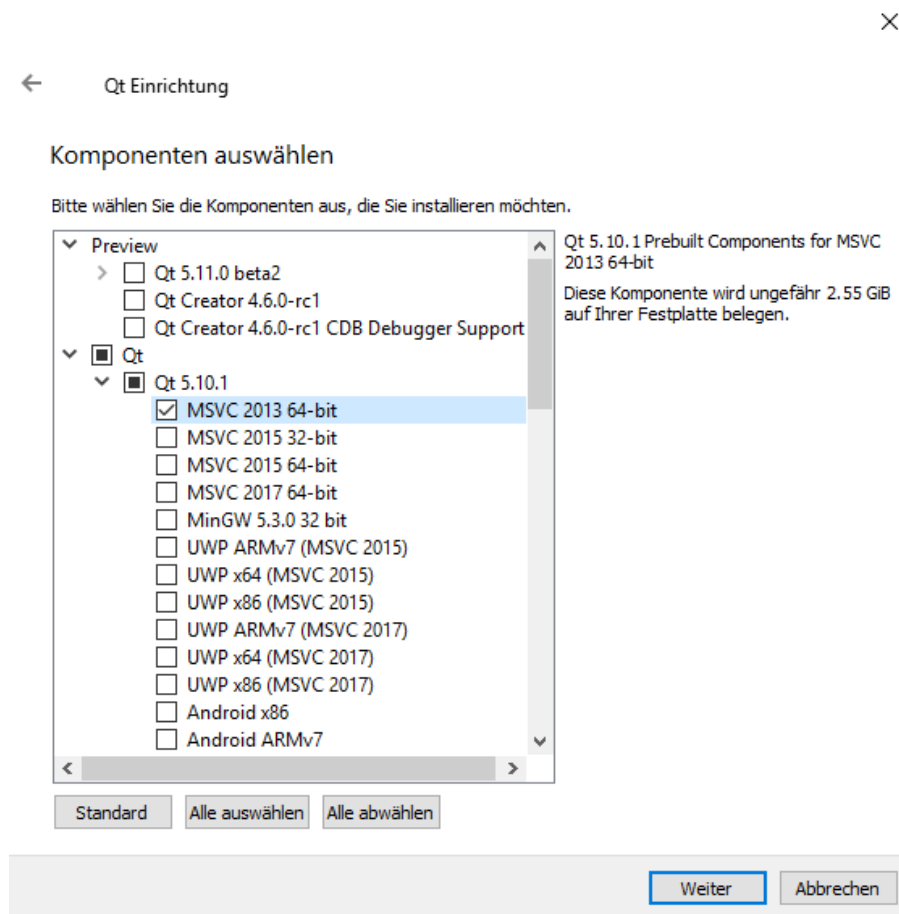
The Qt-versions used in this guide are Qt 5.10.1 and Qt Creator 4.6.0. All screenshots are based on these versions.

To download Qt Creator, go to <https://www.qt.io/download/> and follow the dialogs.

5.1.1 Installing the Qt framework / Qt Creator

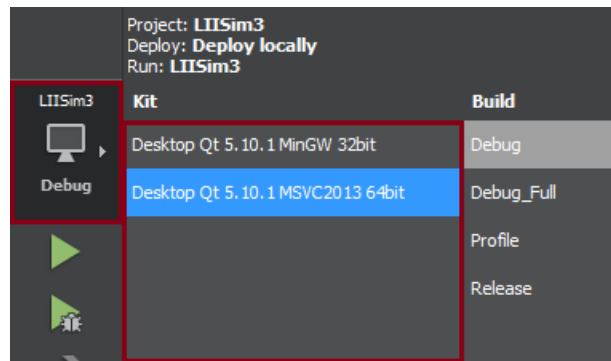
If you want to use only a single Qt version / compiler version, you can just follow the default installation process.

Select the compiler you like to use when the following screen comes up (in this example Visual Studio 2013 64-bit is used as compiler, but you can use multiple compilers):



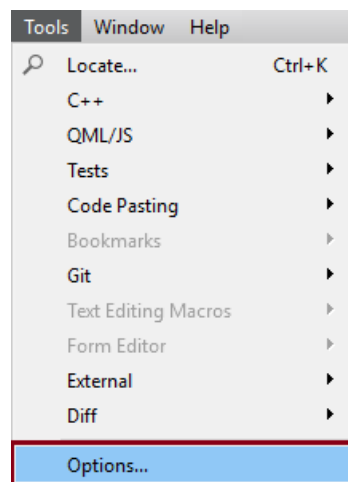
5.2 “Kits” / Configuring Qt Creator with multiple compilers

“Kits” (a combination of a compiler version and Qt version) can be used to switch between different compiler versions and architectures (32-bit, 64-bit) on the fly, if you have installed multiple Qt Creator versions. Kits can be shown and switched in the lower left corner in Qt Creator:

