Squidstat API User Manual

Generated by Admiral Instruments LLC

July 20, 2023

Chapter 1

Squidstat API User Manual

The Admiral Instruments API gives more control of `our potentiostats`, and gives you the tools to integrate running our experiments in your pipeline and automating your workflow.

Our API lets you programmatically start an experiment, pause an experiment and stop an experiment. You can `Download` our API from our git repository.

For example, you may want to start an experiment with our device automatically whenever another device you have reads a certain temperature. Among other things, whenever starting, pausing or stopping an experiment happens, our API also sends a signal that you can use to control your workflow. For example, you may choose to start the next step in your pipeline whenever the experiment stops.

Let us start by going through the basics.

Chapter 2

Identifying USB Serial Ports

2.0.0.1 Introduction

In order for the API to communicate with a device, it is crucial to know its serial (i.e. COM) port, which enables the software to establish the correct communication pathway with the intended device. If the device is a Squidstat, you can easily locate the serial port in the Squidstat User Interface (SUI) software in the "More Options" tab, under "Device Information." If the SUI is not downloaded on your computer, you must determine the serial port using a different method. This guide outlines how to locate and identify the serial port of a device on Windows, Mac, and Linux platforms.

2.0.0.2 Windows

1. Connect the device to the computer via USB and power on the device.
2. Open Device Manager. You can open it directly using the search function (press Windows key) and type "Device Manager" to launch. You can also access it through Control Panel:
   - Go to Control Panel
   - Select 'Hardware and Sound'
   - Under 'Devices and Printers' click on 'Device Manager'
3. Expand the dropdown menu labeled 'Ports' to view the list of connected devices. Look for entries referring to a USB (e.g. `USB Serial Port(COM3)`). If there are multiple COM ports, power cycle or disconnect the device to determine which is correct.
4. When referring to the serial port from the example above in the API, the format for the entry would be `COM3`.

2.0.0.3 Mac

1. Connect the device to the computer via USB and power on the device.
2. Open Terminal. You can open it by going to 'Applications' -> 'Utilities' -> 'Terminal.' Alternatively, you can use Spotlight Search (press Cmd + Space) and type "Terminal" to launch.
3. List the serial devices. In the Terminal window, enter the following command and press Enter:

   ```
   ls /dev/cu.*
   ```
4. Identify the USB serial port. Look for the entry in the list that corresponds to your USB device.

   Example output:

```
/dev/cu.Bluetooth-Incoming-Port
/dev/cu.usbmodem14201
/dev/cu.usbserial-Admiral_1409
```

In this example, `/dev/cu.usbmodem14201` and `/dev/cu.usbserial-Admiral_1409` are the USB serial ports associated with the connected devices. If there are multiple entries, power cycle or disconnect the device and enter the same command to determine which is correct.

1. When referring to a serial port from the example above in the API, the format for the entry would be `cu.↩ usbmodem14201`.

2.0.0.4 Linux

1. Connect the device to the computer via USB and power on the device.
2. Open a Terminal window by pressing Ctrl + Alt + T.
3. Execute the `ls /dev` command. In the Terminal, enter the following command and press Enter:
   `ls /dev | grep tty`
4. Identify the USB serial port. Look for the lines that refer to USB devices (e.g., "`ttyACM0`" or "`ttyUSB1`"). If there are multiple entries, power cycle or disconnect the device and enter the same command to determine which is correct.
5. When referring to the serial port from the example above in the API, the format for the entry would be `tty↩ ACM0`.

Chapter 3

Building API using CMake

3.0.0.1 Introduction

This section provides guidance to developers on building the SquidstatLibrary using the command line. By following the instructions outlined here, developers can effectively compile and construct the SquidstatLibrary, enabling them to incorporate its functionality into their projects. The step-by-step process explained below will help developers easily set up the SquidstatLibrary and make it ready for integration, ensuring a seamless experience for utilizing its capabilities through the command line interface.

3.0.0.2 Clone API from Git

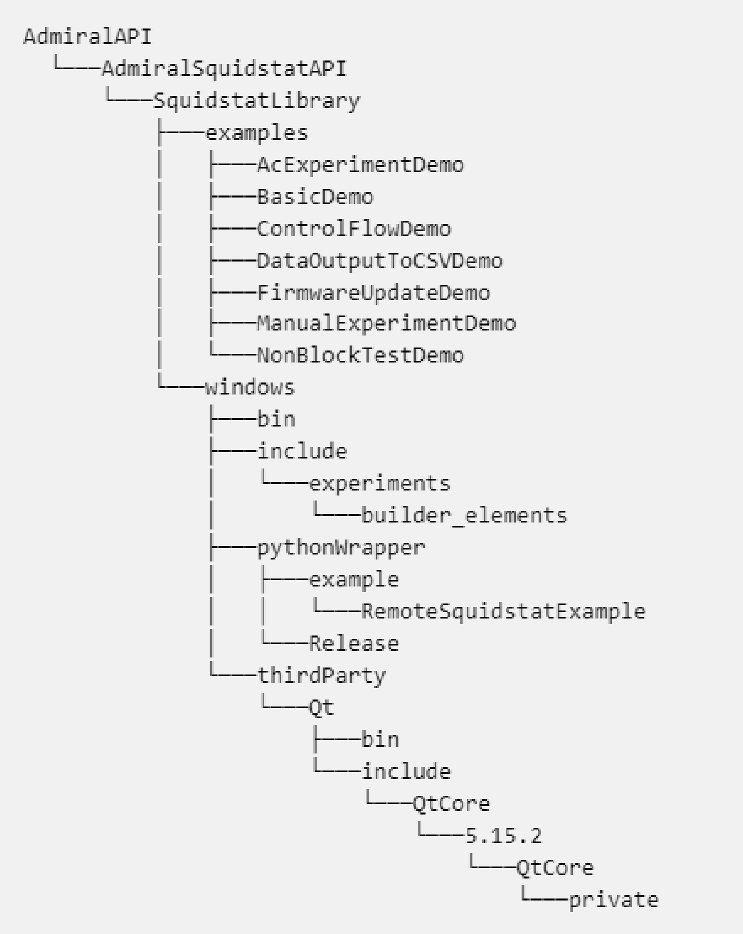
1. To clone the repository, you will need the Git tool. Depending on your platform, you can download the Git tool from `this link`.
2. To verify if Git is properly installed, you can follow these steps:
   - Go to your desktop.
   - Open the command prompt.
   - Type git -v and press Enter.
   - If Git is installed correctly, you should see the version information displayed in the terminal.
     ```
     git version 2.41.0.windows.1
     ```
3. To create a new directory with a specific name, such as `AdmiralAPI`, it is recommended to choose a name without spaces.
4. Click on the newly created directory, `AdmiralAPI`, and open the command prompt.
5. To clone the API from the `git repository`, type the following command in the command prompt and press Enter.

   git clone `https://github.com/Admiral-Instruments/AdmiralSquidstatAPI`

The result in the command prompt will look like this:

```
Cloning into 'AdmiralSquidstatAPI'...
remote:  Enumerating objects:  1846, done.
remote:  Counting objects:  100% (508/508), done.
remote:  Compressing objects:  100% (159/159), done.
remote:  Total 1846 (delta 440), reused 360 (delta 349), pack-reused 1338
Receiving objects:  100% (1846/1846), 79.18 MiB | 10.10 MiB/s, done.
Resolving deltas:  100% (1044/1044), done.
```

If you check in your directory, you will find a new directory named "AdmiralSquidstatAPI" which contains the Admiral Instruments API.



```
AdmiralAPI
  └───AdmiralSquidstatAPI
      └───SquidstatLibrary
          ├───examples
          │   ├───AcExperimentDemo
          │   ├───BasicDemo
          │   ├───ControlFlowDemo
          │   ├───DataOutputToCSVDemo
          │   ├───FirmwareUpdateDemo
          │   ├───ManualExperimentDemo
          │   └───NonBlockTestDemo
          └───windows
              ├───bin
              ├───include
              │   └───experiments
              │       └───builder_elements
              ├───pythonWrapper
              │   ├───example
              │   │   └───RemoteSquidstatExample
              │   └───Release
              └───thirdParty
                  └───Qt
                      ├───bin
                      └───include
                          └───QtCore
                              └───5.15.2
                                  └───QtCore
                                      └───private
```

Figure 3.1 API Bundler Directory Structure

3.0.0.3 Cmake Installtion

1. To utilize the CMakeLists.txt file, you need to install CMake. You can download CMake from here.
2. Provide the Cmake path in your environment variable.
3. To verify if Cmake is installed correctly, type `cmake` in the command prompt and press Enter.
4. Once the installation is complete, start the build.

3.0.0.4 Build project

1. Go to the directory using cd `AdmiralAPI`.
2. Open command prompt. Type the command below. This command will generate the build. It will take compiler which is available on your computer. Make sure on Windows you have the MSVC 64 compiler, and on Mac You have the Clang compiler.
   ```
   cmake -B build -S "AdmiralSquidstatAPI/SquidstatLibrary/"
   ```
   Note: If you want use a different build generator, type the name of that generator followed by `-G`. You can check the build generator option with the command `cmake -G`.
3. Build the project using the command below, which will compile all examples present the in `Squidstat↩ Library` directory.
   ```
   cmake --build ./build
   ```

Chapter 4

Running the API with Qt

4.0.0.1 Introduction

This section is dedicated to guiding developers on building the SquidstatLibrary using Qt Creator. By following the instructions provided here, developers can seamlessly compile and construct the SquidstatLibrary within the Qt Creator IDE. The step-by-step process outlined below will assist developers in setting up the SquidstatLibrary in Qt Creator, allowing them to leverage its functionalities effectively. With the intuitive interface and powerful features of Qt Creator, developers can easily integrate the SquidstatLibrary into their projects, enhancing their ability to analyze and manipulate Squidstat experiment data.

4.0.0.2 Clone API from Git

1. To clone the repository, you will need the Git tool. Depending on your platform, you can download the Git tool from `this link`.
2. To verify if Git is properly installed, you can follow these steps:
   - Go to your desktop.
   - Open the command prompt.
   - Type git -v and press Enter.
   - If Git is installed correctly, you should see the version information displayed in the terminal.

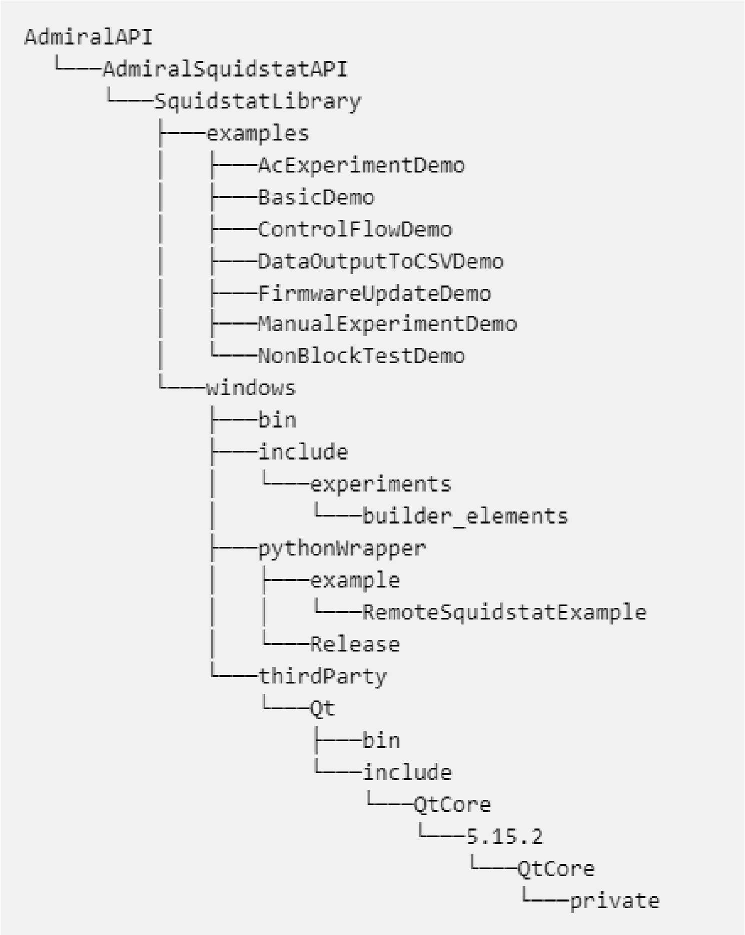
     ```
     git version 2.41.0.windows.1
     ```
3. To create a new directory with a specific name, such as `AdmiralAPI`, it is recommended to choose a name without spaces.
4. Click on the newly created directory, `AdmiralAPI`, and open the command prompt.
5. To clone the API from the `git repository`, type the following command in the command prompt and press Enter.

   git clone `https://github.com/Admiral-Instruments/AdmiralSquidstatAPI`

The result in the command prompt will look like this:

```
Cloning into 'AdmiralSquidstatAPI'...
remote: Enumerating objects: 1846, done.
remote: Counting objects: 100% (508/508), done.
remote: Compressing objects: 100% (159/159), done.
remote: Total 1846 (delta 440), reused 360 (delta 349), pack-reused 1338
Receiving objects: 100% (1846/1846), 79.18 MiB | 10.10 MiB/s, done.
Resolving deltas: 100% (1044/1044), done.
```

If you check in your directory, you will find a new directory named "AdmiralSquidstatAPI" which contains the Admiral Instruments API.



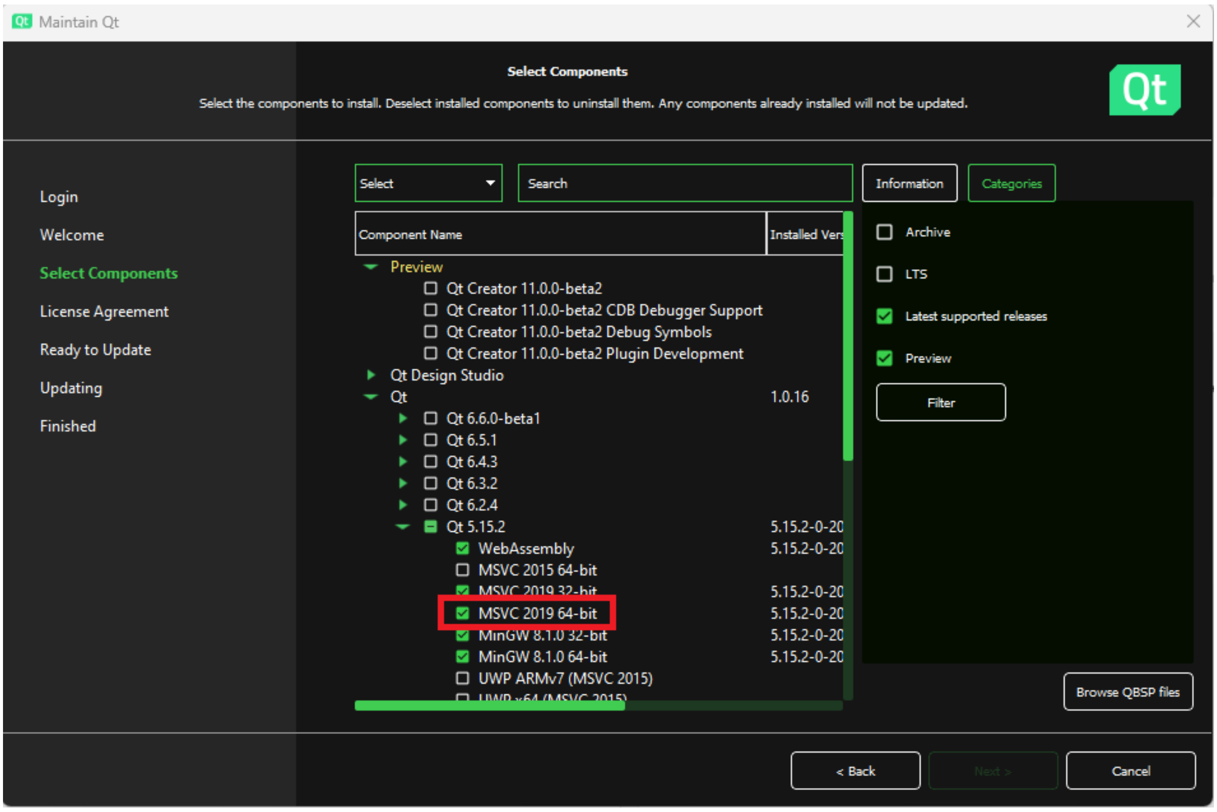
```
AdmiralAPI
  └───AdmiralSquidstatAPI
      └───SquidstatLibrary
          ├───examples
          │   ├───AcExperimentDemo
          │   ├───BasicDemo
          │   ├───ControlFlowDemo
          │   ├───DataOutputToCSVDemo
          │   ├───FirmwareUpdateDemo
          │   ├───ManualExperimentDemo
          │   └───NonBlockTestDemo
          └───windows
              ├───bin
              ├───include
              │   └───experiments
              │       └───builder_elements
              ├───pythonWrapper
              │   ├───example
              │   │   └───RemoteSquidstatExample
              │   └───Release
              └───thirdParty
                  └───Qt
                      ├───bin
                      └───include
                          └───QtCore
                              └───5.15.2
                                  └───QtCore
                                      └───private
```

Figure 4.1 API Bundler Directory Structure

4.0.0.3 Qt Installation.

1. Download Qt by clicking here. To compile the API with Qt, it is required to also install the MSVC 64-bit compiler kit on Windows, Dekstop GCC 64bit on Linux, and Clang 64 on Mac during the Qt installation process.

    - Enter your Qt login account information. If you don't have an account, you can sign up and create a new one to proceed with the installation.
    - During the installation process, please ensure that you add at least one of the following components: MSVC2019 64-bit or any MSVC∗∗∗∗ 64-bit kit on Windows, Dekstop GCC 64bit on Linux, and Clang 64 on Mac.



Select Components

Select the components to install. Deselect installed components to uninstall them. Any components already installed will not be updated.