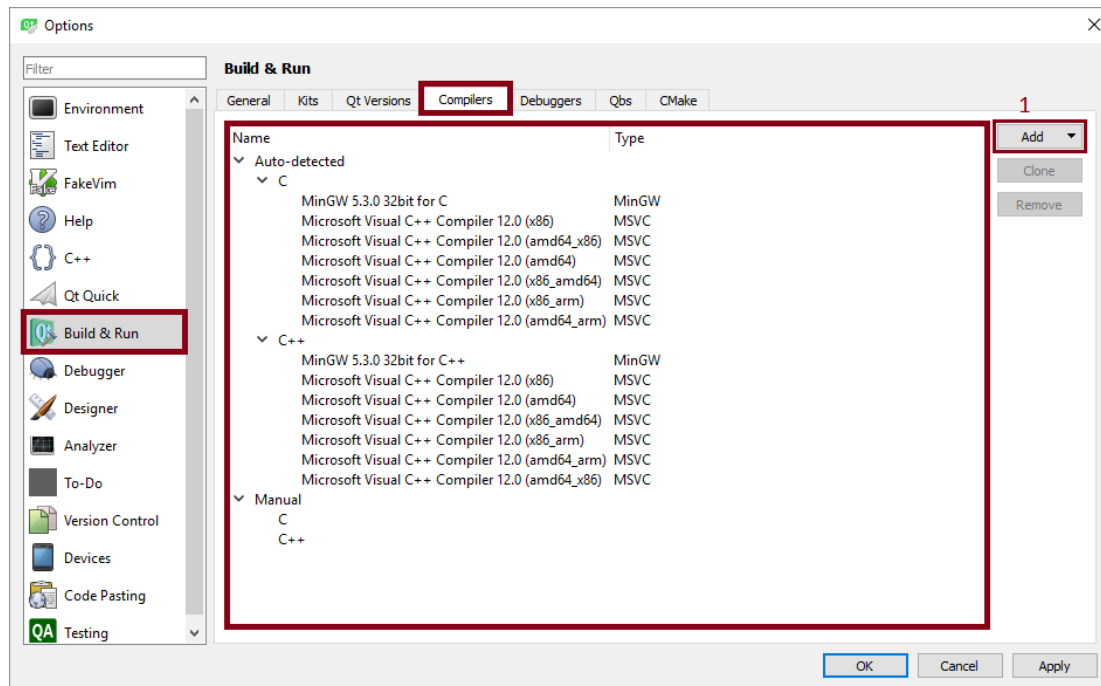


5.2.1 Adding another compiler / architecture manually

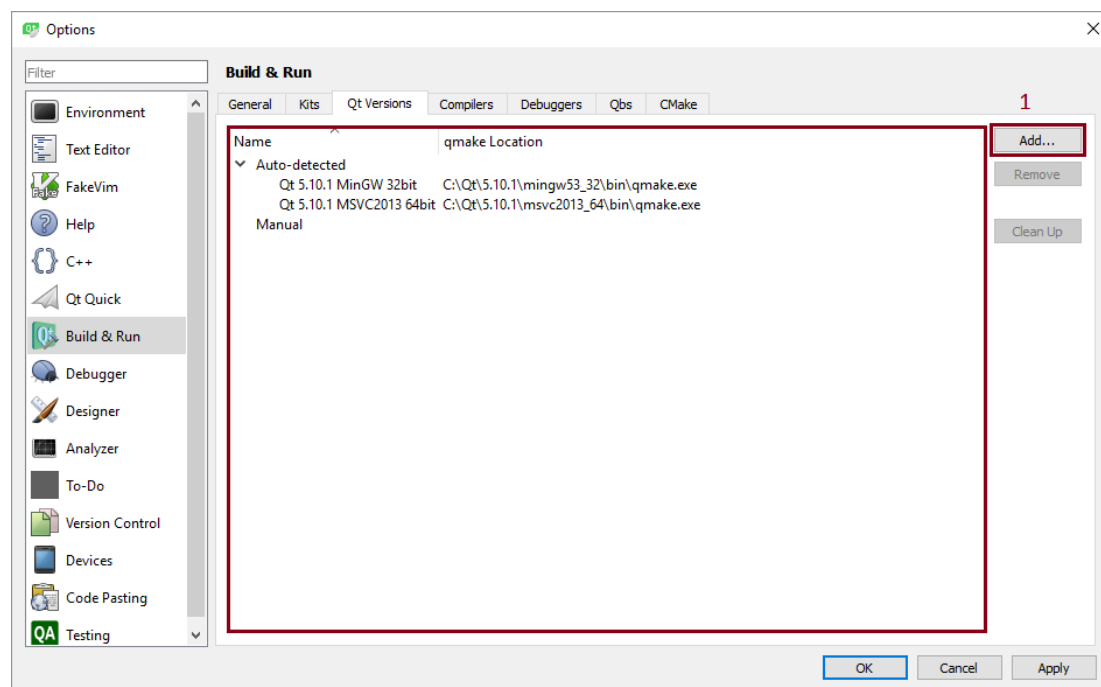
Normally, Qt Creator will auto-detect additional kits you install. If this does not work, you can add another compiler / architecture / Qt version manually. Start Qt Creator and open the options:

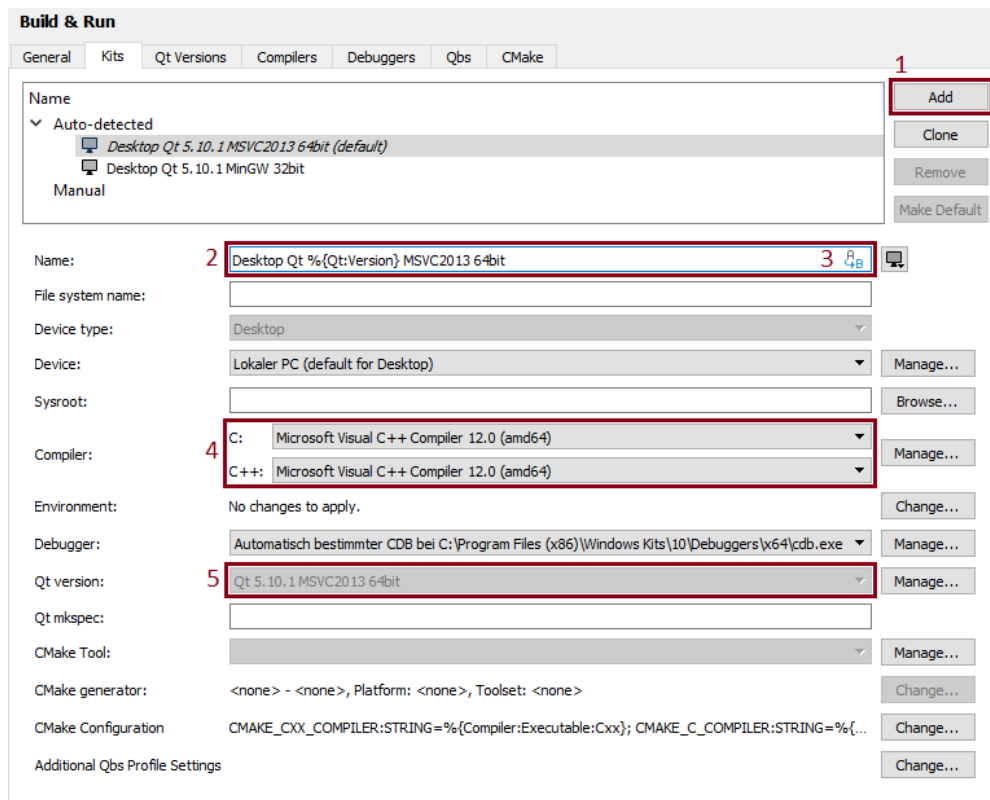


Before you add any kits, check if the compiler you would like to add is detected, otherwise add it with (1):



Also you need to add the corresponding Qt version manually, using (1), see the paths depicted for example (additionally, see section 4.2):



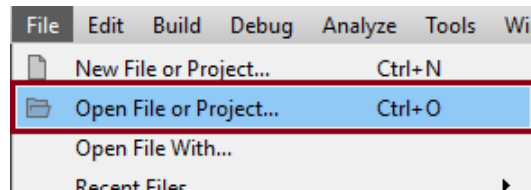


To add a kit:

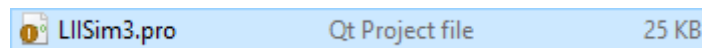
- Press (1)
- Enter a name for this kit in (2). You can use variables which are automatically replaced with (3).
- Choose your compiler at (4).
- Choose your Qt version at (5).

5.3 Open, configure and build LIISim in Qt Creator

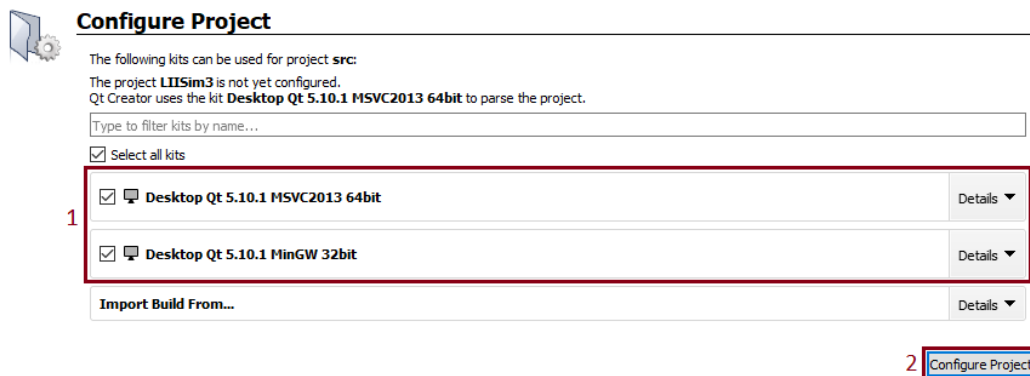
To open the LIISim3 project in Qt Creator, use the following menu option:



Select the LIISim3 project file (inside the project root directory):



In the following dialog, configure the kits (1) you would like to use and press “Configure Project” (2):



QtCreator should now open the “Edit” window, with the lower half showing the “General Messages” tab. This tab should display the following output:

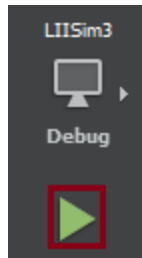
```

1 Project MESSAGE: OK: LIISim3.pro external library path
  (C:/Users/Dummy/Projects/LIISim3/externalLibraries)
2 Project MESSAGE: OK: LIISim3.pro MML library
3 Project MESSAGE: OK: LIISim3.pro boost library
4 Project MESSAGE: OK: qwt directory for Qt 5.10.1
5 Project MESSAGE: INFO: x86_64 build
6 Project MESSAGE: OK: qwt directory for current compiler:
  (C:/Users/Dummy/Projects/LIISim3/externalLibraries/qwt/msvc/x
  64)
7 Project MESSAGE: OK: LIISim3.pro external library path
  (C:/Users/Dummy/Projects/LIISim3/externalLibraries)
8 Project MESSAGE: OK: LIISim3.pro MML library
9 Project MESSAGE: OK: LIISim3.pro boost library
10 Project MESSAGE: OK: qwt directory for Qt 5.10.1
11 Project MESSAGE: INFO: x86_64 build
12 Project MESSAGE: OK: qwt directory for current compiler
   (C:/Users/Dummy/Projects/LIISim3/externalLibraries/qwt/msvc/x
   64)