Component Name	Installed Vers
Preview	
□ Qt Creator 11.0.0-beta2	
□ Qt Creator 11.0.0-beta2 CDB Debugger Support	
□ Qt Creator 11.0.0-beta2 Debug Symbols	
□ Qt Creator 11.0.0-beta2 Plugin Development	
Qt Design Studio	
Qt	1.0.16
□ Qt 6.6.0-beta1	
□ Qt 6.5.1	
□ Qt 6.4.3	
□ Qt 6.3.2	
□ Qt 6.2.4	
■ Qt 5.15.2	5.15.2-0-20
☑ WebAssembly	5.15.2-0-20
□ MSVC 2015 64-bit	
☑ MSVC 2019 32-bit	5.15.2-0-20
☑ MSVC 2019 64-bit	5.15.2-0-20
☑ MinGW 8.1.0 32-bit	5.15.2-0-20
☑ MinGW 8.1.0 64-bit	5.15.2-0-20
□ UWP ARMv7 (MSVC 2015)	
□ UWP x64 (MSVC 2015)	

□ Archive

□ LTS

☑ Latest supported releases

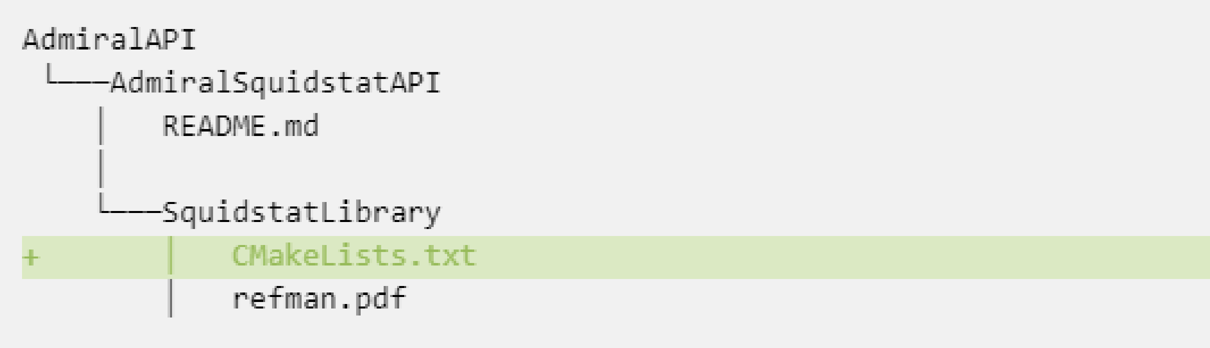
☑ Preview

Figure 4.2 MSVC kit selection

2. If you have already installed Qt on your computer but do not have the appropritate kit, you can navigate to the Qt installation directory and open `MaintenanceTool` tool. From there, you can install the appropriate kit by selecting the "Add or remove components" option. However, new Qt users can skip this step.
3. To utilize the CMakeLists.txt file, you need to install CMake and a build generator such as ninja; otherwise, you will have to manually specify the header file includes and library paths. You can download CMake from `here` or select CMake, build generator (ninja) in the "Developer and Designer tools" section of the Qt installation process.
4. Once the installation is complete, you can open the API project.

4.0.0.4 Open Project with Qt and Cmake

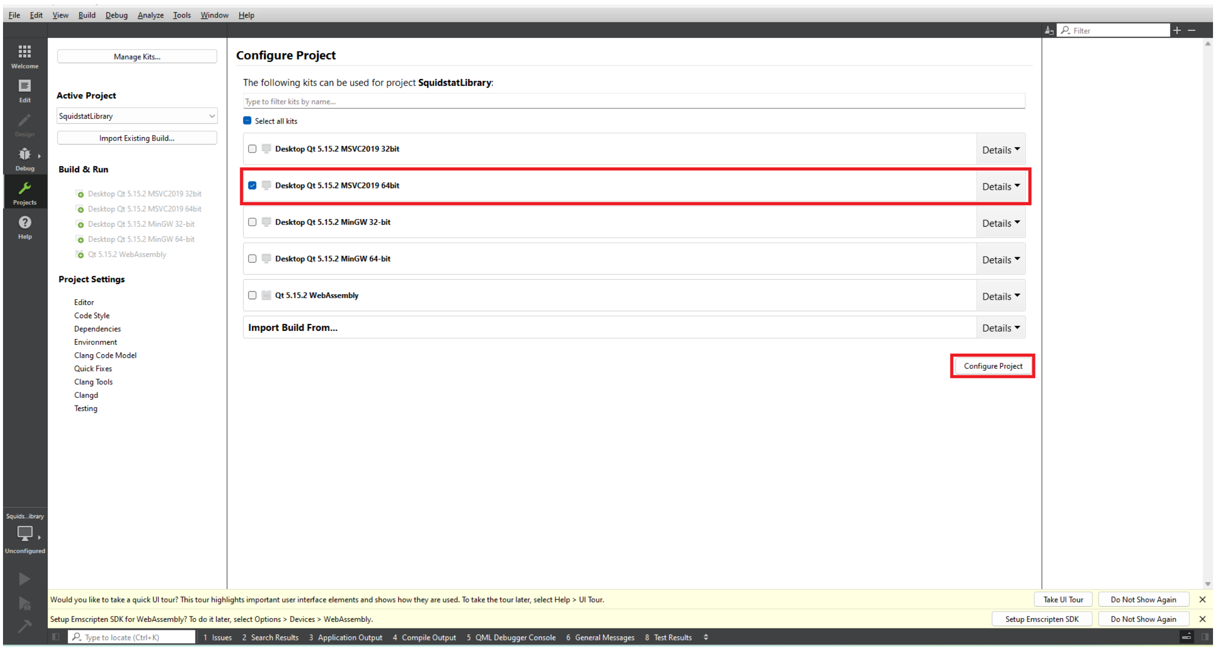
1. Open Qt Creator and select the `File` tab. Within the File tab, choose the `Open File or Project` option.
2. Select the CMakeLists.txt file located inside the `AdmiralSquidstatAPI > SquidstatLibrary` directory.



```
AdmiralAPI
 └──AdmiralSquidstatAPI
     │   README.md
     │
     └──SquidstatLibrary
+          │   CMakeLists.txt
           │   refman.pdf
```

Figure 4.3 Top Level Cmakelist file

3. Once you open the Qt CMakeLists.txt in Qt, it will provide you with the option to select the kit. Choose the MSVC 64-bit kit or appropriate kit w.r.t platform ,and click on "Configure Project."



Active Project

SquidstatLibrary

Build & Run

- Desktop Qt 5.15.2 MSVC2019 32bit
- Desktop Qt 5.15.2 MSVC2019 64bit
- Desktop Qt 5.15.2 MinGW 32-bit
- Desktop Qt 5.15.2 MinGW 64-bit
- Qt 5.15.2 WebAssembly

Project Settings

- Editor
- Code Style
- Dependencies
- Environment
- Clang Code Model
- Quick Fixes
- Clang Tools
- Clangd
- Testing

Configure Project

The following kits can be used for project **SquidstatLibrary**:

Type to filter kits by name...

☑ Select all kits

☐ Desktop Qt 5.15.2 MSVC2019 32bit

☑ Desktop Qt 5.15.2 MSVC2019 64bit

☐ Desktop Qt 5.15.2 MinGW 32-bit

☐ Desktop Qt 5.15.2 MinGW 64-bit

☐ Qt 5.15.2 WebAssembly

Import Build From...

Would you like to take a quick UI tour? This tour highlights important user interface elements and shows how they are used. To take the tour later, select Help > UI Tour.

Setup Emscripten SDK for WebAssembly? To do it later, select Options > Devices > WebAssembly.

Figure 4.4 MSVC kit check mark

4. You can check the General Message section located in the footer. CMake will automatically configure the project, including the required libraries and header files.

5. The project solution will look like the image below.

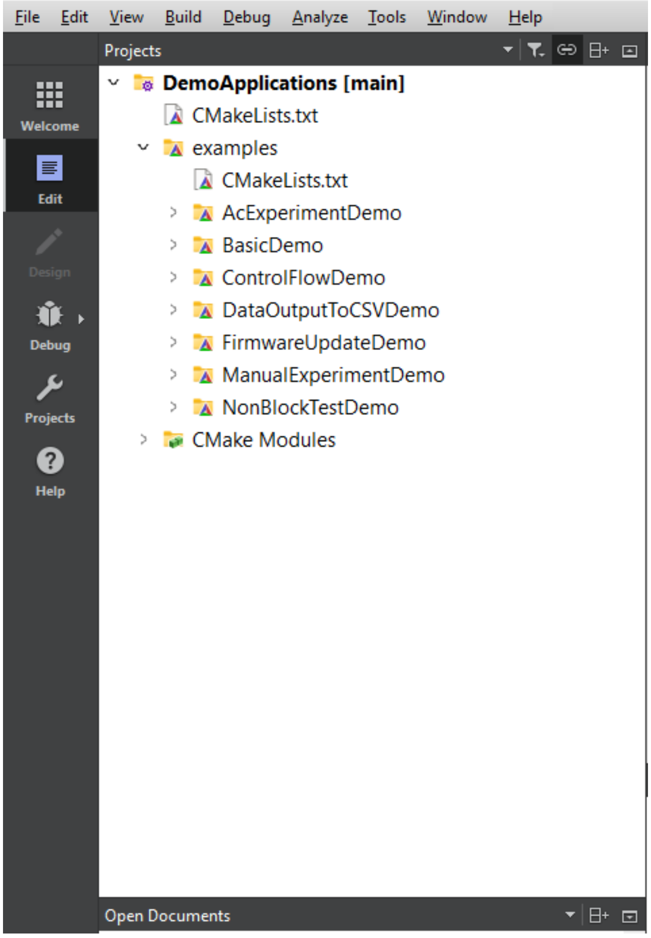
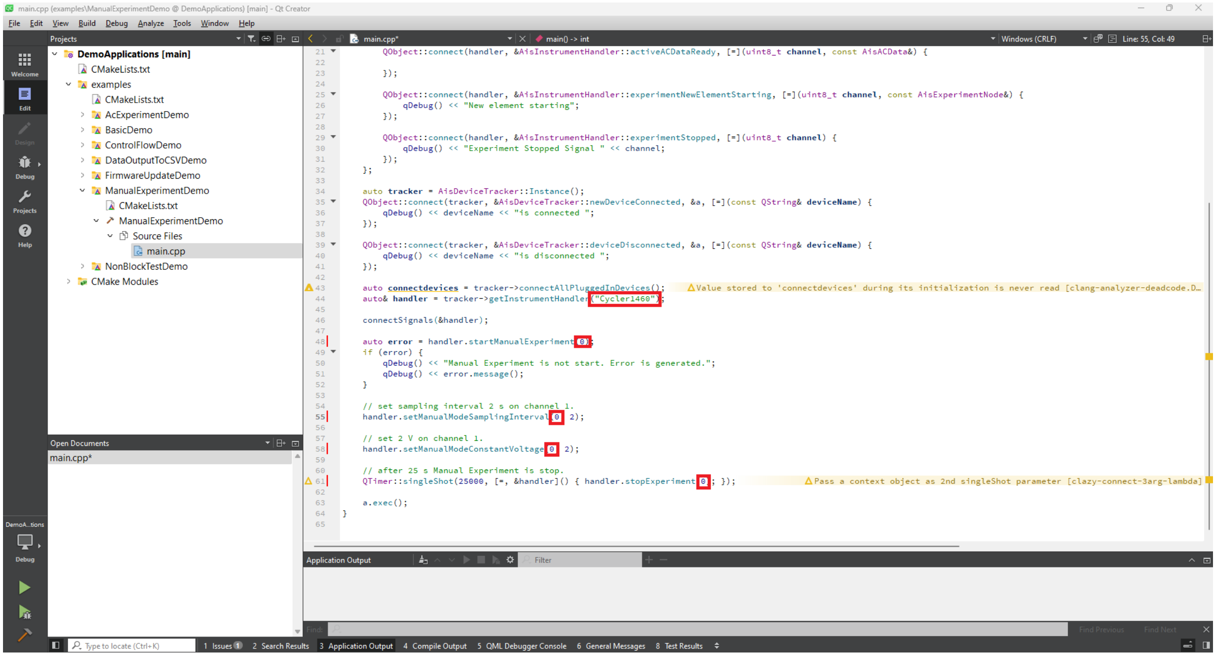


Figure 4.5 Qt project solution image

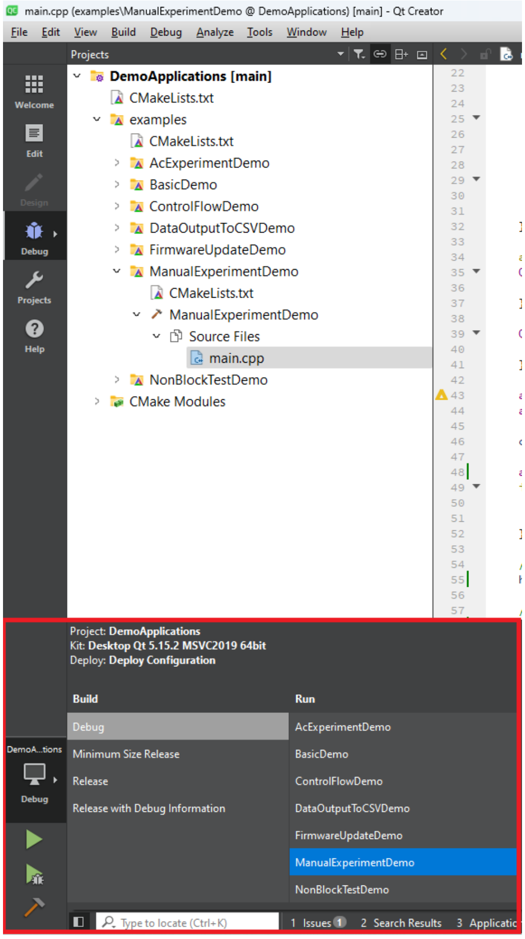
6. You can select any example from the list. For the purpose of this tutorial, select the "ManualexperimentDemo" project and open the main.cpp file. You are required to change the deviceName and channel number.



```
        QObject::connect(handler, &AisInstrumentHandler::activeACDataReady, [=](uint8_t channel, const AisACData&) {

        });

        QObject::connect(handler, &AisInstrumentHandler::experimentNewElementStarting, [=](uint8_t channel, const AisExperimentNode&) {
            qDebug() << "New element starting";
        });

        QObject::connect(handler, &AisInstrumentHandler::experimentStopped, [=](uint8_t channel) {
            qDebug() << "Experiment Stopped Signal " << channel;
        });
    };

    auto tracker = AisDeviceTracker::Instance();
    QObject::connect(tracker, &AisDeviceTracker::newDeviceConnected, &a, [=](const QString& deviceName) {
        qDebug() << deviceName << "is connected ";
    });

    QObject::connect(tracker, &AisDeviceTracker::deviceDisconnected, &a, [=](const QString& deviceName) {
        qDebug() << deviceName << "is disconnected ";
    });

    auto connectdevices = tracker->connectAllPluggedInDevices();    ⚠Value stored to 'connectdevices' during its initialization is never read [clang-analyzer-deadcode.D…
    auto& handler = tracker->getInstrumentHandler("Cycler1460");

    connectSignals(&handler);

    auto error = handler.startManualExperiment(0);
    if (error) {
        qDebug() << "Manual Experiment is not start. Error is generated.";
        qDebug() << error.message();
    }

    // set sampling interval 2 s on channel 1.
    handler.setManualModeSamplingInterval(0, 2);

    // set 2 V on channel 1.
    handler.setManualModeConstantVoltage(0, 2);

    // after 25 s Manual Experiment is stop.
    QTimer::singleShot(25000, [=, &handler]() { handler.stopExperiment(0); });    ⚠Pass a context object as 2nd singleShot parameter [clazy-connect-3arg-lambda]

    a.exec();
}
```

Figure 4.6 Qt manual experiment code

7. Select either Debug or Release mode according to your requirements and click on the run button to execute the manualExperimentDemo.



Project: DemoApplications
Kit: Desktop Qt 5.15.2 MSVC2019 64bit
Deploy: Deploy Configuration

Figure 4.7 Qt selection of Debug and Release

8. You can view the output data from the manual experiment in the Application Output window.

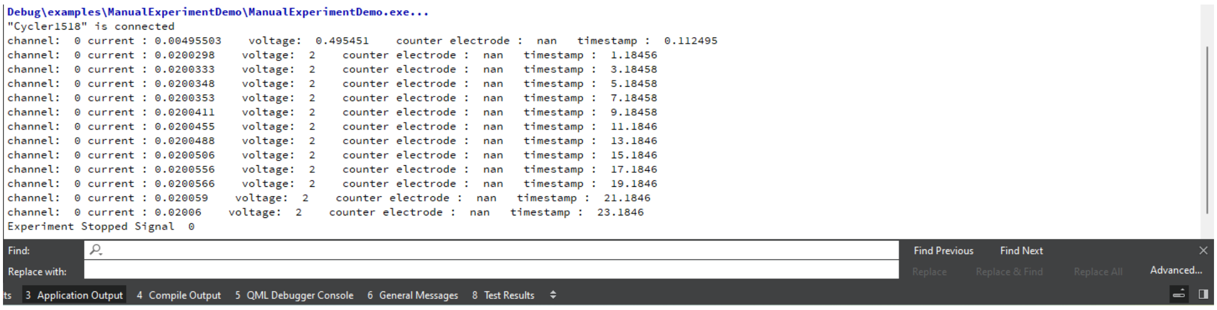

```
Debug\examples\ManualExperimentDemo\ManualExperimentDemo.exe...
"Cycler1518" is connected
channel:  0 current : 0.00495503    voltage:  0.495451    counter electrode :  nan   timestamp :  0.112495
channel:  0 current : 0.0200298    voltage:  2    counter electrode :  nan   timestamp :  1.18456
channel:  0 current : 0.0200333    voltage:  2    counter electrode :  nan   timestamp :  3.18458
channel:  0 current : 0.0200348    voltage:  2    counter electrode :  nan   timestamp :  5.18458
channel:  0 current : 0.0200353    voltage:  2    counter electrode :  nan   timestamp :  7.18458
channel:  0 current : 0.0200411    voltage:  2    counter electrode :  nan   timestamp :  9.18458
channel:  0 current : 0.0200455    voltage:  2    counter electrode :  nan   timestamp :  11.1846
channel:  0 current : 0.0200488    voltage:  2    counter electrode :  nan   timestamp :  13.1846
channel:  0 current : 0.0200506    voltage:  2    counter electrode :  nan   timestamp :  15.1846
channel:  0 current : 0.0200556    voltage:  2    counter electrode :  nan   timestamp :  17.1846
channel:  0 current : 0.0200566    voltage:  2    counter electrode :  nan   timestamp :  19.1846
channel:  0 current : 0.020059    voltage:  2    counter electrode :  nan   timestamp :  21.1846
channel:  0 current : 0.02006    voltage:  2    counter electrode :  nan   timestamp :  23.1846
Experiment Stopped Signal  0
```

Figure 4.8 Qt output window

Chapter 5

Updating Firmware

5.0.0.1 Introduction

The following example will help illustrate the use of the Squidstatlibrary for updating the firmware of Device

We will go through an example.