```

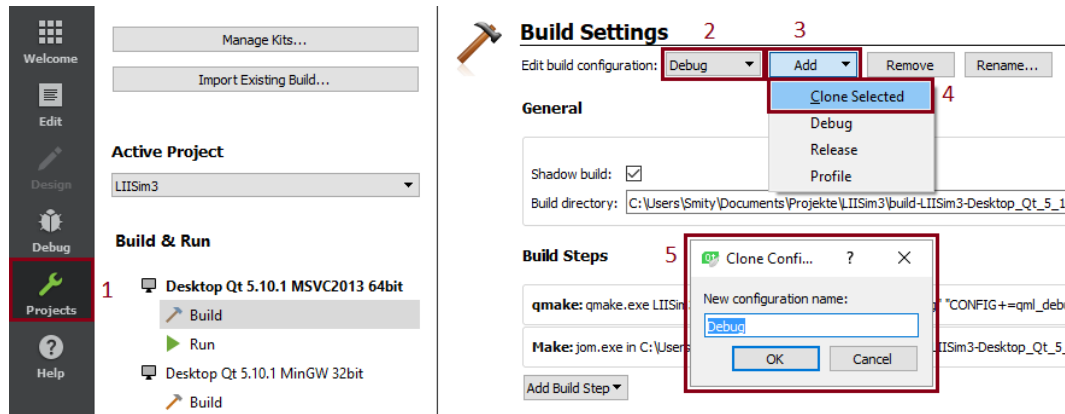
Check for the in this case green marked texts if there are any errors, especially wrong paths for the external libraries.

If you have configured the compiler right, you can now press the “play button” to build and run LIISim:



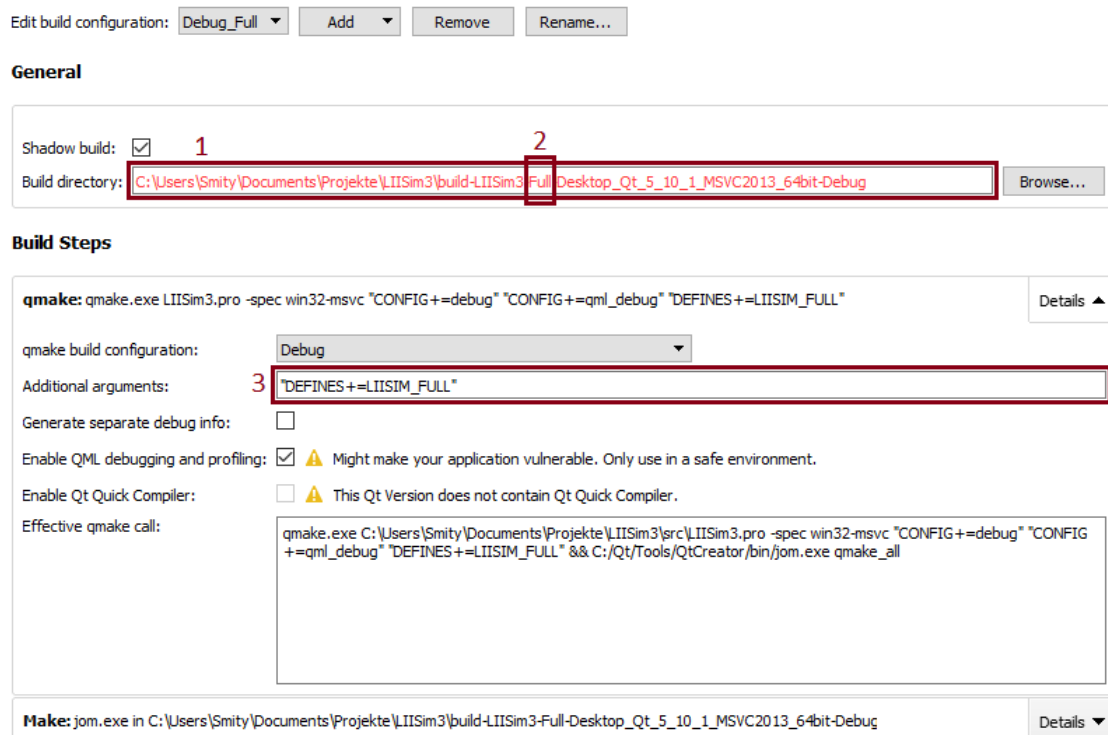
5.4 Switching between lite and full version

With “build configurations”, you can switch between the lite and full version of LIISim on the fly. To enable this, go into the settings of the loaded LIISim project (1):