5.0.0.2 Building a Custom Experiment with Python

5.0.0.2.1 Import all the required basic class from SquidstatLibrary, and Qt Library.

```
#include
      "AisDeviceTracker.h"
#include <QCoreApplication>
#include <qdebug.h>
#include <qfileinfo.h>
```

5.0.0.2.2 Connect the Notification signal.

```
QObject::connect(tracker,
      &AisDeviceTracker::firmwareUpdateNotification, &a, [=](const QString& message) {
          qInfo() « message;
});
```

5.0.0.2.3 request to update the firmware to all connected device.

```
auto nmberOfDevice =
      tracker->updateFirmwareOnAllAvailableDevices();
if (nmberOfDevice == 0) {
     qInfo() « "Firmware update is not start in any of device";
} else {
     qInfo() « "Firmware update start in " « nmberOfDevice « "device.";
}
```

5.0.0.2.4 request to update the firmware to specific device using comport.

```
QRegExp
      rx("^[Cc][Oo][Mm][0-9]+$");
if (rx.exactMatch(comPort) == false) {
   qInfo() « " Arguments is not valid.  Example:  " «
      QFileInfo(QCoreApplication::applicationFilePath()).fileName() « " COM3";
} else {
   auto error = tracker->updateFirmwareOnComPort(comPort);
   if (error) {
      qInfo() « error.message();
   }
}
```

5.0.0.2.5 Full Example Here is everything put together, and complete working examples.

```
#include "AisDeviceTracker.h"
#include <QCoreApplication>
#include <qdebug.h>
#include <qfileinfo.h>
int main(int argc, char* argv[])
{
    QCoreApplication a(argc, argv);
    auto tracker = AisDeviceTracker::Instance();
    QObject::connect(tracker, &AisDeviceTracker::firmwareUpdateNotification, &a, [=](const QString& message)
      {
            qInfo() « message;
        });
    if (argc == 1) {
        auto nmberOfDevice = tracker->updateFirmwareOnAllAvailableDevices();
        if (nmberOfDevice == 0) {
            qInfo() « "Firmware update is not start in any of device";
        } else {
            qInfo() « "Firmware update start in " « nmberOfDevice « "device.";
        }

    } else {
        if (argc == 2) {
            auto comPort = argv[1];

            QRegExp rx("^[Cc][Oo][Mm][0-9]+$");
            if (rx.exactMatch(comPort) == false) {
                qInfo() « " Arguments is not valid.  Example:  " «
      QFileInfo(QCoreApplication::applicationFilePath()).fileName() « " COM3";

            } else {
                auto error = tracker->updateFirmwareOnComPort(comPort);
                if (error) {
                    qInfo() « error.message();

                }
            }
        }
    }
    a.exec();
}
```

Chapter 6

The Basics of Running Experiments

The basic building block of a custom experiment are the elements. An element is an elementary experiment such as Constant Voltage/Potential (CV) or Constant Current (CC). A custom experiment can have one or more elements. The elements inside could be run one or more times. A custom experiment can also contain another custom experiment as a sub-experiment. The sub-experiment can be run one or more times as well.

We will go through an example of building and running an experiment.

6.0.1 Creating A Custom Experiment

First, we will have some environment setup by creating our application:

```
#include "AisDeviceTracker.h"
#include "AisCustomExperiment.h"
#include "experiments/builder_elements/AisConstantCurrentElement.h"
#include "experiments/builder_elements/AisConstantPotElement.h"
char** test = nullptr;
int args;
QCoreApplication app(args, test);
```

To build a custom experiment, we need at least one element. In the example we will build below, we have two elements and a sub-experiment. The sub-experiment has the same two elements only with their parameters changed.

Let us go through it step by step:

We first create a constant voltage element and set its parameters as seen in the following code block. You can see a full list of the available elements in the classes section. For now, we are only setting the required parameters. You can get a complete list of settable parameters for any given element type by examining the corresponding element class.

```
// constructing a constant potential element with required arguments
AisConstantPotElement cvElement(
    5, // voltage:  5v
    1, // sampling interval:  1s
    10 // duration:  10s
);
```

Note

for each element you use, you need to include its corresponding header file.

We create another element of a different type.

```
// constructing a constant current element with required arguments
AisConstantCurrentElement ccElement(
    1, // current:  1A
    1, // sampling interval:  1s
    60 // duration:  60s
);
```

We create a custom experiment and add the previously created elements to it.

```
auto customExperiment = std::make_shared<AisExperiment>();  // at this point, it is an empty custom
      experiment, so, we add the elements we created to it.
customExperiment->appendElement(ccElement, 1); // append the CC element to the end of the experiment and set
      it to run 1 time
customExperiment->appendElement(cvElement, 1); // append the CV element to the end of the experiment and set
      it to run 1 time
```

Note

> Elements are run in the order that they are added to the experiment

Next, we create a second experiment as a sub-experiment i.e. we are going to then add it to the main experiment.

```
auto subExperiment = std::make_shared<AisExperiment>(); // this line creates a custom experiment, intended
      to be used as a sub-subExperiment
subExperiment.appendElement(ccElement, 2); // append the CC element to the sub-experiment and set it to run
      2 times
subExperiment.appendElement(cvElement, 3); // append the CV element to the sub-experiment and set it to run
      3 times
customExperiment->appendSubExperiment(&subExperiment, 2); // append the sub-experiment to the main
      experiment and set the sub-experiment to run 2 times.
```

Again, the order adding/appending the elements and the sub-experiment here corresponds to the order at which they will run. The sub-experiment and the elements it contains will be run after the elements already added to the main experiment

We create an additional constant voltage element with a different voltage setpoint.

```
AisConstantPotElement cvElement_2(
    4, // voltage:  4v
    1, // sampling interval:  1s
    10 // duration:  10s
);
```

This concludes creating the experiment. Next is how to control the workflow of the experiment.

6.0.2 Controlling The Experiment

So far, we have only created the experiment. But we need to start it and control it. The next code section employs a callback mechanism specific to Qt, called signals and slots. Callbacks are used to take an action when a specified condition is met i.e. control the workflow. For simplicity, we provided some common slots related to our API with comments inside, on what you can do. You can read more about Qt signals and slots in the following link: https://doc.qt.io/qt-5/signalsandslots.html

Reading this document should still cover most of what is needed. Basically, a signal can be emitted when an event happens. If a slot is connected to that signal, whatever is inside that slot will be executed when the signal is emitted. You can think of a signal as a condition and a slot is what will be executed once a corresponding condition is met. The only difference is the order of execution. Normal execution have sequential order. However, a slot can be emitted at anytime. Whenever that happens, the slot will execute no matter where the connection has been made (as long as a connection has been made prior). That is how we can have extra control on how and when things are executed.

An experiment is run on a specific channel of a device. You may have more than one device connected. A single device has up to 4 channels. Any channel on a specific device can run a single experiment at a time. To start an experiment, we specify the device and the channel and, then start it. To stop or pause that experiment, we need to specify its corresponding device and channel. We need to keep track of the device and channel for each experiment we start so we can control it later.

We can control a device, including starting, pausing and stopping an experiment on a specific channel using an AisInstrumentHandler A device/instrument handler can be created given a device name that we detect.

We have two parts below: one that creates logic using signals and slots. The second part assigns that logic to an instrument handler which will discuss in a bit. The first part below is creating some control-flow logic that we can assign to a handler. We can also create other logics in the same way that can be assigned to different handlers which can be used to control different devices. If we only have one device, all the logic will be handled with one handler. We can then have further control within, based on channels.

6.0.2.1 Creating Control Flow Logic Specific To A Handler

The first part is a lambda function called "connectHandlerSignals" which takes a handler as an argument and connects some of the handler signals to slots. We have other signals related to a handler you can add, which you can find in the AisInstrumentHandler This example logic has four conditions on which we can perform other tasks. That is, when we assign this logic to a specific handler, this logic will execute for that handler. The four signals and slots below in the first part are examples for you to follow in order to add other connections.

```
auto connectHandlerSignals = [=](AisInstrumentHandler* handler) {
    QObject::connect(handler, &AisInstrumentHandler::activeDCDataReady, [=](uint8_t channel, const
      AisDCData& data) {
        // do something when DC data are received, such as writing to a CSV file output
        // THIS IS WHERE YOU RECEIVE DC DATA FROM THE DEVICE
        //example:  print the data to the standard output as follows:
        qDebug() « "channel:  " « (int)channel « "current :" « data.current « "   voltage:  " «
      data.workingElectrodeVoltage « "   counter electrode :  " « data.counterElectrodeVoltage « "
      timestamp :  " « data.timestamp;
    });
    QObject::connect(handler, &AisInstrumentHandler::activeACDataReady, [=](uint8_t channel, const
      AisACData&) {
        // do something when AC data are received
        // THIS IS WHERE YOU RECEIVE AC (EIS) DATA FROM THE DEVICE
    });
    QObject::connect(handler, &AisInstrumentHandler::experimentNewElementStarting, [=](uint8_t channel,
      const AisExperimentNode&) {
        // do something when a new element is starting
        // for example, print to the standard output:  "New element starting"
        qDebug() « "New element starting";
    });
    QObject::connect(handler, &AisInstrumentHandler::experimentStopped, [=](uint8_t channel) {
        // do something when the experiment has stopped or has been stopped.  For example, you can invoke
      starting the next step in your workflow
        // print to the standard output:  "Experiment stopped Signal "
        qDebug() « "Experiment Stopped Signal " « channel;
    });
};
```

For a more complex logic for running a sequence of experiments, please refer to this example

If you would like to output the incoming data to a file such as a CSV file, you may modify the last block to something as follows:

```
QString filePath;
auto connectHandlerSignals = [=, &filePath](const AisInstrumentHandler* handler) {
    QObject::connect(handler, &AisInstrumentHandler::experimentNewElementStarting, [=, &filePath](uint8_t
      channel, const AisExperimentNode& node) {
        auto utcTime = handler->getExperimentUTCStartTime(0);
        auto name = "/" + QString::number(node.stepNumber) + " " + node.stepName + " " " +
      QString::number(utcTime) + ".csv";
        filePath = (QString(QStandardPaths::writableLocation(QStandardPaths::DesktopLocation)) + name);
        QFile file(filePath);
        if (!file.open(QIODevice::WriteOnly | QIODevice::Text)) // overwrite existing files with the same
      name
            return;
        QTextStream out(&file);
        out « "Time Stamp,"
            « "Counter Electrode Voltage,"
            « "Working Electrode Voltage,"
            « "Current"
            « "\n";
        file.close();
        qDebug() « "New element beginning:  " « node.stepName « "step:  " « node.stepNumber;
    });
    QObject::connect(handler, &AisInstrumentHandler::activeDCDataReady, [=, &filePath](uint8_t channel,
      const AisDCData& data) {
        qDebug() « "current :" « data.current « "   voltage:  " « data.workingElectrodeVoltage « "   counter
      electrode :  " « data.counterElectrodeVoltage « "  timestamp :  " « data.timestamp;
        QFile file(filePath);
        if (!file.open(QIODevice::Append | QIODevice::WriteOnly | QIODevice::Text))
            return;
        QTextStream out(&file);
        out « data.timestamp « ","
            « data.counterElectrodeVoltage « ","
            « data.workingElectrodeVoltage « ","
            « data.current
            « "\n";
        file.close();
    });
    QObject::connect(handler, &AisInstrumentHandler::activeACDataReady, [=](uint8_t channel, const
      AisACData& data) {
        qDebug() « data.frequency « "         " « data.absoluteImpedance « "        " « data.phaseAngle;
    });
    QObject::connect(handler, &AisInstrumentHandler::experimentStopped, [=](uint8_t channel) {
        qDebug() « "Experiment Completed Signal " « channel;
    });
};
```

Here we output the DC data to a CSV file but, you may do that for AC data as well in the same manner.

You may also find it useful to refer to C++ lambdas documentation: https://docs.microsoft.↩ com/en-us/cpp/cpp/lambda-expressions-in-cpp

6.0.2.2 Connecting Slots To Device-Tracker Signals

There are two signals related to a device tracker: when a device is connected and second, when a device is disconnected.

6.0.2.2.1 When a Device is Connected This connects a slot to the device tracker's AisDeviceTracker::newDeviceConnected signal that provides the device name. Because we have the device name, we can create a device handler and do whatever a handler can do. In this slot example, we are creating a handler, assigning the previously created logic to this handler and then starting an experiment.

```
QObject::connect(tracker, &AisDeviceTracker::newDeviceConnected, &app, [=](const QString& deviceName) {
    // Do something when a new device is detected to be connected.  The device name is given in the variable
      'deviceName'
    // The following lines start the experiment that we created
    auto handler = tracker->getInstrumentHandler(deviceName); // create a device handler using the given
      device name
    connectHandlerSignals(handler); // connect the signals we created for the device.  This is done once per
      device.
    auto error = handler->uploadExperimentToChannel(0, customExperiment); // upload to a specific channel
      (first arg) an experiment (second arg) on the given
    device controlled by the handler.
    if (error) {
        qDebug() « error.message();
        return;
    }
    auto error = handler->startUploadedExperiment(0); // start the previously uploaded experiment on the
      given channel.
    if (error) {
        qDebug() « error.message();
        return;
    }
});
```

Please refer to AisInstrumentHandler for possible errors that may occur when performing operations such as uploading and starting an experiment. For example, when uploading an experiment, AisInstrumentHandler::uploadExperimentToChannel may return an AisErrorCode::InvalidParameters error if the parameters are out of range where you can display the message to check which parameter was not supported for your device.

Note

> Specific to the cycler model, before starting an experiment, you have the option of linking channels so that you can share the electric current over multiple channels using AisInstrumentHandler::setLinkedChannels. If using paralleled channels, AisInstrumentHandler::setLinkedChannels MUST be called before each experiment that uses paralleled channels. To link channels on the cycler, you can modify the last code by first linking the channels, and then uploading and starting the experiment on the master channel for the linked channels:

```
auto masterchannel = handler->setLinkedChannels({ 0, 1 }); // this does two things, first links the given
      channels and second returns the masterchannel used to control the combined output.
handler->uploadExperimentToChannel(masterchannel, customExperiment);
handler->startUploadedExperiment(masterchannel);
```

6.0.2.2.2 When a Device is Disconnected The following code connects a slot to the device tracker's AisDeviceTracker::deviceDisconnected signal with the device name.

```
QObject::connect(tracker, &AisDeviceTracker::deviceDisconnected, &app, [=](const QString& deviceName) {
        // do something when a device has been disconnected.  The device name is given in the variable
      'deviceName'
        // for example, print to the standard output that the device given is disconnected
        qDebug() « deviceName « "is disconnected ";
    });
```

We still have not started the experiment, we've only created an experiment and setup callback functions via signals. When we connect a device using the tracker as shown below, the AisDeviceTracker::newDeviceConnected signal will be emitted with the device name. As a result, the slot we connected earlier to the signal AisDeviceTracker::newDeviceConnected will execute (connecting the other signals and running the experiment).

Note

> in the example we showed, the function connectHandlerSignals is intentionally called inside the AisDeviceTracker::newDeviceConnected slot because connectHandlerSignals needs a valid handler. When AisDeviceTracker::newDeviceConnected is emitted, we know we can get a device handler for the newly connected device and then control the device with the handler.

Now to connect the device, the easiest way to connect all plugged-in devices is to call AisDeviceTracker::connectAllPluggedInDevices←
:

```
tracker->connectAllPluggedInDevices();
```

To connect specific devices, you may alternatively call AisDeviceTracker::connectToDeviceOnComPort with a specific COM port.

```
tracker->connectToDeviceOnComPort("COM3"); // change the port number to yours.  For example, in windows, you
      can find it from the device manager
```

Finally, we can start the application by calling:

```
app.exec();
```

In the next section, we introduce a more advanced control flow.

Chapter 7

Manual Experiments

We have seen before the basics of running experiments. We created our custom experiment using prebuilt elements. These elements have presets for controlling the voltage and current. You can still do that yourself in real time, if you wish to do so, using manual experiments. With a manual experiment, you have the option of running in galvanostatic mode -where you can control the current- or potentiostatic mode where you can control the voltage.

First, we do environment setup as usual:

```
char** test = nullptr;
int args;
QCoreApplication app(args, test);
```

Since we are doing a manual experiment, we will not create a custom experiment but jump to creating the logic. The following is a simple logic:

```
auto createLogic = [=] (const AisInstrumentHandler* handler) {
    QObject::connect(handler, &AisInstrumentHandler::activeDCDataReady,  [=](uint8_t channel, const
      AisDCData& data) {
        qDebug() « "channel :  " « channel «" current :" « data.current « "   voltage:  " «
      data.workingElectrodeVoltage
            « "   counter electrode :  " « data.counterElectrodeVoltage « "  time-stamp :  " «
      data.timestamp;
    });
    QObject::connect(handler, &AisInstrumentHandler::activeACDataReady, [=](uint8_t channel, const
      AisACData& data) {
        qDebug() « data.frequency « "         " « data.absoluteImpedance « "        " « data.phaseAngle;
    });
    QObject::connect(handler, &AisInstrumentHandler::experimentNewElementStarting, [=](uint8_t channel,
      const AisExperimentNode&) {
        qDebug() « "New Node beginning ";
    });
    QObject::connect(handler, &AisInstrumentHandler::experimentStopped,  [=](uint8_t channel) {
        qDebug() « "Experiment Completed Signal " « channel;
    });
};
```

You can see more advanced logic in in the Advanced Control Flow.

Next, we will start the manual experiment after getting the handler:

```
QObject::connect(tracker, &AisDeviceTracker::newDeviceConnected, &app, [=](const QString& deviceName) {
    auto& handler = tracker->getInstrumentHandler(deviceName); // get an instrument handler once the device
      is connected
    createLogic(&handler); // assign the previously created logic to the handler.
    handler.startManualExperiment(1); // start a manual experiment on channel 1
    handler.setManualModeSamplingInterval(1, 2); // set manual experiment sampling interval on channel 1 to
      be 2 seconds
    handler.setManualModeConstantVoltage(1, 2); // on channel 1, set constant 2V
});
```

Note

> Unlike when creating elements, you set the parameters after starting the manual experiment because control is done in real time.

We can utilize timers to control the experiment and perform other manual operations:

```
// stop the experiment after 25 seconds
QTimer::singleShot(25000, [=,&handler]() {
    handler.stopExperiment(1);
});
```

You can see all the manual operations in AisInstrumentHandler

Finally, we start the application:

```
app.exec();
```

Chapter 8

Automatically Update Firmware

8.0.0.1 Introduction

Normally, when introducing a new Squidstat or switching to a different version of the API, we recommend that the user runs the Firmware Update example which will ensure that all connected Squidstats are up to date. However, there may be reasons that a user wishes to only operate on a single Squidstat or to simply increase the portability of their own program and eliminate this step. This example will take you through one way to automatically update an out of date Squidstat and then start the experiment when the update finishes.

8.0.0.2 Implementation

For this example, rather than describing the program top to bottom, we will only be covering the relevant parts for making the firmware automatically update. We will also follow the program's logical flow, meaning we will be starting just before the QT application starts at the bottom of the code and then move to our signal definition in the middle.

```
// Attempt to connect to the device just before starting the QT app.
auto error = tracker->connectToDeviceOnComPort(COMPORT);
if (error != error.Success) {
    if (error == error.FirmwareNotSupported) {
        qDebug() « "Firmware is out of date for the device on" « COMPORT;
        tracker->updateFirmwareOnComPort(COMPORT);
    }
    else {
        qDebug() « "Error:  " « error.message();
    }
}
a.exec();
```

At this step we try to connect to the Squidstat. There are several errors associated with connection, but at the moment we are only interested in the result if it was an out of date firmware response represented by AisErrorCode::FirmwareNotSupported. If this is the case, we are going to tell our tracker to update the firmware. This will kick off the update, so we want to jump into our application quickly after this so that we can see our firmware update messages. In the case that the firmware was already up to date, we will end up falling into our AisDeviceTracker::newDeviceConnected signal.

```
QObject::connect(tracker, &AisDeviceTracker::firmwareUpdateNotification, [=](const QString& message) {
    qInfo() « message;
    if (message.contains("firmware is updated.")) {
        const int retryCount = 3;
        // Give the Squidstat some time to reconnect
        AisErrorCode error(AisErrorCode::ConnectionFailed);
        for (int i = 0; i < retryCount && error == error.ConnectionFailed; i++) {
            QThread::sleep(1); //Give the last Squidstat a moment to re-establish the comport
            error = tracker->connectToDeviceOnComPort(COMPORT);
        }
        if (error != error.Success) {
            qDebug() « "Error:  " « error.message();
        }
    }
});
```

This is the crux of our automatic updating. We connect to our signal which will be sending us our firmware notifications. Each one is sent as a string by the API as the device is updating. We will print all of the messages, but the only relevant one to us is the one that contains the string "firmware is updated.". This will indicate that our firmware updating process is completed. At this step it is important to note that the API will not automatically re-establish connection with the device, so we will need to do that manually here using the same `tracker->connectToDeviceOnComPort(COMPORT);` call from earlier. It can take a little time for the updated Squidstat to return to its comport, so we use a wait time of 1 second, and try to reconnect 3 times. This should give the Squidstat enough time, but if you are getting the `AisErrorCode::ConnectionFailed` error after the retries are exhausted you may wish to increase either the retry count or the sleep time. If the issue persists, ensure that the device is still on the expected comport. Once the device reconnects, the tracker will emit the `AisDeviceTracker::newDeviceConnected` signal as it would if the firmware had been updated to begin with.

The remainder of the program will function as it does for most of the other examples, starting a small experiment and running it to completion.

8.0.0.3 Full Example

```
#include "AisInstrumentHandler.h"
#include "AisDeviceTracker.h"
#include "AisExperiment.h"
#include "experiments/builder_elements/AisConstantCurrentElement.h"
#include <QCoreApplication>
#include <QThread>
#include <QDebug>
#define COMPORT "COM5"
#define CHANNEL 0
int main()
{
    int args;
    QCoreApplication a(args, nullptr);
    auto tracker = AisDeviceTracker::Instance();
    // Custom Experiment with one constant current element
    std::shared_ptr<AisExperiment> experiment = std::make_shared<AisExperiment>();
    AisConstantCurrentElement ccElement(1, 1, 10);
    experiment->appendElement(ccElement, 1);
    // This set up the signals and slots for each device that gets connected
    auto createLogic = [=](const AisInstrumentHandler* handler) {
        QObject::connect(handler, &AisInstrumentHandler::activeDCDataReady, [=](uint8_t channel, const
      AisDCData& data) {
            auto utcTime = handler->getExperimentUTCStartTime(0);
            qDebug() « "current :" « data.current « "   voltage:   " « data.workingElectrodeVoltage « "
      counter electrode :  "
                « data.counterElectrodeVoltage « "  timestamp :   " « data.timestamp
                « " start UTC time:   " « qSetRealNumberPrecision(20) « utcTime;
        });
        QObject::connect(handler, &AisInstrumentHandler::experimentNewElementStarting, [=](uint8_t channel,
      const AisExperimentNode& data) {
            qDebug() « "New Node beginning " « data.stepName « " step number  " « data.stepNumber « " step
      sub :  " « data.substepNumber;
        });
        QObject::connect(handler, &AisInstrumentHandler::experimentStopped, [=](uint8_t channel) {
            qDebug() « "Experiment Completed on channel" « channel;
        });
    };
    // When a device is connected, create the signals and slots to print status messages, and then start the
      experiment.
    QObject::connect(tracker, &AisDeviceTracker::newDeviceConnected, &a, [=](const QString& deviceName) {
        auto& handler = tracker->getInstrumentHandler(deviceName);
        createLogic(&handler);
        handler.uploadExperimentToChannel(CHANNEL, experiment);
        qDebug() « "Starting experiment on" « deviceName « "channel" « CHANNEL+1;
        handler.startUploadedExperiment(CHANNEL);
    });
    // While a device is updating firmware, print out the messages.
    // When the update is complete, connect to the device, which will start the experiment
    QObject::connect(tracker, &AisDeviceTracker::firmwareUpdateNotification, [=](const QString& message) {
        qInfo() « message;
        if (message.contains("firmware is updated.")) {
            const int retryCount = 3;
            // Give the Squidstat some time to reconnect
            AisErrorCode error(AisErrorCode::ConnectionFailed);
            for (int i = 0; i < retryCount && error == error.ConnectionFailed; i++) {
                QThread::sleep(1); //Give the last Squidstat a moment to re-establish the comport
                error = tracker->connectToDeviceOnComPort(COMPORT);
            }
            if (error != error.Success) {
                qDebug() « "Error:  " « error.message();
            }
        }
    });
    QObject::connect(tracker, &AisDeviceTracker::deviceDisconnected, &a, [=](const QString& deviceName) {
        qDebug() « deviceName « "is disconnected ";
    });
    // Attempt to connect to the device just before starting the QT app.
    auto error = tracker->connectToDeviceOnComPort(COMPORT);
    if (error != error.Success) {
        if (error == error.FirmwareNotSupported) {
            qDebug() « "Firmware is out of date for the device on" « COMPORT;
            tracker->updateFirmwareOnComPort(COMPORT);
        }
        else {
            qDebug() « "Error:  " « error.message();
        }
    }
    a.exec();
}
```

Chapter 9

Advanced Control Flow

This page assumes familiarity with concepts covered in the basics. This shows how to run a sequence of experiments and controlling when to stop an experiment and start another based on external conditions. For simplicity, we are assuming having a single device connected and we are running on a single channel. So, we will not have to keep track of devices and channels. We will just focus on running and controlling the workflow of different experiments.

First we will set the environment and create the experiments:

```
// environment setup: creating the app
char** test = nullptr;
int args;
QCoreApplication app(args, test);
// constructing a constant potential element with required arguments
AisConstantPotElement cvElement(
    5, // voltage: 5v
    1, // sampling interval: 1s
    10 // duration: 10s
);
// constructing a constant current element with required arguments
AisConstantCurrentElement ccElement(
    0.002, // current: 0.002A
    1, // sampling interval: 1s
    10 // duration: 10s
);
auto experimentA = std::make_shared<AisExperiment>(); // create a custom experiment
experimentA->appendElement(cvElement, 1); // append to experimentA, the created CV element and set it to run 1 time
auto experimentB = std::make_shared<AisExperiment>(); // create a second experiment
experimentB->appendElement(ccElement, 1); // append to experimentB, the created CC element and set it to run 1 time
auto experimentC = std::make_shared<AisExperiment>(); // create a third experiment
experimentC->appendElement(cvElement, 2); // append to experimentC, the created CV element and set it to run 2 times
```

Now we have the experiments set up. Next we will create the logic for the sequence of experiments. We will be using timers as external conditions to control the workflow. You may substitute that with your own conditions.

The following lambda function creates a logic and assigns it to the given handler. We will call this function after the AisDeviceTracker::newDeviceConnected signal has been emitted and a handler has been created. The workflow will be as follows:

- Start the first timer
- Once the timer times out, start Experiment A
- Once Experiment A completes, start the second timer
- Once the second timer times out, start Experiment B
- Start a third timer to stop Experiment B early
- Once the third timer times out, stop Experiment B

- Start a fourth timer
- Once the fourth timer times out, start Experiment C
- Once Experiment C completes, start Experiment B

```
// Lambda function
auto createLogic = [&](AisInstrumentHandler* handler) {
    QTimer* timer1 = new QTimer(); // the first timer is used in lieu of the first external condition
    timer1->setSingleShot(true);
    timer1->start(1000);
    QObject::connect(timer1, &QTimer::timeout, [=]() {
        qDebug() « "Initial condition met.  Starting Experiment A ";
        handler->uploadExperimentToChannel(0, experimentA);
        handler->startUploadedExperiment(0);
        // once the first experiment is completed (Experiment A), start the next experiment (Experiment B).
        // this signal will be emitted for any experiment not just A so, we will track of the sequence with
      experimentStep
        // once an experiment has completed or has been stopped, continue to the next experimentStep
        QObject::connect(handler, &AisInstrumentHandler::experimentStopped, [&](uint8_t channel) {
            static int experimentStep = 0;
            qDebug() « "Experiment Step " « experimentStep « " Completed";
            experimentStep++; //increment the experiment step
            if (experimentStep == 1) {
                // Wait for external start condition
                QTimer* timer = new QTimer(); // the timer is used in lieu of an external condition
                timer->setSingleShot(true);
                timer->start(10000); // when this timer times out, the next experiment (Experiment B) will
      start
                // Create an external condition that will stop the upcoming experiment early
                QTimer* StopEarlyTimer = new QTimer();
                StopEarlyTimer->setSingleShot(true);
                QObject::connect(StopEarlyTimer, &QTimer::timeout, [&]() {
                    qDebug() « "External early stop condition met";
                    handler->StopExperiment(0); // Once the external condition is met, experiment B will
      stop, and the experimentCompleted signal will be emitted
                });
                QObject::connect(timer, &QTimer::timeout, [&,StopEarlyTimer]() {
                    qDebug() « "External condition met, starting experiment B";
                    handler->uploadExperimentToChannel(0, experimentB); // start Experiment B
                    handler->startUploadedExperiment(0);
                    StopEarlyTimer->start(2000);
                });
            } else if (experimentStep == 2) {
                QTimer* timer = new QTimer(); // the timer is used in lieu of an external condition
                timer->setSingleShot(true);
                timer->start(10000); // when this timer times out, the next experiment (Experiment C) will
      start
                QObject::connect(timer, &QTimer::timeout, [&]() {
                    qDebug() « "External condition met, starting Experiment C ";
                    handler->uploadExperimentToChannel(0, experimentC); // start Experiment C
                    handler->startUploadedExperiment(0);
                });
            } else if (experimentStep == 3) {
                QTimer* timer = new QTimer(); // the timer is used in lieu of an external condition
                timer->setSingleShot(true);
                timer->start(10000); // when this timer times out, the next experiment (Experiment B) will
      start
                QObject::connect(timer, &QTimer::timeout, [&]() {
                    qDebug() « "External condition met, starting Experiment B ";
                    handler->uploadExperimentToChannel(0, experimentB); // start Experiment B
                    handler->startUploadedExperiment(0);
                });
            }
        });
    });
};
```

This logic we have shown demonstrates how to start and stop experiments based on external conditions/variables, and how to do so based on the behavior of other experiments as well.

We then connect the tracker's signals as we have explained in more details before.

```
// this is a signal-slot connection where the slot assigns the logic to the device handler when
      newDeviceConnected signal is emitted.
auto tracker = AisDeviceTracker::Instance(); // create a tracker that tracks connected devices
QObject::connect(tracker, &AisDeviceTracker::newDeviceConnected, &app, [&](const QString& deviceName) {
    auto handler = tracker->getInstrumentHandler(deviceName);
    createLogic(handler); // controlling experiments is to be done only after a device handler has been
      assigned.  That is why it is placed inside this slot.
    });
// here we have a signal and slot for when a device has been disconnected
QObject::connect(tracker, &AisDeviceTracker::deviceDisconnected, &app, [=](const QString& deviceName) {
    qDebug() « deviceName « "is disconnected ";
    });
```

```
tracker->connectToDeviceOnComPort("COM3"); // change the port number to your device.  For example, in
      windows, you can find it from the device manager
```

Finally, you can start the application as follows:

```
app.exec();
```

Note however that this will hold your execution thread. That would be fine if this is your main application or if you have previously spawned a thread specifically for this application. Alternatively, you can start the application as follows:

```
// process events while channel 0 is busy
while (handler.isChannelBusy(0)) {
    app.processEvents();
}
app.processEvents();
```

You can learn more about Qt app execution here: https://doc.qt.io/qt-5/qcoreapplication.html#static-public-members

Chapter 10

Python Example

10.0.0.1 Introduction

The following example will help illustrate the use of the SquidstatPyLibrary with Python. SquidstatPyLibrary is currently supported on windows platform.

10.0.0.2 How To Use Squidstatlibrary with Python.

1. To use the SquidstatPyLibrary library, you need to install Python version >= 3.7 and < 3.11.
   - Visit the official Python website at `https://www.python.org/downloads/`.
   - Download the installer for the desired Python version (>= 3.7 and < 3.11).
   - Run the installer and follow the installation wizard's instructions.
   - Make sure to select the option to add Python to the system environmental (PATH) variables during the installation process. This will enable you to run Python from any location on your computer.
2. Make sure you have installed Python correctly by checking the Python version using `python -V` command.
3. Now you can choose to install the library in either the global environment or a virtual environment. If you want to install the library in the global environment, you can skip this step.
   - If you prefer using a virtual environment, you can create a virtual environment by running the command: `python -m venv VIRTUAL_ENVIRONMENT_NAME`
   - Open the command prompt and activate the virtual environment by typing: `./VIRTUAL_↩ ENVIRONMENT_NAME/Scripts/activate.bat`
4. Now you can proceed to install the SquidstatPyLibrary. You can download the .whl file from `here`. After downloading, you can install it using the command `pip install YOUR_DOWNLOADED_FILE.whl`
5. Now, let's run an example script. If you have an Experiment.py file that you created to run an experiment, you can execute that script by using the command `python Experiment.py`.

The necessary Python library files are also located inside the `pythonWrapper` directory.

Now We will go through an example of building and running an experiment.