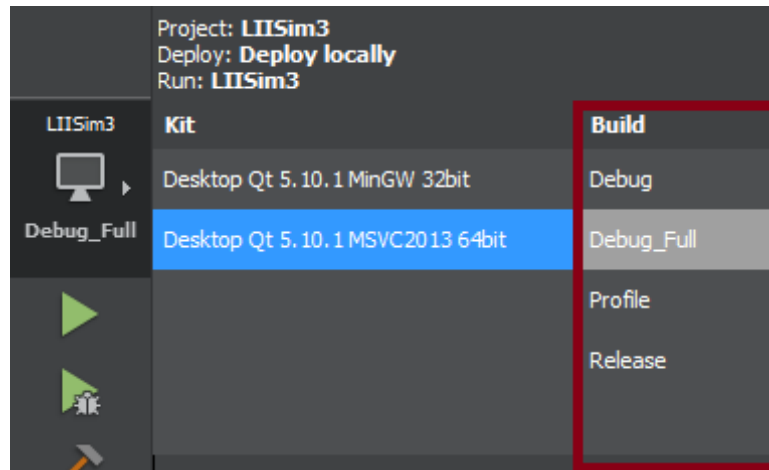


Choose the base build configuration with (2) (if you either want to work with the full version out of the Qt Creator [Debug] or like to build a full version release [Release]). Then use “Add” (3) and “Clone Selected” (4) to clone the chosen build configuration. Give the cloned configuration a new name in the appearing window (5) (for instance, add “_Full” behind the original name).

Build Settings



After cloning, the new configuration should automatically be shown. Edit the “Build directory” (1), for instance add a “-Full” in position (2), so the two builds are separated (if the build path is shown in red, ignore this, as it only shows that the path does not exist, but it is created automatically at first build). Then add the line shown in text field (3).



You now can switch between lite and full version with the kit and build selector.

6 Building a release

1) Before building the release binary, you need to **edit the current version number** and the **root directory** in the “Core.cpp” file:

```
const QString Core::LIISIM_VERSION = "3.0.5";

// -----
// PROGRAM ROOT DIRECTORY
// -----
//const QString Core::rootDir = "../src/"; // useful for development (build directory is in LIISim3 directory)
const QString Core::rootDir = ""; // use same folder as .exe
```

2) **Build** the project and **copy the binary (build directory/release/LIISim3.exe)** to the release preparation folder

3)

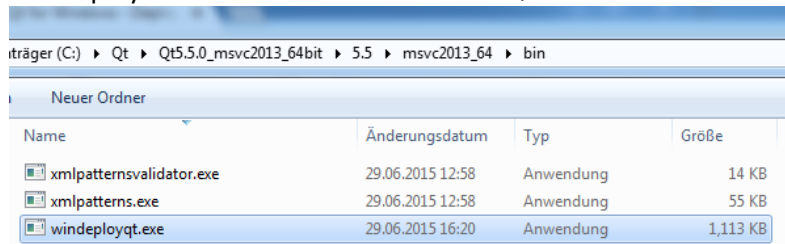
Run the Qt windows deploy tool on the LIISim3.exe.

“The Windows deployment tool is designed to automate the process of creating a deployable folder containing the Qt-related dependencies (libraries, QML imports, plugins, and translations) required to run the application from that folder.”

For further information, see:

<https://doc.qt.io/qt-5/windows-deployment.html>

The deployment tool can be found in the Qt-folder:



Name	Änderungsdatum	Typ	Größe
xmlpatternsvalidator.exe	29.06.2015 12:58	Anwendung	14 KB
xmlpatterns.exe	29.06.2015 12:58	Anwendung	55 KB
windeployqt.exe	29.06.2015 16:20	Anwendung	1,113 KB

Execute the following command in windows console:

```
/<QT-BIN-DIR>/windeployqt.exe /<RELEASE-DIR>/LIISim3.exe
```

<QT-BIN-DIR> replace this by the absolute path of the Qt binary directory

<RELEASE-DIR> replace this by the absolute path of the release preparation folder.

This should give you a similar output:

```

C:\Users\User>C:\Qt\Qt5.5.0\msvc2013_64bit\5.5\msvc2013_64\bin\windeployqt.exe D:\Projects\LIISim\ReleasePreparation\LIISim3_2017-12-28_win_x64\LIISim3.exe
D:\Projects\LIISim\ReleasePreparation\LIISim3_2017-12-28_win_x64\LIISim3.exe 64
bit, release executable
Adding Qt5Svg for qsvgicon.dll
Direct dependencies: Qt5Core Qt5Gui Qt5Widgets Qt5Xml
All dependencies : Qt5Core Qt5Gui Qt5Widgets Qt5Xml
To be deployed : Qt5Core Qt5Gui Qt5Svg Qt5Widgets Qt5Xml
Warning: Cannot find Visual Studio installation directory, VCINSTALLDIR is not set.
Updating Qt5Core.dll.
Updating Qt5Gui.dll.
Updating Qt5Svg.dll.
Updating Qt5Widgets.dll.
Updating Qt5Xml.dll.
Updating libGLU.dll.
Updating libEGL.dll.
Updating D3Dcompiler_47.dll.
Updating opengl32sw.dll.
Creating directory iconengines.
Updating qsvgicon.dll.
Creating directory imageformats.
Updating qdds.dll.
Updating qgif.dll.
Updating qicns.dll.
Updating qico.dll.
Updating qjp2.dll.
Updating qjpeg.dll.
Updating qmng.dll.
Updating qsvg.dll.
Updating qtga.dll.
Updating qtiff.dll.
Updating qwbmp.dll.
Updating qwebp.dll.
Creating directory platforms.
Updating qwindows.dll.
Creating D:\Projects\LIISim\ReleasePreparation\LIISim3_2017-12-28_win_x64\translations...
Creating qt_ca.qm...
Creating qt_cs.qm...
Creating qt_de.qm...
Creating qt_fi.qm...
Creating qt_fr.qm...
Creating qt_hu.qm...
Creating qt_it.qm...
Creating qt_ja.qm...
Creating qt_ko.qm...
Creating qt_lv.qm...
Creating qt_ru.qm...
Creating qt_sk.qm...
Creating qt_uk.qm...
C:\Users\User>

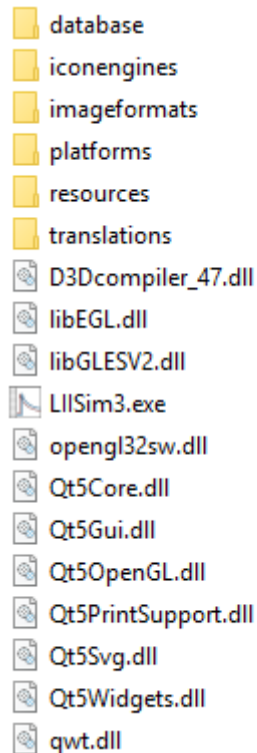
```