10.0.0.3 Building a Custom Experiment with Python

10.0.0.3.1 Import all the required basic modules from SquidstatPyLibrary.

```
import sys
import struct
from PySide2.QtWidgets import QApplication
from SquidstatPyLibrary import AisDeviceTracker
from SquidstatPyLibrary import AisCompRange
from SquidstatPyLibrary import AisDCData
from SquidstatPyLibrary import AisACData
from SquidstatPyLibrary import AisExperimentNode
from SquidstatPyLibrary import AisErrorCode
from SquidstatPyLibrary import AisExperiment
from SquidstatPyLibrary import AisInstrumentHandler
```

10.0.0.3.2 Import experiment modules depeneding on the requirement.

```
from SquidstatPyLibrary import
      AisConstantPotElement
from SquidstatPyLibrary import AisEISPotentiostaticElement
from SquidstatPyLibrary import AisConstantCurrentElement
```

10.0.0.3.3 Create the custom experiment.

```
experiment = AisExperiment();
cvElement = AisConstantPotElement(5, 1, 10)
eisElement = AisEISPotentiostaticElement(10000, 1, 10, 0.15, 0.1);
ccElement = AisConstantCurrentElement(1, 1, 10);
subExperiment = AisExperiment()
subExperiment.appendElement(ccElement, 1);
subExperiment.appendElement(cvElement, 1);
experiment.appendElement(ccElement,1)
experiment.appendElement(cvElement,1)
experiment.appendSubExperiment(subExperiment, 2)  # Here we repeating sub experiment 2 times
experiment.appendElement(eisElement,1)
```

10.0.0.3.4 Get the Instrument Handler and connect the required signal to receive data from the Device.

```
app = QApplication()
tracker = AisDeviceTracker.Instance()
tracker.newDeviceConnected.connect(lambda deviceName:  print("Device is Connected:  %s" % deviceName))
tracker.connectToDeviceOnComPort("COM19")
handler = tracker.getInstrumentHandler("Ace1102");
handler.activeDCDataReady.connect(lambda channel, data:  print("timestamp:",
      "{:.9f}".format(data.timestamp), "workingElectrodeVoltage:  ",
      "{:.9f}".format(data.workingElectrodeVoltage)))
handler.activeACDataReady.connect(lambda channel, data:  print("frequency:",
      "{:.9f}".format(data.frequency), "absoluteImpedance:  ", "{:.9f}".format(data.absoluteImpedance),
      "phaseAngle:  ", "{:.9f}".format(data.phaseAngle)))
handler.experimentNewElementStarting.connect(lambda channel, data:  print("New Node beginning:",
      data.stepName, "step number:  ", data.stepNumber, " step sub :  ", data.substepNumber))
handler.experimentStopped.connect(lambda channel :  print("Experiment Completed:  %d" % channel))
handler.uploadExperimentToChannel(0,experiment)
handler.startUploadedExperiment(0)
sys.exit(app.exec_())
```

10.0.0.3.5 Full Example

Here is everything put together. You can also find this in the pythonWrapper directory.

```
import sys
import struct
from PySide2.QtWidgets import QApplication
from SquidstatPyLibrary import AisDeviceTracker
from SquidstatPyLibrary import AisCompRange
from SquidstatPyLibrary import AisDCData
from SquidstatPyLibrary import AisACData
from SquidstatPyLibrary import AisExperimentNode
from SquidstatPyLibrary import AisErrorCode
from SquidstatPyLibrary import AisExperiment
from SquidstatPyLibrary import AisInstrumentHandler
from SquidstatPyLibrary import AisConstantPotElement
from SquidstatPyLibrary import AisEISPotentiostaticElement
from SquidstatPyLibrary import AisConstantCurrentElement
app = QApplication()
tracker = AisDeviceTracker.Instance()
tracker.newDeviceConnected.connect(lambda deviceName:  print("Device is Connected:  %s" % deviceName))
tracker.connectToDeviceOnComPort("COM19")
handler = tracker.getInstrumentHandler("Ace1102");
handler.activeDCDataReady.connect(lambda channel, data:  print("timestamp:",
      "{:.9f}".format(data.timestamp), "workingElectrodeVoltage:  ",
      "{:.9f}".format(data.workingElectrodeVoltage)))
handler.activeACDataReady.connect(lambda channel, data:  print("frequency:",
      "{:.9f}".format(data.frequency), "absoluteImpedance:  ", "{:.9f}".format(data.absoluteImpedance),
      "phaseAngle:  ", "{:.9f}".format(data.phaseAngle)))
handler.experimentNewElementStarting.connect(lambda channel, data:  print("New Node beginning:",
      data.stepName, "step number:  ", data.stepNumber, " step sub :  ", data.substepNumber))
handler.experimentStopped.connect(lambda channel :  print("Experiment Completed:  %d" % channel))
experiment = AisExperiment();
cvElement = AisConstantPotElement(5, 1, 10)
eisElement = AisEISPotentiostaticElement(10000, 1, 10, 0.15, 0.1);
```

```
ccElement = AisConstantCurrentElement(1, 1, 10);
subExperiment = AisExperiment()
subExperiment.appendElement(ccElement, 1);
subExperiment.appendElement(cvElement, 1);
experiment.appendElement(ccElement,1)
experiment.appendElement(cvElement,1)
experiment.appendSubExperiment(subExperiment, 2)
experiment.appendElement(eisElement,1)
handler.uploadExperimentToChannel(0,experiment)
handler.startUploadedExperiment(0)
sys.exit(app.exec_())
```

Chapter 11

Operating Squidstats Remotely

11.0.0.1 Introduction

Although the Squidstat cannot directly communicate over a network, it is possible to create a simple server-client interface that can allow a remote computer to configure and run experiments over a network. In this example, we will set up a server and client via Python's socket library and run a predefined Open Circuit Potential experiment with a variable `duration` specified by a client. The server will be responsible for managing the Squidstat. When the experiment finishes, both the client and the server terminate. The full example can be found at both the bottom of this page and in the example folder of the API. *Note: This example assumes that the Squidstat is already running the appropriate firmware already and so it does not cover updating the firmware.*

11.0.0.2 Server Implementation

Toward the beginning of our server file, we have some definitions that you will need to change accordingly to match your settings:

```
# Define the server address and port
HOST = 'localhost'
PORT = 12345
```

The first two will define the address and port that the server listens on. They should match the ones in the client file (See Client Implementation). A few notes:

1. This example assumes that the server and client are both on the same computer. That will almost certainly not be the case for your system, so you will need to change these to match your local connection. For example, if your server computer is located at `10.0.1.5` that is the address you will use here.
2. The port must not be in use by another program running on the server.
3. If your server and client are not located on local networks (E.G. behind a NAT router over an ISP's network) you will most likely need to portforward your chosen port on your router.
4. A firewall may block incoming connections from other network devices. If you are having connection issues, you should try adding an exception to the firewall for this program at the chosen port.

```
# The comport the Squidstat is connected to
SQUIDCOMPORT = "COM4"
SQUIDNAME = "Plus1700"
```

`SQUIDCOMPORT` and `SQUIDNAME` represent how the server will communicate with the Squidstat. They must match the Squidstat's COM port and serial number. On Windows the COM port can be found through device manager.

```
# This will build a start the Open Circuit Potential experiment
def start_ocp_experiment(handler, durationSec=60):
    # Create an experiment with elements
    experiment = AisExperiment()
    ocpElement = AisOpenCircuitElement(durationSec, 1)
    experiment.appendElement(ocpElement, 1)
    # Upload the experiment to channel 0
    error = handler.uploadExperimentToChannel(0, experiment)
    if error.value() != AisErrorCode.ErrorCode.Success:
        return error
    # Start the experiment
    return(handler.startUploadedExperiment(0))
```

`start_ocp_experiment` will create and start an Open Circuit Potential experiment on channel 1 of the Squidstat. The duration is passed in via the duration variable. This function will be called when the client sends a `startExperiment` command.

```
# Send a specified command to our Squidstat
def command_to_device(command, handler):
    #Check if we had an argument associated with the command
    splitCommand = command.split(" ")
    action = splitCommand[0]
    actionArg = 0
    if(len(splitCommand) > 1):
        try:
            actionArg = int(splitCommand[1])
        except:
            actionArg = 0
    response = None
    if action == 'startExperiment':
        #print("Starting experiment...")
        response = start_ocp_experiment(handler, actionArg)
    elif action == 'stopExperiment':
        #print("Stopping experiment...")
        response = handler.stopExperiment(0)
    else:
        #print("Invalid command:", command)
        pass
    return response
```

`command_to_device` is the function that controls the communication to the Squidstat. It takes in a command as plain text which determines how the software will interact with the Squidstat. You may optionally choose to uncomment the print statements to make the server more verbose.

```
# Listen for the client's messages, and disconnect signals and terminate program when finished
def handle_client(handler, client_socket):
    print("Client connected")
    while True:
        # Receive data from the client
        try:
            data = client_socket.recv(1024).decode()
        except ConnectionResetError:
            break
        # Check if the client has closed the connection
        if not data:
            break
        # Handle the command
        handle_command(data, handler, client_socket)
    handler.activeDCDataReady.disconnect()
    handler.activeACDataReady.disconnect()
    handler.experimentNewElementStarting.disconnect()
    handler.experimentStopped.disconnect()
    command_to_device("stopExperiment", handler)
    # Close the client socket
    client_socket.close()
    print("Client disconnected")
    os._exit(1)
```

`handle_client` listens for commands from the client. Once the client has established a connection, it will loop until either the program terminates, typically through experiment completion, or the client drops the session.

```
def send_data_to_client(client_socket, event_type, data):
    if event_type == "DCData":
        message = "timestamp: {:.9f}, workingElectrodeVoltage: {:.9f}".format(data.timestamp,
      data.workingElectrodeVoltage)
    elif event_type == "ACData":
        message = "frequency: {:.9f}, absoluteImpedance: {:.9f}, phaseAngle:
      {:.9f}".format(data.frequency, data.absoluteImpedance, data.phaseAngle)
    elif event_type == "NewElement":
        message = "New Node beginning: {}, step number: {}, step sub: {}".format(data.stepName,
      data.stepNumber, data.substepNumber)
    elif event_type == "ExperimentCompleted":
        message = "Experiment Completed: {}".format(data)
    else:
        return
    client_socket.send(message.encode())
```

send_data_to_client is the transmission function for all data that is coming from the experiments. These are hooked up via a QT signal/slot connection. This function is called each time the device sends a signal that there is information ready to be processed. event_type is our hint as to which type of data/message we are processing.

```
# Create the device tracker and connect to the Squidstat we will be using
print(f"Attempting to connect to the Squidstat {SQUIDNAME} on {SQUIDCOMPORT}...")
tracker = AisDeviceTracker.Instance()
tracker.newDeviceConnected.connect(lambda deviceName:  print("Device is Connected:  %s" % deviceName))
error = tracker.connectToDeviceOnComPort(SQUIDCOMPORT)
if error.value() != AisErrorCode.ErrorCode.Success:
    print(error.message())
    exit()
# Create the instrument handler
handler = tracker.getInstrumentHandler(SQUIDNAME)
print("Connection successful\n")
```

Here we establish our connection to the Squidstat and print out any error that may result when attempting it.

```
# Create the TCP/IP socket and bind it to our host
print("Starting server...")
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
activeSockets.append(server_socket)
server_socket.bind((HOST, PORT))
# Listen for incoming connections
server_socket.listen(1)


...
# Accept a client connection
client_socket, client_address = server_socket.accept()
activeSockets.append(client_socket)
# Connect the signals to send data to the client
handler.activeDCDataReady.connect(lambda channel, data:  send_data_to_client(client_socket, "DCData", data))
handler.activeACDataReady.connect(lambda channel, data:  send_data_to_client(client_socket, "ACData", data))
handler.experimentNewElementStarting.connect(lambda channel, data:  send_data_to_client(client_socket,
      "NewElement", data))
handler.experimentStopped.connect(lambda channel:  send_data_to_client(client_socket, "ExperimentCompleted",
      channel))
# Start the listening process in a separate thread
listening_thread = threading.Thread(target=handle_client, args=(handler, client_socket))
listening_thread.start()
```

We then open the server port, accept our client, and set up the QT connections. Our client listener handle_client is sent to execute on its own thread.

11.0.0.3 Client Implementation

In this section we will go over some of the functional aspects of the client.

```
SERVER_HOST = "localhost"
SERVER_PORT = 12345
```

At the beginning of the client file, we have some definitions that must mirror the server. See Server Implementation for more details.

```
def send_command(command):
    # Send the command to the server
    try:
        client_socket.send(command.encode())
    except:
        print("Connection was closed by host")
        os._exit(1)
    # Receive and print the response from the server
    response = client_socket.recv(1024).decode()
    print("Server response:", response)
```

send_command is exactly as it sounds. Once we establish the connection to the server, this function sends our commands to the server, listens to the response, and prints it. Note that this is somewhat different from the listening thread that prints the remote Squidstat's active data. This example assumes that all commands are sent prior to sending the startExperiment command, and calling this function after the start of the experiment can cause unexpected behavior due to having two recv functions running at the same time.

```
try:
    client_socket.connect((SERVER_HOST, SERVER_PORT))
except Exception as ex:
    print("Unable to establish connection to server:\n%s" % ex)
    exit()
```

Establish our connection to the server. If the server is not running or some problem occurs, we will terminate the program now.

```
send_command(f'startExperiment {duration}')
```

After we get the duration from the user at the terminal, we will kick off the experiment by sending the 'start←↩ Experiment' command to the server. At this point, the server will translate the message and call the appropriate function to notify the Squidstat.

```
while True:
    try:
        data = client_socket.recv(1024).decode()
    except (ConnectionAbortedError, BrokenPipeError):
        # This exception will be raised when the user presses <ENTER>
        print("Finishing connection")
        break
    except ConnectionResetError:
        print("The server closed the connection suddenly.")
        break
    if not data:
        break
    # Handle the data that was received.
    print(data)
    if("Experiment Completed:  " in data):
        break
```

Finally, we start a loop that will listen to the server, which at this point will be transmitting the experiment data and the stop response. When we get the data we will simply print it, but this could be modified to any other data handling function. When we get the stop response we can break the loop which will terminate the program.

11.0.0.4 Full Example

11.0.0.4.1 TCP_Server.py

```
import os
import socket
import threading
from PySide2.QtWidgets import QApplication
from SquidstatPyLibrary import AisDeviceTracker
from SquidstatPyLibrary import AisExperiment
from SquidstatPyLibrary import AisOpenCircuitElement
from SquidstatPyLibrary import AisErrorCode
# Define the server address and port
HOST = 'localhost'
PORT = 12345
# The comport the Squidstat is connected to
SQUIDCOMPORT = "COM4"
SQUIDNAME = "Plus1700"
# Create the QT application
app = QApplication([])
activeSockets = []
# This will build a start the Open Circuit Potential experiment
def start_ocp_experiment(handler, durationSec=60):
    # Create an experiment with elements
    experiment = AisExperiment()
    ocpElement = AisOpenCircuitElement(durationSec, 1)
    experiment.appendElement(ocpElement, 1)
    # Upload the experiment to channel 0
    error = handler.uploadExperimentToChannel(0, experiment)
    if error.value() != AisErrorCode.ErrorCode.Success:
        return error
    # Start the experiment
    return(handler.startUploadedExperiment(0))
# Send a specified command to our Squidstat
def command_to_device(command, handler):
    #Check if we had an argument associated with the command
    splitCommand = command.split(" ")
    action = splitCommand[0]
    actionArg = 0
    if(len(splitCommand) > 1):
        try:
            actionArg = int(splitCommand[1])
        except:
            actionArg = 0
    response = None
    if action == 'startExperiment':
        #print("Starting experiment...")
        response = start_ocp_experiment(handler, actionArg)
    elif action == 'stopExperiment':
        #print("Stopping experiment...")
        response = handler.stopExperiment(0)
    else:
        #print("Invalid command:", command)
        pass
    return response
# Handle commands from the client
def handle_command(command, handler, client_socket):
    # Send a response back to the client
    responseMsg = "Unknown Command"
    response = command_to_device(command, handler)
    if(response != None):
        responseMsg = response.message()
    response = "{}".format(responseMsg)
    client_socket.send(response.encode())
# Listen for the client’s messages, and disconnect signals and terminate program when finished
def handle_client(handler, client_socket):
    print("Client connected")
    while True:
        # Receive data from the client
        try:
            data = client_socket.recv(1024).decode()
        except ConnectionResetError:
            break
        # Check if the client has closed the connection
        if not data:
            break
        # Handle the command
        handle_command(data, handler, client_socket)
    handler.activeDCDataReady.disconnect()
    handler.activeACDataReady.disconnect()
    handler.experimentNewElementStarting.disconnect()
    handler.experimentStopped.disconnect()
    command_to_device("stopExperiment", handler)
    # Close the client socket
    client_socket.close()
    print("Client disconnected")
    os._exit(1)
# Send data the the client based on the type of event (Hooked up to signals)
def send_data_to_client(client_socket, event_type, data):
    if event_type == "DCData":
        message = "timestamp:  {:.9f}, workingElectrodeVoltage:  {:.9f}".format(data.timestamp,
      data.workingElectrodeVoltage)
    elif event_type == "ACData":
        message = "frequency:  {:.9f}, absoluteImpedance:  {:.9f}, phaseAngle:
      {:.9f}".format(data.frequency, data.absoluteImpedance, data.phaseAngle)
    elif event_type == "NewElement":
        message = "New Node beginning:  {}, step number:  {}, step sub:  {}".format(data.stepName,
      data.stepNumber, data.substepNumber)
    elif event_type == "ExperimentCompleted":
        message = "Experiment Completed:  {}".format(data)
    else:
        return
    client_socket.send(message.encode())
def terminate_program():
    print("Press <CTRL>+c to close the server")
    try:
        while True:
            input()
    except (EOFError, KeyboardInterrupt):
        pass
    for socket in activeSockets:
        socket.close()
    app.quit()
    os._exit(1)
# Create the device tracker and connect to the Squidstat we will be using
print(f"Attempting to connect to the Squidstat {SQUIDNAME} on {SQUIDCOMPORT}...")
tracker = AisDeviceTracker.Instance()
tracker.newDeviceConnected.connect(lambda deviceName:  print("Device is Connected:  %s" % deviceName))
error = tracker.connectToDeviceOnComPort(SQUIDCOMPORT)
if error.value() != AisErrorCode.ErrorCode.Success:
    print(error.message())
    exit()
# Create the instrument handler
handler = tracker.getInstrumentHandler(SQUIDNAME)
print("Connection successful\n")
# Create the TCP/IP socket and bind it to our host
print("Starting server...")
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
activeSockets.append(server_socket)
server_socket.bind((HOST, PORT))
# Listen for incoming connections
server_socket.listen(1)
print("Server started successfully.  Waiting for client connection...")
terminal_thread = threading.Thread(target=terminate_program)
terminal_thread.start()
# Accept a client connection
client_socket, client_address = server_socket.accept()
activeSockets.append(client_socket)
# Connect the signals to send data to the client
handler.activeDCDataReady.connect(lambda channel, data:  send_data_to_client(client_socket, "DCData", data))
handler.activeACDataReady.connect(lambda channel, data:  send_data_to_client(client_socket, "ACData", data))
handler.experimentNewElementStarting.connect(lambda channel, data:  send_data_to_client(client_socket,
      "NewElement", data))
handler.experimentStopped.connect(lambda channel:  send_data_to_client(client_socket, "ExperimentCompleted",
      channel))
# Start the listening process in a separate thread
listening_thread = threading.Thread(target=handle_client, args=(handler, client_socket))
listening_thread.start()
# Start the QT event loop
app.exec_()
```

11.0.0.4.2 TCP_Client.py

```
import os
import socket
import threading
import time
# Create a TCP/IP socket
client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
activeSockets = [client_socket]
# Define the server address and port
SERVER_HOST = "localhost"
SERVER_PORT = 12345
# Function to send a command to the server
def send_command(command):
    # Send the command to the server
    try:
        client_socket.send(command.encode())
    except:
        print("Connection was closed by host")
        os._exit(1)
    # Receive and print the response from the server
    response = client_socket.recv(1024).decode()
    print("Server response:", response)
def interupt_listener():
    print("Press <CTRL>+c to stop the program at any time.")
    try:
        while True:
            input()
    except (EOFError, KeyboardInterrupt):
        pass
    for socket in activeSockets:
        socket.close()
    os._exit(1)
# Try and open a socket to the server
try:
    client_socket.connect((SERVER_HOST, SERVER_PORT))
except Exception as ex:
    print("Unable to establish connection to server:\n%s" % ex)
    exit()
print("Connected to the server.")
# Get a duration from the user
duration = 0
while duration == 0:
    try:
        duration = int(input("Enter a duration for the Open Circuit Potential:  "))
    except ValueError:
        duration = 0
    if(duration < 1):
        print("Invalid entry.")
        duration = 0
# Send the start command to the server with the duration
send_command(f’startExperiment {duration}’)
interupt_thread = threading.Thread(target=interupt_listener)
interupt_thread.start()
# Listen for information from the server, which at this point will be data and the experiment stop message
while True:
    try:
        data = client_socket.recv(1024).decode()
    except (ConnectionAbortedError, BrokenPipeError):
        # This exception will be raised when the user presses <ENTER>
        print("Finishing connection")
        break
    except ConnectionResetError:
        print("The server closed the connection suddenly.")
        break
    if not data:
        break
    # Handle the data that was received.
    print(data)
    if("Experiment Completed:  " in data):
        break
os._exit(1)
```

Chapter 12

Hierarchical Index

12.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 13

Class Index

13.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 14

File Index

14.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 15

Class Documentation

15.1 AisACData Struct Reference

a structure containing AC data information.

```
#include <AisDataPoints.h>
```

Public Attributes

- double **timestamp**

  the time at which the AC data arrived.
- double **frequency**

  the applied frequency in Hz.
- double **absoluteImpedance**

  the magnitude of the complex impedance.
- double **realImpedance**

  the real part of the complex impedance.
- double **imagImpedance**

  the imaginary part of the complex impedance.
- double **phaseAngle**

  the phase angle between the real and the imaginary parts of the impedance.
- double **totalHarmonicDistortion**

  the percentage of the total harmonic distortion in the AC signal.
- double numberOfCycles

  the number of cycles specific to the reported frequency.
- double **workingElectrodeDCVoltage**

  the DC working electrode voltage in volts.
- double **DCCurrent**

  the DC electric current value in Amps
- double **currentAmplitude**

  the amplitude of the AC current.
- double **voltageAmplitude**

  the amplitude of the AC voltage.

15.1.1 Detailed Description

a structure containing AC data information.

15.1.2 Member Data Documentation

15.1.2.1 numberOfCycles

```
double AisACData::numberOfCycles
```

the number of cycles specific to the reported frequency.

In EIS, we run a range of frequencies. For each frequency, a specific number of cycles are run. The higher the frequency, the more number of cycles.

The documentation for this struct was generated from the following file:

- AisDataPoints.h

15.2 AisCompRange Class Reference

This class has advanced options controlling the device stability including the bandwidth index and the stability factor.

```
#include <AisCompRange.h>
```

Public Member Functions

- AisCompRange (const QString &compRangeName, uint8_t bandwidthIndex, uint8_t stabilityFactor)

  constructor for the compensation-range object.
- **AisCompRange** (const AisCompRange &)

  copy constructor for the compensation-range object.
- uint8_t getBandwidthIndex () const

  get the value set for the bandwidth index.
- void setBandwidthIndex (uint8_t index)

  set the index value for the bandwidth.
- uint8_t getStabilityFactor () const

  get the value set for the stability factor.
- void setStabilityFactor (uint8_t factor)

  set a value for the stability factor.
- void setCompRangeName (const QString &compRangeName)

  set a name for the compensation range for reference purposes.
- const QString & getCompRangeName () const

  get the name set for the compensation range.

15.2.1 Detailed Description

This class has advanced options controlling the device stability including the bandwidth index and the stability factor.

See also

setBandwidthIndex
setStabilityFactor

15.2.2 Constructor & Destructor Documentation

15.2.2.1 AisCompRange()

```
AisCompRange::AisCompRange (
            const QString & compRangeName,
            uint8_t bandwidthIndex,
            uint8_t stabilityFactor )  [explicit]
```

constructor for the compensation-range object.

Parameters

compRangeName	a name to set for the compensation range for reference purposes.
bandwidthIndex	the index value for the bandwidth.
stabilityFactor	the factor value for the stability.

See also

setBandwidthIndex
setStabilityFactor

15.2.3 Member Function Documentation

15.2.3.1 getBandwidthIndex()

```
uint8_t AisCompRange::getBandwidthIndex ( ) const
```

get the value set for the bandwidth index.

Returns

the set value for the bandwidth index.

See also

setBandwidthIndex

15.2.3.2 getCompRangeName()

```
const QString & AisCompRange::getCompRangeName ( ) const
```

get the name set for the compensation range.

Returns

the name set for the compensation range.

15.2.3.3 getStabilityFactor()

```
uint8_t AisCompRange::getStabilityFactor ( ) const
```

get the value set for the stability factor.

Returns

the value set for the stability factor.

15.2.3.4 setBandwidthIndex()

```
void AisCompRange::setBandwidthIndex (
            uint8_t index )
```

set the index value for the bandwidth.

Usually, the device's default index value is optimal for running experiments. You may still increase the index within the range 0-10 as you run higher frequency experiments to see what best fits.

Parameters

index	the index value for the bandwidth (0-10).

15.2.3.5 setCompRangeName()

```
void AisCompRange::setCompRangeName (
            const QString & compRangeName )
```

set a name for the compensation range for reference purposes.

Parameters

compRangeName	the name to set for the compensation range.

15.2.3.6 setStabilityFactor()

```
void AisCompRange::setStabilityFactor (
            uint8_t factor )
```

set a value for the stability factor.

Usually, the device's default factor value is optimal for running experiments. You may still increase the factor within the range 0-10 as you run experiments with more oscillations to see what best fits.

Parameters

factor	the stability-factor value (0-10)

The documentation for this class was generated from the following file:

- AisCompRange.h

15.3 AisConstantCurrentElement Class Reference

an experiment that simulates a constant current flow with more advance options for stopping the experiment.

```
#include <AisConstantCurrentElement.h>
```

Inherits AisAbstractElement.

Public Member Functions

- AisConstantCurrentElement (double current, double samplingInterval, double duration)

  the constant current element constructor.
- **AisConstantCurrentElement** (const AisConstantCurrentElement &)

  copy constructor for the AisConstantCurrentElement object.
- AisConstantCurrentElement & **operator=** (const AisConstantCurrentElement &)

  overload equal to operator for the AisConstantCurrentElement object.
- QString getName () const override

  get the name of the element.
- QStringList getCategory () const override

  get a list of applicable categories of the element.
- double getCurrent () const

  get the value set for the current.

- void setCurrent (double current)

    set the value for the current.
- double getSamplingInterval () const

    get how frequently we are sampling the data.
- void setSamplingInterval (double samplingInterval)

    set how frequently we are sampling the data.
- double getMinSamplingVoltageDifference () const

    get the minimum sampling voltage difference for reporting the data.
- void setMinSamplingVoltageDifference (double minVoltageDifference)

    set a minimum sampling voltage difference for reporting the voltage.
- double getMaxVoltage () const

    get the value set for the maximum voltage. The experiment will end when it reaches this value.
- void setMaxVoltage (double maxVoltage)

    set a maximum voltage to stop the experiment.
- double getMinVoltage () const

    get the value set minimum for the voltage in volts.
- void setMinVoltage (double minVoltage)

    set a minimum voltage to stop the experiment.
- double getMaxDuration () const

    get the maximum duration set for the experiment. The experiment will end when the duration of the experiment reaches this value.
- void setMaxDuration (double maxDuration)

    set the maximum duration for the experiment.
- double getMaxCapacity () const

    get the value set for the maximum capacity / cumulative charge.
- void setMaxCapacity (double maxCapacity)

    set the value for the maximum capacity / cumulative charge in Coulomb.
- bool isAutoRange () const

    tells whether the current range is set to auto-select or not.
- void setAutoRange ()

    set to auto-select the current range.
- double getApproxMaxCurrent () const

    get the value set for the expected maximum current.
- void setApproxMaxCurrent (double approxMaxCurrent)

    set maximum current expected, for manual current range selection.
- bool isAutoVoltageRange () const

    tells whether the voltage range is set to auto-select or not.
- void setAutoVoltageRange ()

    set to auto-select the voltage range.
- double getApproxMaxVoltage () const

    get the value set for the expected maximum voltage.
- void setApproxMaxVoltage (double approxMaxVoltage)

    set maximum voltage expected, for manual voltage range selection.

15.3.1 Detailed Description

an experiment that simulates a constant current flow with more advance options for stopping the experiment.

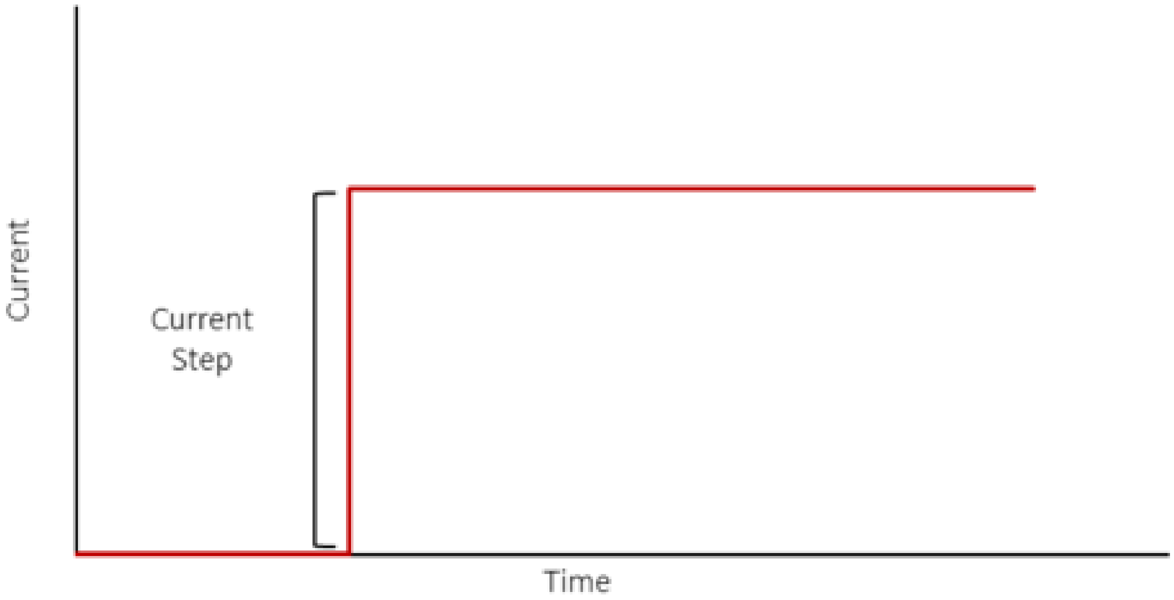


Figure 15.1 ConstantCurrent

15.3.2 Constructor & Destructor Documentation

15.3.2.1 AisConstantCurrentElement()

```
AisConstantCurrentElement::AisConstantCurrentElement (
            double current,
            double samplingInterval,
            double duration )  [explicit]
```

the constant current element constructor.

Parameters

current	the value for the current in Amps.
samplingInterval	the data sampling interval value in seconds.
duration	the maximum duration for the experiment in seconds.

15.3.3 Member Function Documentation

15.3.3.1 getApproxMaxCurrent()

```
double AisConstantCurrentElement::getApproxMaxCurrent ( ) const
```

get the value set for the expected maximum current.

Returns

the value set for the expected maximum current in Amps.

Note

if nothing was manually set, the device will auto-select the current range and the return value will be positive infinity.

15.3.3.2 getApproxMaxVoltage()

```
double AisConstantCurrentElement::getApproxMaxVoltage ( ) const
```

get the value set for the expected maximum voltage.

Returns

the value set for the expected maximum Voltage in volt.

Note

if nothing was manually set, the device will auto-select the voltage range and the return value will be positive infinity.

15.3.3.3 getCategory()

```
QStringList AisConstantCurrentElement::getCategory ( ) const  [override]
```

get a list of applicable categories of the element.

Returns

A list of applicable categories: ("Galvanostatic Control", "Basic Experiments").

15.3.3.4 getCurrent()

```
double AisConstantCurrentElement::getCurrent ( ) const
```

get the value set for the current.

Returns

the value for the current in Amps.

15.3.3.5 getMaxCapacity()

```
double AisConstantCurrentElement::getMaxCapacity ( ) const
```

get the value set for the maximum capacity / cumulative charge.

Returns

the value set for the maximum capacity in Coulomb.

Note

this is an optional parameter. If no value has been set, the default value is positive infinity.

15.3.3.6 getMaxDuration()

```
double AisConstantCurrentElement::getMaxDuration ( ) const
```

get the maximum duration set for the experiment. The experiment will end when the duration of the experiment reaches this value.

Returns

the maximum duration for the experiment in seconds.

15.3.3.7 getMaxVoltage()

```
double AisConstantCurrentElement::getMaxVoltage ( ) const
```

get the value set for the maximum voltage. The experiment will end when it reaches this value.

Returns

the value set for the maximum voltage.

Note

this is an optional parameter. If no value has been set, the default value is positive infinity

15.3.3.8 getMinSamplingVoltageDifference()

```
double AisConstantCurrentElement::getMinSamplingVoltageDifference ( ) const
```

get the minimum sampling voltage difference for reporting the data.

get the value set for the minimum sampling voltage difference.

Returns

the value set for the minimum sampling voltage difference.

See also

setMinSamplingVoltageDifference

Note

this is an optional parameter. If no value has been set, the default value is negative infinity.

15.3.3.9 getMinVoltage()

```
double AisConstantCurrentElement::getMinVoltage ( ) const
```

get the value set minimum for the voltage in volts.

Returns

the value set for the minimum voltage in volts.

Note

this is an optional parameter. If no value has been set, the default value is negative infinity

15.3.3.10 getName()

```
QString AisConstantCurrentElement::getName ( ) const  [override]
```

get the name of the element.

Returns

The name of the element: "Constant Current, Advanced".

15.3.3.11 getSamplingInterval()

```
double AisConstantCurrentElement::getSamplingInterval ( ) const
```

get how frequently we are sampling the data.

Returns

the data sampling interval value in seconds.

15.3.3.12 isAutoRange()

```
bool AisConstantCurrentElement::isAutoRange ( ) const
```

tells whether the current range is set to auto-select or not.

Returns

true if the current range is set to auto-select and false if a range has been selected.

15.3.3.13 isAutoVoltageRange()

```
bool AisConstantCurrentElement::isAutoVoltageRange ( ) const
```

tells whether the voltage range is set to auto-select or not.

Returns

true if the voltage range is set to auto-select and false if a range has been selected.

15.3.3.14 setApproxMaxCurrent()

```
void AisConstantCurrentElement::setApproxMaxCurrent (
             double approxMaxCurrent )
```

set maximum current expected, for manual current range selection.

The is an **optional** parameter. If nothing is set, the device will auto-select the current range.

Parameters

approxMaxCurrent	the value for the maximum current expected in Amps.

15.3.3.15 setApproxMaxVoltage()

```
void AisConstantCurrentElement::setApproxMaxVoltage (
            double approxMaxVoltage )
```

set maximum voltage expected, for manual voltage range selection.

The is an **optional** parameter. If nothing is set, the device will auto-select the voltage range.

Parameters

approxMaxVoltage	the value for the maximum current expected in V.

15.3.3.16 setAutoRange()

```
void AisConstantCurrentElement::setAutoRange ( )
```

set to auto-select the current range.

This option is set by default. There is no need to call this function to auto-select if the range was not manually set.

15.3.3.17 setAutoVoltageRange()

```
void AisConstantCurrentElement::setAutoVoltageRange ( )
```

set to auto-select the voltage range.

This option is set by default. There is no need to call this function to auto-select if the range was not manually set.

15.3.3.18 setCurrent()

```
void AisConstantCurrentElement::setCurrent (
            double current )
```

set the value for the current.

Parameters

current	the value for the current in Amps.

15.3.3.19 setMaxCapacity()

```
void AisConstantCurrentElement::setMaxCapacity (
            double maxCapacity )
```

set the value for the maximum capacity / cumulative charge in Coulomb.

The is an **optional** condition. If nothing is set, then the experiment will not stop based on an upper-limit cumulative charge value. If a maximum capacity is set, the experiment will continue to run as long as the cumulative charge is below that value.

Parameters

maxCapacity	the value to set for the cell maximum capacity.

15.3.3.20 setMaxDuration()

```
void AisConstantCurrentElement::setMaxDuration (
            double maxDuration )
```

set the maximum duration for the experiment.

The experiment will continue to run as long as the time passed is less than the value to set.

Parameters

maxDuration	the maximum duration for the experiment in seconds.

15.3.3.21 setMaxVoltage()

```
void AisConstantCurrentElement::setMaxVoltage (
            double maxVoltage )
```

set a maximum voltage to stop the experiment.

The is an **optional** condition. If nothing is set, then the experiment will not stop based on an upper-limit voltage value. If a maximum voltage is set, the experiment will continue to run as long as the measured voltage is below that value.

Parameters

maxVoltage	the maximum voltage value in volts at which the experiment will stop.

15.3.3.22 setMinSamplingVoltageDifference()

```
void AisConstantCurrentElement::setMinSamplingVoltageDifference (
            double minVoltageDifference )
```

set a minimum sampling voltage difference for reporting the voltage.

The is an **optional** condition. If nothing is set, then the experiment will report the data at time sampling interval. When this is set, then the voltage is reported when there is a voltage difference of at least the given minimum sampling voltage difference. So, when one voltage data point is reported (at the minimum possible time sampling interval), the next data point is not reported unless the difference between the two voltage data points exceeds this given minimum sampling voltage difference value.