4) Additionally, the following files need to be added to the release preparation folder:

- **qwt.dll**
(from the compiled QWT binaries in the 'externalLibraries' folder, should be from the same compiler and architecture as the LIISim binary)
- **Qt5OpenGL.dll** and **Qt5PrintSupport.dll**
(from the Qt "bin"-directory, same architecture and Qt version as LIISim binary)
- **libmatio.dll** and **zlibwapi.dll**
(from the compiled MATIO binaries in the 'externalLibraries' folder, should be from the same compiler and architecture as the LIISim binary)

5) Then, copy the '**src/resources**' and '**src/data**' directories into the release directory.

The release directory should now look like this:

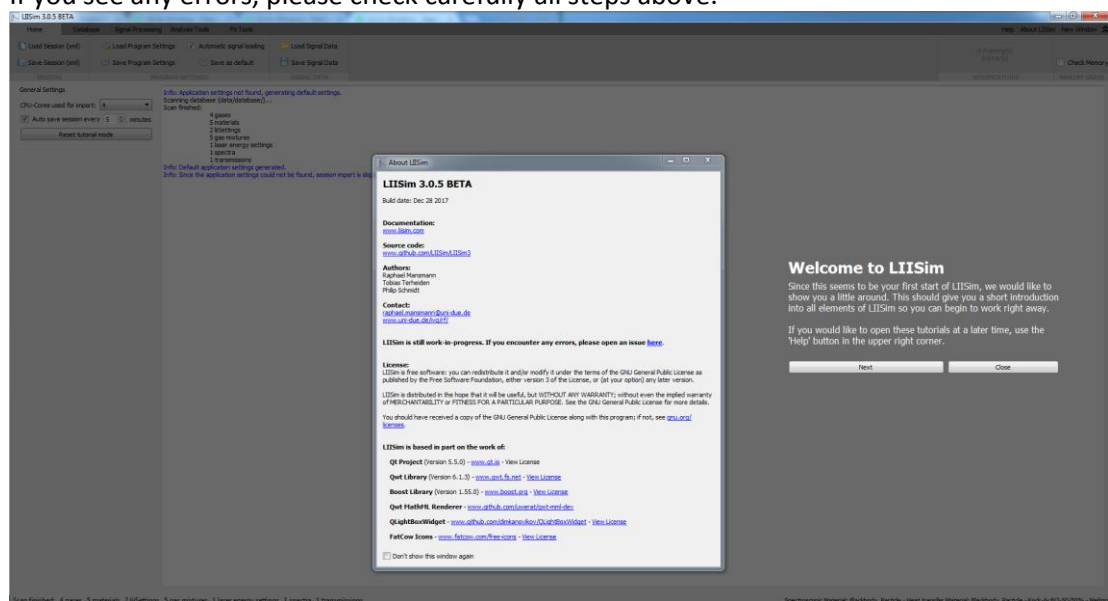


Note: To run LIISim3 on another PC, make sure you have the corresponding **Microsoft Visual C++ Redistributable** installed / included (same version / year as your Visual Studio version).

Additionally, for the **LIISim full version**, the **PicoScope drivers** and the **NI DAQmx drivers** need to be installed.

Now you should be able to execute the LIISim3.exe file without any errors:

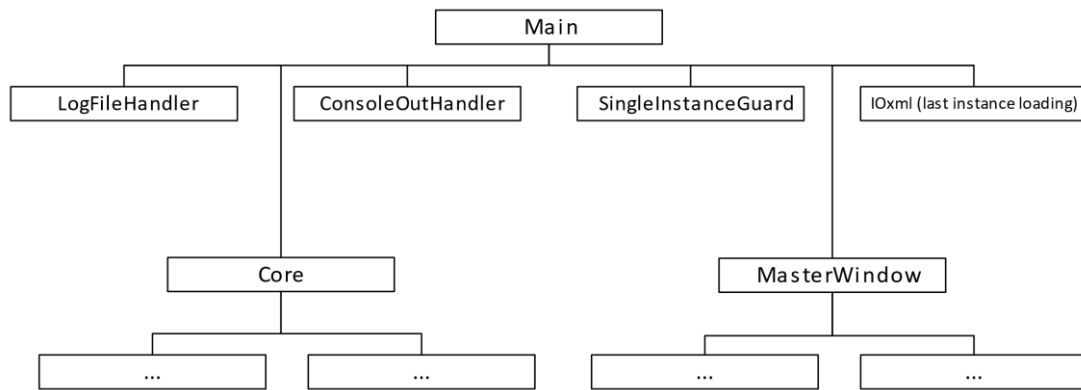
If you see any errors, please check carefully all steps above.