Note

when this is set, this overrides the set value for the sampling interval.

Parameters

minVoltageDifference	the minimum sampling voltage difference value in volts.

15.3.3.23 setMinVoltage()

```
void AisConstantCurrentElement::setMinVoltage (
            double minVoltage )
```

set a minimum voltage to stop the experiment.

The is an **optional** condition. If nothing is set, then the experiment will not stop based on an lower-limit voltage value. If a maximum voltage is set, the experiment will continue to run as long as the measured voltage is above that value.

Parameters

minVoltage	the minimum voltage value in volts at which the experiment will stop.

15.3.3.24 setSamplingInterval()

```
void AisConstantCurrentElement::setSamplingInterval (
            double samplingInterval )
```

set how frequently we are sampling the data.

Parameters

samplingInterval	the data sampling interval value in seconds.

The documentation for this class was generated from the following file:

- AisConstantCurrentElement.h

15.4 AisConstantPotElement Class Reference

an experiment that simulates a constant applied voltage.

```
#include <AisConstantPotElement.h>
```

Inherits AisAbstractElement.

Public Member Functions

- AisConstantPotElement (double voltage, double samplingInterval, double duration)

  the constant potential element constructor.
- **AisConstantPotElement** (const AisConstantPotElement &)

  copy constructor for the AisConstantPotElement object.
- AisConstantPotElement & **operator=** (const AisConstantPotElement &)

  overload equal to operator for the AisConstantPotElement object.
- QString getName () const override

  get the name of the element.
- QStringList getCategory () const override

  get a list of applicable categories of the element.
- double getPotential () const

  get the value set for the potential in volts.
- void setPotential (double potential)

  set the value for the potential in volts.
- bool isVoltageVsOCP () const

  tells whether the specified voltage is set against the open-circuit voltage or the reference terminal.
- void setVoltageVsOCP (bool vsOCP)

  set whether to reference the specified voltage against the open-circuit voltage or the reference terminal.
- double getSamplingInterval () const

  get how frequently we are sampling the data.
- void setSamplingInterval (double samplingInterval)

  set how frequently we are sampling the data.
- double getMaxDuration () const

  get the maximum duration set for the experiment. The experiment will end when the duration of the experiment reaches this value.
- void setMaxDuration (double maxDuration)

  set the maximum duration for the experiment.
- double getMaxCurrent () const

  get the maximum value set for the absolute current in Amps. The experiment will end when the absolute current reaches this value.
- void setMaxCurrent (double maxCurrent)

  set the maximum value for the absolute current in Amps.
- double getMinCurrent () const

get the minimum value set for the absolute current in Amps. The experiment will end when the absolute current falls down to this value.

- void setMinCurrent (double minCurrent)

  set the minimum value for the absolute current in Amps.
- double getMaxCapacity () const

  get the value set for the maximum capacity / cumulative charge.
- void setMaxCapacity (double maxCapacity)

  set the value for the maximum capacity / cumulative charge in Coulomb.
- double getMindIdt () const

  get the value set for the minimum current rate of change with respect to time (minimum di/dt).
- void setMindIdt (double mindIdt)

  set the minimum value for the current rate of change with respect to time (minimum di/dt).
- bool isAutoRange () const

  tells whether the current range is set to auto-select or not.
- void setAutoRange ()

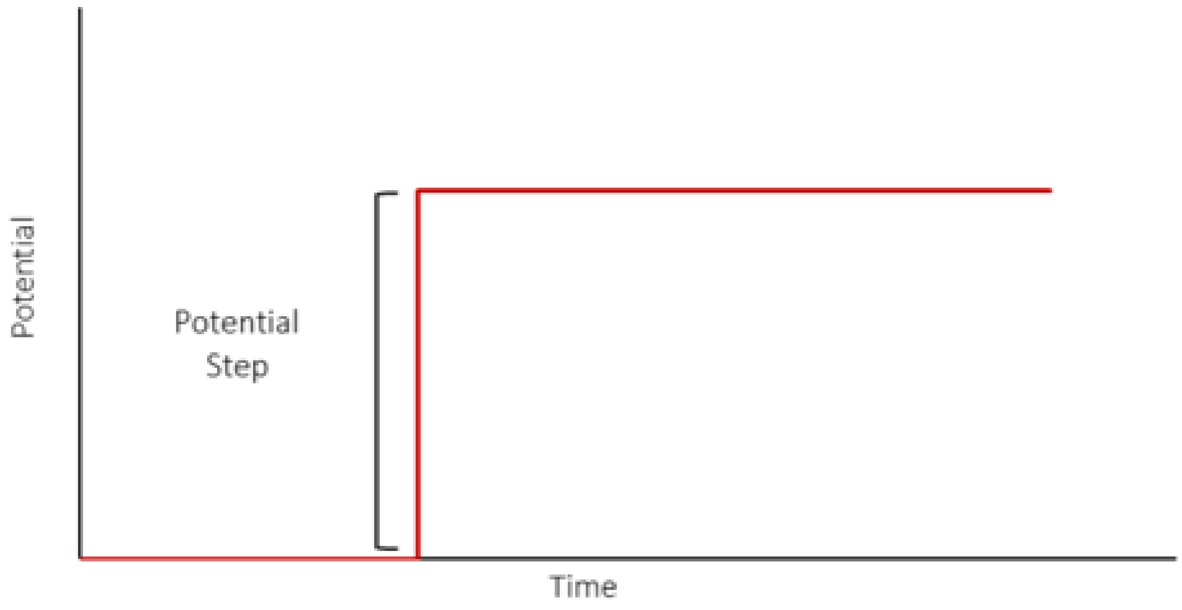
  set to auto-select the current range.
- double getApproxMaxCurrent () const

  get the value set for the expected maximum current.
- void setApproxMaxCurrent (double approxMaxCurrent)

  set maximum current expected, for manual current range selection.

15.4.1 Detailed Description

an experiment that simulates a constant applied voltage.



15.4.2 Constructor & Destructor Documentation

15.4.2.1 AisConstantPotElement()

```
AisConstantPotElement::AisConstantPotElement (
             double voltage,
             double samplingInterval,
             double duration )  [explicit]
```

the constant potential element constructor.

Parameters

voltage	the value set for the voltage/potential in volts.
samplingInterval	the data sampling interval value in seconds.
duration	the maximum duration for the experiment in seconds.

15.4.3 Member Function Documentation

15.4.3.1 getApproxMaxCurrent()

```
double AisConstantPotElement::getApproxMaxCurrent ( ) const
```

get the value set for the expected maximum current.

Returns

the value set for the expected maximum current in Amps.

Note

if nothing was manually set, the device will auto-select the current range and the return value will be positive infinity.

15.4.3.2 getCategory()

```
QStringList AisConstantPotElement::getCategory ( ) const  [override]
```

get a list of applicable categories of the element.

Returns

A list of applicable categories: ("Potentiostatic Control", "Basic Experiments")

15.4.3.3 getMaxCapacity()

```
double AisConstantPotElement::getMaxCapacity ( ) const
```

get the value set for the maximum capacity / cumulative charge.

Returns

the value set for the maximum capacity in Coulomb.

Note

this is an optional parameter. If no value has been set, the default value is positive infinity.

15.4.3.4 getMaxCurrent()

```
double AisConstantPotElement::getMaxCurrent ( ) const
```

get the maximum value set for the absolute current in Amps. The experiment will end when the absolute current reaches this value.

Returns

the maximum current value in Amps.

Note

this is an optional parameter. If no value has been set, the default value is positive infinity.

15.4.3.5 getMaxDuration()

```
double AisConstantPotElement::getMaxDuration ( ) const
```

get the maximum duration set for the experiment. The experiment will end when the duration of the experiment reaches this value.

Returns

the maximum duration for the experiment in seconds.

15.4.3.6 getMinCurrent()

```
double AisConstantPotElement::getMinCurrent ( ) const
```

get the minimum value set for the absolute current in Amps. The experiment will end when the absolute current falls down to this value.

Returns

the minimum current value in Amps.

Note

this is an optional parameter. If no value has been set, the default value is zero.

15.4.3.7 getMindIdt()

```
double AisConstantPotElement::getMindIdt ( ) const
```

get the value set for the minimum current rate of change with respect to time (minimum di/dt).

Returns

the value set for the minimum current rate of change with respect to time (minimum di/dt).

Note

this is an optional parameter. If no value has been set, the default value is zero.

15.4.3.8 getName()

```
QString AisConstantPotElement::getName ( ) const [override]
```

get the name of the element.

Returns

The name of the element: "Constant Potential, Advanced".

15.4.3.9 getPotential()

```
double AisConstantPotElement::getPotential ( ) const
```

get the value set for the potential in volts.

Returns

the value set for the potential in volts.

15.4.3.10 getSamplingInterval()

```
double AisConstantPotElement::getSamplingInterval ( ) const
```

get how frequently we are sampling the data.

Returns

the data sampling interval value in seconds.

15.4.3.11 isAutoRange()

```
bool AisConstantPotElement::isAutoRange ( ) const
```

tells whether the current range is set to auto-select or not.

Returns

true if the current range is set to auto-select and false if a rage has been selected.

15.4.3.12 isVoltageVsOCP()

```
bool AisConstantPotElement::isVoltageVsOCP ( ) const
```

tells whether the specified voltage is set against the open-circuit voltage or the reference terminal.

Returns

true if the specified voltage is set against the open-circuit voltage and false if it is set against the reference terminal.

See also

setVsOcp

15.4.3.13 setApproxMaxCurrent()

```
void AisConstantPotElement::setApproxMaxCurrent (
             double approxMaxCurrent )
```

set maximum current expected, for manual current range selection.

The is an **optional** parameter. If nothing is set, the device will auto-select the current range.

Parameters

approxMaxCurrent	the value for the maximum current expected in Amps.

15.4.3.14 setAutoRange()

```
void AisConstantPotElement::setAutoRange ( )
```

set to auto-select the current range.

This option is set by default. There is no need to call this function to auto-select if the range was not manually set.

15.4.3.15 setMaxCapacity()

```
void AisConstantPotElement::setMaxCapacity (
            double maxCapacity )
```

set the value for the maximum capacity / cumulative charge in Coulomb.

The is an **optional** condition. If nothing is set, then the experiment will not stop based on an upper-limit cumulative charge value. If a maximum capacity is set, the experiment will continue to run as long as the cumulative charge is below that value.

Parameters

maxCapacity	the value to set for the cell maximum capacity.

15.4.3.16 setMaxCurrent()

```
void AisConstantPotElement::setMaxCurrent (
            double maxCurrent )
```

set the maximum value for the absolute current in Amps.

The is an **optional** condition. If nothing is set, then the experiment will not stop based on an upper-limit current value. If a maximum current is set, the experiment will continue to run as long as the measured current is below that value.

Parameters

maxCurrent	the maximum current value in Amps.

15.4.3.17 setMaxDuration()

```
void AisConstantPotElement::setMaxDuration (
            double maxDuration )
```

set the maximum duration for the experiment.

The is an **optional** condition. If nothing is set, then the experiment will not stop based on an duration. If a maximum duration is set, the experiment will continue to run as long as the passed time is less than that value.

Parameters

maxDuration	the maximum duration for the experiment in seconds.

15.4.3.18 setMinCurrent()

```
void AisConstantPotElement::setMinCurrent (
            double minCurrent )
```

set the minimum value for the absolute current in Amps.

The is an **optional** condition. If nothing is set, then the experiment will not stop based on an lower-limit current value. If a maximum current is set, the experiment will continue to run as long as the measured current is above that value.

Parameters

minCurrent	the value to set for the absolute minimum current.

15.4.3.19 setMindIdt()

```
void AisConstantPotElement::setMindIdt (
            double mindIdt )
```

set the minimum value for the current rate of change with respect to time (minimum di/dt).

The is an **optional** condition. If nothing is set, then the experiment will not stop based on an lower-limit rate of change value. If a minimum value is set, the experiment will continue to run as long as the rage of change is above that value.

Parameters

mindIdt	the minimum value for the current rate of change with respect to time (minimum di/dt).

15.4.3.20 setPotential()

```
void AisConstantPotElement::setPotential (
            double potential )
```

set the value for the potential in volts.

Parameters

potential	the value to set for the potential in volts.

15.4.3.21 setSamplingInterval()

```
void AisConstantPotElement::setSamplingInterval (
            double samplingInterval )
```

set how frequently we are sampling the data.

Parameters

samplingInterval	the data sampling interval value in seconds.

15.4.3.22 setVoltageVsOCP()

```
void AisConstantPotElement::setVoltageVsOCP (
            bool vsOCP )
```

set whether to reference the specified voltage against the open-circuit voltage or the reference terminal.

The reference terminal is for you to connect to any reference point you like. Connect it to the working electrode to reference ground.

Parameters

vsOCP	true to set the specified voltage to reference the open-circuit voltage and false to set against the reference terminal.

The documentation for this class was generated from the following file:

- AisConstantPotElement.h

15.5 AisConstantPowerElement Class Reference

This experiment simulates a constant power, charge or discharge".

```
#include <AisConstantPowerElement.h>
```

Inherits AisAbstractElement.

Public Member Functions

- AisConstantPowerElement (bool isCharge, double power, double duration, double smaplingInterval)

  the constant power element constructor
- **AisConstantPowerElement** (const AisConstantPowerElement &)

  copy constructor for the AisConstantPowerElement object.
- AisConstantPowerElement & **operator=** (const AisConstantPowerElement &)

  overload equal to operator for the AisConstantPowerElement object.
- QString getName () const override

  get the name of the element.
- QStringList getCategory () const override

  get a list of applicable categories of the element.
- bool isCharge () const

  tells whether the experiment is set to simulate charge or discharge.
- void setCharge (bool isCharge)

  set whether the experiment is to simulate charge or discharge.
- double getPower () const

  get the value set for the power.
- void setPower (double power)

  set the value for the power.
- double getSamplingInterval () const

  get how frequently we are sampling the data.
- void setSamplingInterval (double samplingInterval)

  set how frequently we are sampling the data.
- double getMaxVoltage () const

  get the value set for the maximum voltage. The experiment will end when it reaches this value.
- void setMaxVoltage (double maxVoltage)

  set a maximum voltage to stop the experiment.
- double getMinVoltage () const

  get the minimum value set for the voltage in volts. The experiment will end when it reaches down this value.
- void setMinVoltage (double minVoltage)

  set a minimum value for the voltage. The experiment will end when it reaches down this value.
- double getMaxDuration () const

  get the maximum duration set for the experiment. The experiment will end when the duration of the experiment reaches this value.
- void setMaxDuration (double maxDuration)

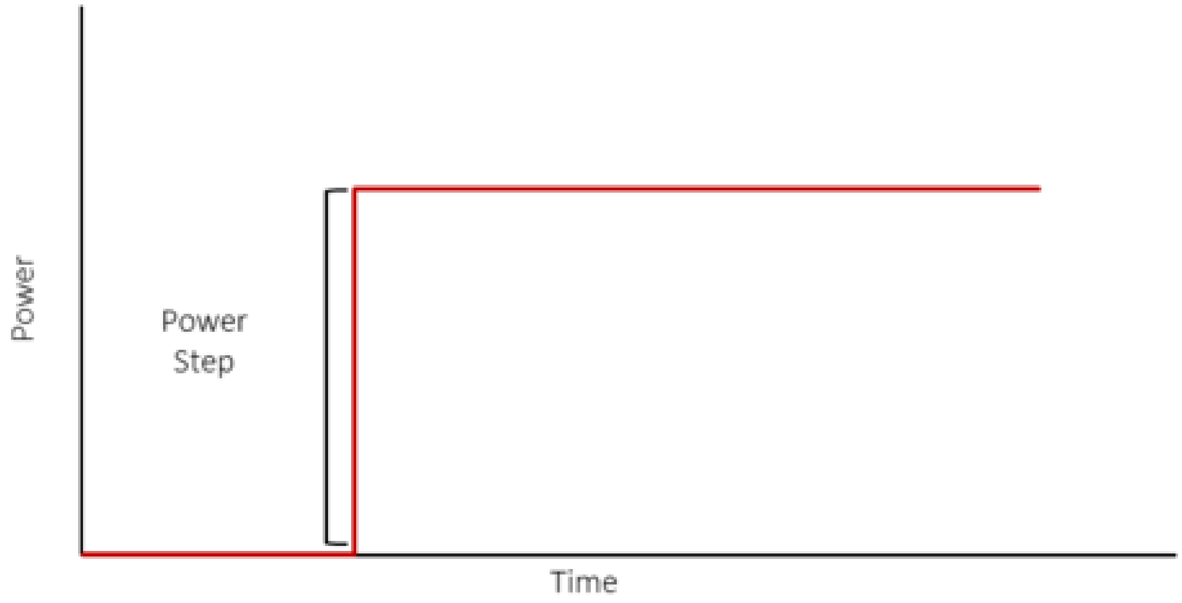
  set the maximum duration for the experiment.
- double getMaxCapacity () const

  get the value set for the maximum capacity / cumulative charge.
- void setMaxCapacity (double maxCapacity)

  set the value for the maximum capacity / cumulative charge in Coulomb.

15.5.1 Detailed Description

This experiment simulates a constant power, charge or discharge".



15.5.2 Constructor & Destructor Documentation

15.5.2.1 AisConstantPowerElement()

```
AisConstantPowerElement::AisConstantPowerElement (
             bool isCharge,
             double power,
             double duration,
             double smaplingInterval )  [explicit]
```

the constant power element constructor

Parameters

isCharge	true to set the experiment simulate charge and false to simulate discharge.
power	the value set for the power in watts.
duration	the maximum duration for the experiment in seconds.
smaplingInterval	the data sampling interval value in seconds.

15.5.3 Member Function Documentation

15.5.3.1 getCategory()

```
QStringList AisConstantPowerElement::getCategory ( ) const  [override]
```

get a list of applicable categories of the element.

Returns

A list of applicable categories: ("Energy Storage", "Charge/Discharge").

15.5.3.2 getMaxCapacity()

```
double AisConstantPowerElement::getMaxCapacity ( ) const
```

get the value set for the maximum capacity / cumulative charge.