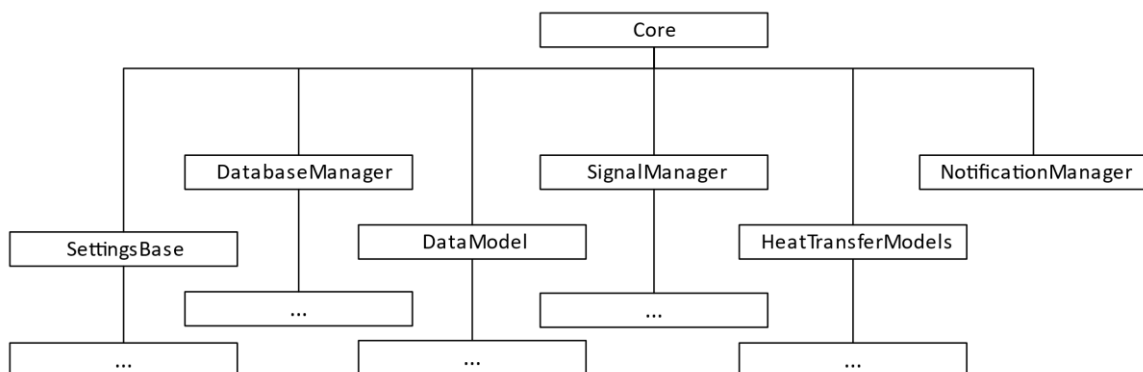
7 Class overview

The following diagrams provide a short overview over the class structure / relation of LIISim3, but without the claim to be complete.

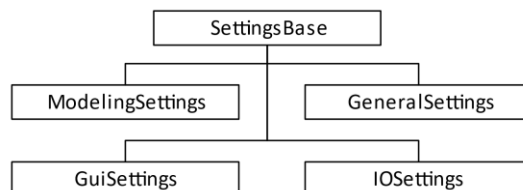
7.1 Main



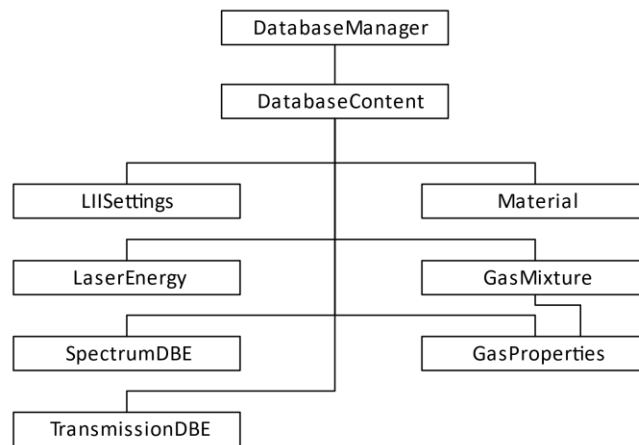
7.2 Core



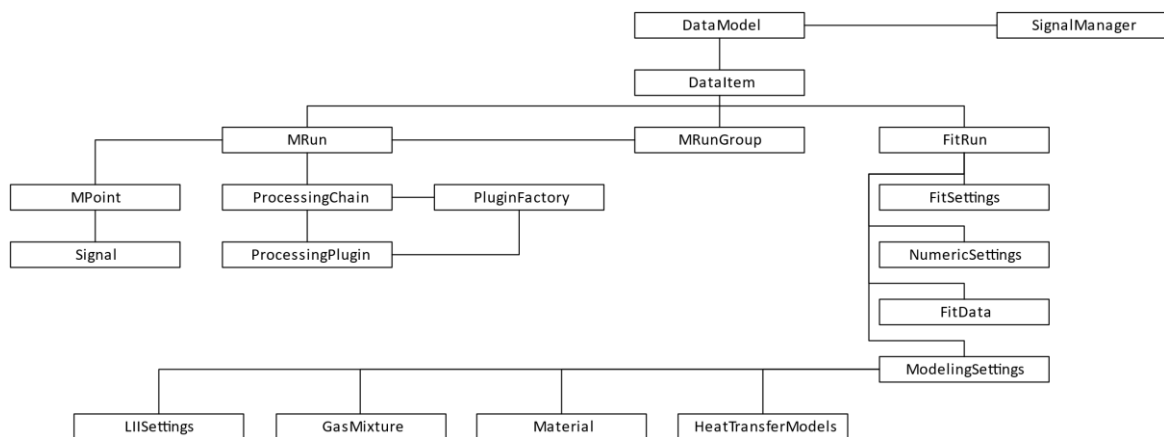
7.2.1 SettingsBase



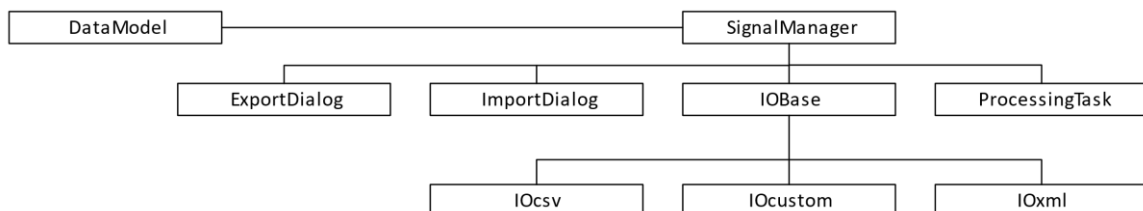
7.2.2 DatabaseManager



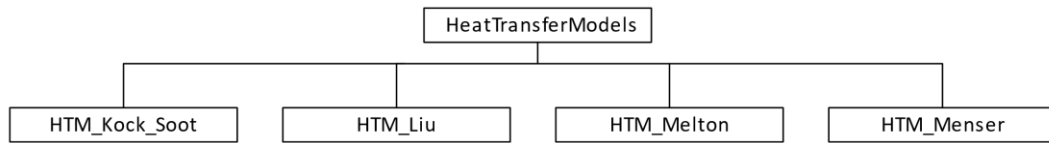
7.2.3 DataModel



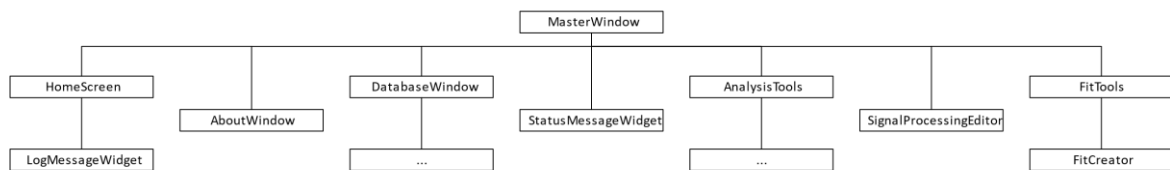
7.2.4 SignalManager



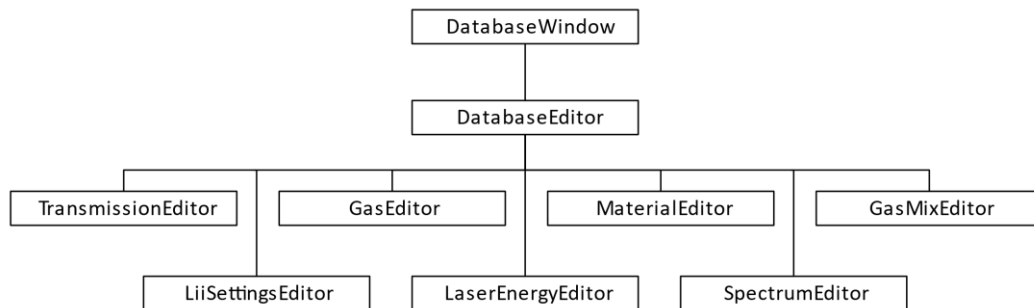
7.2.5 HeatTransferModels



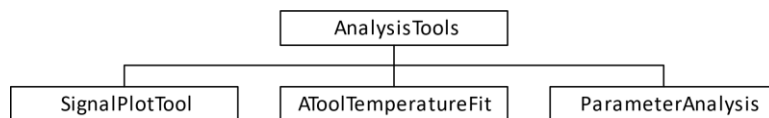
7.3 MasterWindow



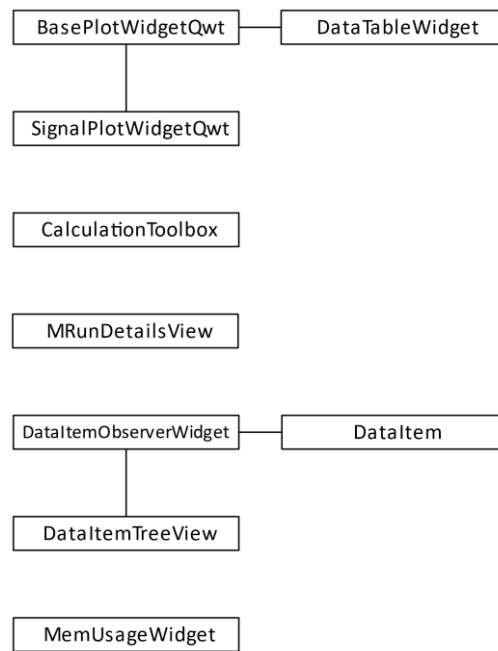
7.3.1 DatabaseWindow



7.3.2 AnalysisTools



7.3.3 UI Elements



8 Troubleshooting

- **“... PS6000.dll not found ...”**

If you get the error “... PS6000.dll not found ...” when executing a release, you need to install the PicoScope driver or SDK (the ‘6000’ in the name defines the PicoScope series, if you try to use another series).

- **“... nicaui.dll not found ...”**

If you get the error “... nicaui.dll not found ...” when executing a release, you need to install the NI DAQmx runtime or SDK.

- **Qt 5.10.1 / MSVC2013 build fails in qstringview.h**

Compilation of LIISim fails, because of incompatibility of Qt 5.10.1 with MSVC2013:

cl : Command line warning D9002 : ignoring unknown option '-Zc:inline'
--

Solution:

<https://forum.qt.io/topic/88001/helloworld-5-10-1-msvc2013-build-fails-in-qstringview-h>

-> *Install update:*

“Microsoft Visual Studio 2013 update 5”

<https://www.visualstudio.com/de-de/news/releasenotes/vs2013-update5-vs>

- If any strange errors occur, try a complete rebuild e.g. delete the build directory or use “Build” → “Rebuild all” in the QtCreator.

9 Frequently Asked Questions

[empty]