Returns

the value set for the maximum capacity in Coulomb.

Note

this is an optional parameter. If no value has been set, the default value is positive infinity.

15.5.3.3 getMaxDuration()

```
double AisConstantPowerElement::getMaxDuration ( ) const
```

get the maximum duration set for the experiment. The experiment will end when the duration of the experiment reaches this value.

Returns

the maximum duration for the experiment in seconds.

15.5.3.4 getMaxVoltage()

```
double AisConstantPowerElement::getMaxVoltage ( ) const
```

get the value set for the maximum voltage. The experiment will end when it reaches this value.

Returns

the value set for the maximum voltage.

Note

this is an optional parameter. If no value has been set, the default value is positive infinity

15.5.3.5 getMinVoltage()

```
double AisConstantPowerElement::getMinVoltage ( ) const
```

get the minimum value set for the voltage in volts. The experiment will end when it reaches down this value.

Returns

the minimum value set for the voltage in volts.

Note

this is an optional parameter. If no value has been set, the default value is negative infinity

15.5.3.6 getName()

```
QString AisConstantPowerElement::getName ( ) const  [override]
```

get the name of the element.

Returns

The name of the element: "Constant Power Charge/Discharge".

15.5.3.7 getPower()

```
double AisConstantPowerElement::getPower ( ) const
```

get the value set for the power.

Returns

the value set for the power in watts.

15.5.3.8 getSamplingInterval()

```
double AisConstantPowerElement::getSamplingInterval ( ) const
```

get how frequently we are sampling the data.

Returns

the data sampling interval value in seconds.

15.5.3.9 isCharge()

```
bool AisConstantPowerElement::isCharge ( ) const
```

tells whether the experiment is set to simulate charge or discharge.

Returns

true if the experiment is set to simulate charge and false if it is set to simulate discharge.

15.5.3.10 setCharge()

```
void AisConstantPowerElement::setCharge (
             bool isCharge )
```

set whether the experiment is to simulate charge or discharge.

Parameters

isCharge	if the given argument is true, the experiment will simulate charge and discharge if given false.

15.5.3.11 setMaxCapacity()

```
void AisConstantPowerElement::setMaxCapacity (
            double maxCapacity )
```

set the value for the maximum capacity / cumulative charge in Coulomb.

The is an **optional** condition. If nothing is set, then the experiment will not stop based on an upper-limit cumulative charge value. If a maximum capacity is set, the experiment will continue to run as long as the cumulative charge is below that value.

Parameters

maxCapacity	the value to set for the cell maximum capacity.

15.5.3.12 setMaxDuration()

```
void AisConstantPowerElement::setMaxDuration (
            double maxDuration )
```

set the maximum duration for the experiment.

The experiment will continue to run as long as the passed time is less than that the set duration value.

Parameters

maxDuration	the maximum duration for the experiment in seconds.

15.5.3.13 setMaxVoltage()

```
void AisConstantPowerElement::setMaxVoltage (
            double maxVoltage )
```

set a maximum voltage to stop the experiment.

The is an **optional** condition. If nothing is set, then the experiment will not stop based on an upper-limit voltage value. If a maximum voltage is set, the experiment will continue to run as long as the measured voltage is below that value.

Parameters

maxVoltage	the maximum voltage value in volts at which the experiment will stop.

15.5.3.14 setMinVoltage()

```
void AisConstantPowerElement::setMinVoltage (
            double minVoltage )
```

set a minimum value for the voltage. The experiment will end when it reaches down this value.

Parameters

minVoltage	the value for the voltage in volts.

Note

this is an optional parameter. If no value has been set, the default value is negative infinity.

15.5.3.15 setPower()

```
void AisConstantPowerElement::setPower (
            double power )
```

set the value for the power.

Parameters

power	the value set for the power in watts.

15.5.3.16 setSamplingInterval()

```
void AisConstantPowerElement::setSamplingInterval (
            double samplingInterval )
```

set how frequently we are sampling the data.

Parameters

samplingInterval	the data sampling interval value in seconds.

The documentation for this class was generated from the following file:

- AisConstantPowerElement.h

15.6 AisConstantResistanceElement Class Reference

This element/experiment simulates a constant resistance load.

```
#include <AisConstantResistanceElement.h>
```

Inherits AisAbstractElement.

Public Member Functions

- AisConstantResistanceElement (double resistance, double duration, double samplingInterval)

  the constant resistance element constructor.
- **AisConstantResistanceElement** (const AisConstantResistanceElement &)

  copy constructor for the AisConstantResistanceElement object.
- AisConstantResistanceElement & **operator=** (const AisConstantResistanceElement &)

  overload equal to operator for the AisConstantResistanceElement object.
- QString getName () const override

  get the name of the element.
- QStringList getCategory () const override

  get a list of applicable categories of the element.
- double getResistance () const

  get the value set for the resistance as a load.
- void setResistance (double resistance)

  set the value for the resistance as a load
- double getSamplingInterval () const

  get how frequently we are sampling the data.
- void setSamplingInterval (double samplingInterval)

  set how frequently we are sampling the data.
- double getMinVoltage () const

  get the value set minimum for the voltage in volts.
- void setMinVoltage (double minVoltage)

  set a minimum voltage to stop the experiment.
- double getMaxDuration () const

  get the maximum duration set for the experiment. The experiment will end when the duration of the experiment reaches this value.
- void setMaxDuration (double maxDuration)

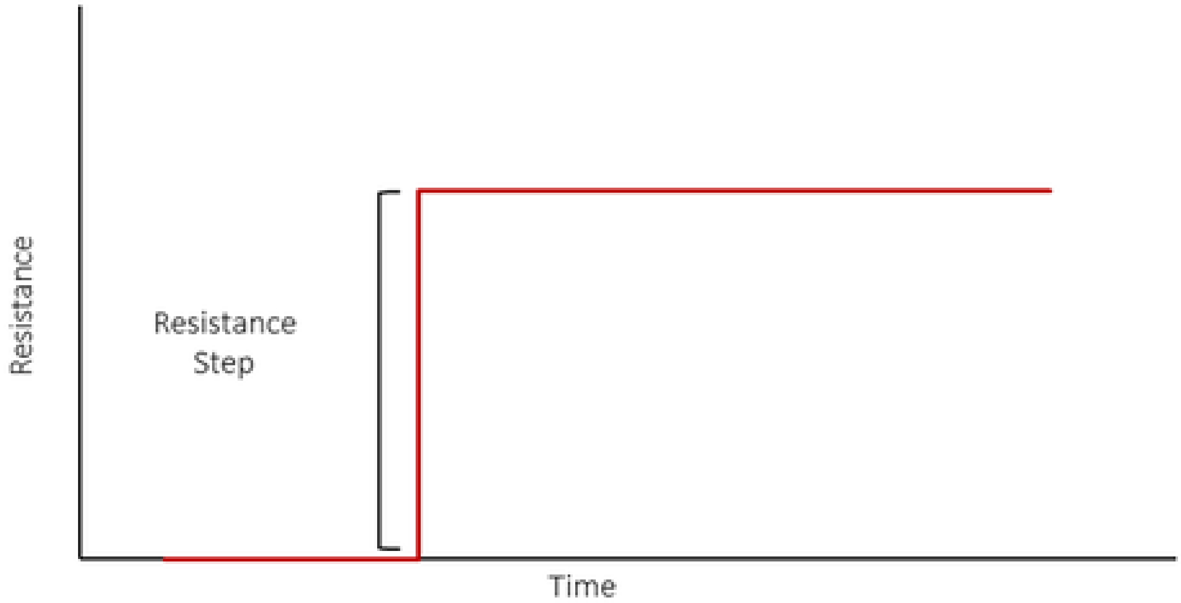
  set the maximum duration for the experiment.
- double getMaxCapacity () const

  get the value set for the maximum capacity / cumulative charge.
- void setMaxCapacity (double maxCapacity)

  set the value for the maximum capacity / cumulative charge in Coulomb.

15.6.1 Detailed Description

This element/experiment simulates a constant resistance load.



15.6.2 Constructor & Destructor Documentation

15.6.2.1 AisConstantResistanceElement()

```
AisConstantResistanceElement::AisConstantResistanceElement (
                double resistance,
                double duration,
                double samplingInterval )  [explicit]
```

the constant resistance element constructor.

Parameters

resistance	the value in ohm of the load resistance
duration	the maximum duration for the experiment in seconds.
samplingInterval	the data sampling interval value in seconds.

15.6.3 Member Function Documentation

15.6.3.1 getCategory()

```
QStringList AisConstantResistanceElement::getCategory ( ) const  [override]
```

get a list of applicable categories of the element.

Returns

A list of applicable categories: ("Energy Storage", "Charge/Discharge").

15.6.3.2 getMaxCapacity()

```
double AisConstantResistanceElement::getMaxCapacity ( ) const
```

get the value set for the maximum capacity / cumulative charge.

Returns

the value set for the maximum capacity in Coulomb.

Note

this is an optional parameter. If no value has been set, the default value is positive infinity.

15.6.3.3 getMaxDuration()

```
double AisConstantResistanceElement::getMaxDuration ( ) const
```

get the maximum duration set for the experiment. The experiment will end when the duration of the experiment reaches this value.

Returns

the maximum duration for the experiment in seconds.

15.6.3.4 getMinVoltage()

```
double AisConstantResistanceElement::getMinVoltage ( ) const
```

get the value set minimum for the voltage in volts.

Returns

the value set for the minimum voltage in volts.

Note

this is an optional parameter. If no value has been set, the default value is negative infinity

15.6.3.5 getName()

```
QString AisConstantResistanceElement::getName ( ) const  [override]
```

get the name of the element.

Returns

The name of the element: "Constant Resistance".

15.6.3.6 getResistance()

```
double AisConstantResistanceElement::getResistance ( ) const
```

get the value set for the resistance as a load.

Returns

the value in ohm of the load resistance.

15.6.3.7 getSamplingInterval()

```
double AisConstantResistanceElement::getSamplingInterval ( ) const
```

get how frequently we are sampling the data.

Returns

the data sampling interval value in seconds.

15.6.3.8 setMaxCapacity()

```
void AisConstantResistanceElement::setMaxCapacity (
             double maxCapacity )
```

set the value for the maximum capacity / cumulative charge in Coulomb.

The is an **optional** condition. If nothing is set, then the experiment will not stop based on an upper-limit cumulative charge value. If a maximum capacity is set, the experiment will continue to run as long as the cumulative charge is below that value.

Parameters

maxCapacity	the value to set for the cell maximum capacity.

15.6.3.9 setMaxDuration()

```
void AisConstantResistanceElement::setMaxDuration (
            double maxDuration )
```

set the maximum duration for the experiment.

The experiment will continue to run as long as the passed time is less than that the set duration value.

Parameters

maxDuration	the maximum duration for the experiment in seconds.

15.6.3.10 setMinVoltage()

```
void AisConstantResistanceElement::setMinVoltage (
            double minVoltage )
```

set a minimum voltage to stop the experiment.

The is an **optional** condition. If nothing is set, then the experiment will not stop based on an lower-limit voltage value. If a maximum voltage is set, the experiment will continue to run as long as the measured voltage is above that value.

Parameters

minVoltage	the minimum voltage value in volts at which the experiment will stop.

15.6.3.11 setResistance()

```
void AisConstantResistanceElement::setResistance (
            double resistance )
```

set the value for the resistance as a load

Parameters

resistance	the value in ohm of the load resistance.

15.6.3.12 setSamplingInterval()

```
void AisConstantResistanceElement::setSamplingInterval (
            double samplingInterval )
```

set how frequently we are sampling the data.

Parameters

samplingInterval	the data sampling interval value in seconds.

The documentation for this class was generated from the following file:

- AisConstantResistanceElement.h

15.7 AisCyclicVoltammetryElement Class Reference

This experiment sweeps the potential of the working electrode back and forth between the **first voltage-limit** and the **second voltage-limit** at a constant **scan rate (dE/dt)** for a specified number of **cycles**.

```
#include <AisCyclicVoltammetryElement.h>
```

Inherits AisAbstractElement.

Public Member Functions

- AisCyclicVoltammetryElement (double startVoltage, double firstVoltageLimit, double secondVoltageLimit, double endVoltage, double dEdt, double samplingInterval)

  constructor of the cyclic voltammetry element.
- **AisCyclicVoltammetryElement** (const AisCyclicVoltammetryElement &)

  copy constructor for the AisCyclicVoltammetryElement object.
- AisCyclicVoltammetryElement & **operator=** (const AisCyclicVoltammetryElement &)

  overload equal to operator for the AisCyclicVoltammetryElement object.
- QString getName () const override

  get the name of the element.
- QStringList getCategory () const override

  get a list of applicable categories of the element.
- double getStartVoltage () const

  get the value set for the start voltage
- void setStartVoltage (double startVoltage)

  set the value for the start voltage.
- bool isStartVoltageVsOCP () const

  tells whether the start voltage is set with respect to the open circuit voltage or not.
- void setStartVoltageVsOCP (bool startVoltageVsOCP)

  set whether to reference the start voltage against the open-circuit voltage or the reference terminal.
- double getFirstVoltageLimit () const

get the value set for the first voltage-limit.

- void setFirstVoltageLimit (double v1)

  set the first voltage-limit
- bool isFirstVoltageLimitVsOCP () const

  tells whether the first voltage-limit is set with respect to the open circuit voltage or not.
- void setFirstVoltageLimitVsOCP (bool firstVoltageLimitVsOCP)

  set whether to reference the first voltage-limit against the open-circuit voltage or not.
- double getSecondVoltageLimit () const

  get the value set for the second voltage-limit
- void setSecondVoltageLimit (double v2)

  set the second voltage-limit
- bool isSecondVoltageLimitVsOCP () const

  tells whether the second voltage-limit is set with respect to the open circuit voltage or not.
- void setSecondVoltageLimitVsOCP (bool secondVoltageLimitVsOCP)

  set whether to reference the second voltage-limit against the open-circuit voltage or not.
- double getNumberOfCycles ()

  get the value set for the number of cycles
- void setNumberOfCycles (int cycles)

  set the number of cycles to oscillate between the first voltage-limit and the second voltage-limit.
- double getEndVoltage () const

  get the value set for the ending potential value.
- void setEndVoltage (double endVoltage)

  set the ending potential value.
- bool isEndVoltageVsOCP () const

  tells whether the end voltage is set with respect to the open circuit voltage or the reference terminal.
- void setEndVoltageVsOCP (bool endVoltageVsOCP)

  set whether to reference the end voltage against the open-circuit voltage or the reference terminal.
- double getdEdt () const

  get the value set for the constant scan rate dE/dt.
- void setdEdt (double dEdt)

  set the value for the constant scan rate dE/dt.
- double getSamplingInterval () const

  get how frequently we are sampling the data.
- void setSamplingInterval (double sampInterval)

  set how frequently we are sampling the data.
- bool isAutoRange () const

  tells whether the current range is set to auto-select or not.
- void setAutoRange ()

  set to auto-select the current range.
- double getApproxMaxCurrent () const

  get the value set for the expected maximum current.
- void setApproxMaxCurrent (double approxMaxCurrent)

  set maximum current expected, for manual current range selection.

15.7.1 Detailed Description

This experiment sweeps the potential of the working electrode back and forth between the **first voltage-limit** and the **second voltage-limit** at a constant **scan rate (dE/dt)** for a specified number of **cycles**.

The scan will always start from the **start voltage** towards the **first voltage-limit**. The experiment will continue to cycle between the **first voltage-limit** and the **second voltage-limit** according to the number of cycles. The cycling scheme is as follow: **start voltage** → [**first voltage-limit** → **first voltage-limit**]n → **Ending potential**, where “n” is number of cycles.

15.7.2 Constructor & Destructor Documentation

15.7.2.1 AisCyclicVoltammetryElement()

```
AisCyclicVoltammetryElement::AisCyclicVoltammetryElement (
             double startVoltage,
             double firstVoltageLimit,
             double secondVoltageLimit,
             double endVoltage,
             double dEdt,
             double samplingInterval )  [explicit]
```

constructor of the cyclic voltammetry element.

Parameters

startVoltage	the value of the start voltage in volts
firstVoltageLimit	the value of the first voltage-limit in volts
secondVoltageLimit	the value of the second voltage-limit in volts
endVoltage	the value of the end voltage in volts
dEdt	the constant scan rate dE/dt in V/s.
samplingInterval	the data sampling interval value in seconds.

15.7.3 Member Function Documentation

15.7.3.1 getApproxMaxCurrent()

```
double AisCyclicVoltammetryElement::getApproxMaxCurrent ( ) const
```

get the value set for the expected maximum current.

Returns

the value set for the expected maximum current in Amps.

Note

if nothing was manually set, the device will auto-select the current range and the return value will be positive infinity.

15.7.3.2 getCategory()

```
QStringList AisCyclicVoltammetryElement::getCategory ( ) const  [override]
```

get a list of applicable categories of the element.

Returns

A list of applicable categories: ("Potentiostatic Control", "Basic Experiments").

15.7.3.3 getdEdt()

```
double AisCyclicVoltammetryElement::getdEdt ( ) const
```

get the value set for the constant scan rate dE/dt.

Returns

the value set for the constant scan rate dE/dt in V/s.

15.7.3.4 getEndVoltage()

```
double AisCyclicVoltammetryElement::getEndVoltage ( ) const
```

get the value set for the ending potential value.

This is the value of the voltage at which the experiment will stop. After the last cycle, the experiment will do one last sweep towards this value.

Returns

the value set for the ending voltage in volts.

15.7.3.5 getFirstVoltageLimit()

```
double AisCyclicVoltammetryElement::getFirstVoltageLimit ( ) const
```

get the value set for the first voltage-limit.

After the starting voltage, the scan will go to the first voltage-limit. This could result in either upward scan first if the first voltage-limit is higher than the start voltage or downward scan first if the first voltage-limit is lower than the start voltage.

Returns

the first voltage-limit value in volts.

15.7.3.6 getName()

```
QString AisCyclicVoltammetryElement::getName ( ) const  [override]
```

get the name of the element.

Returns

The name of the element: "Cyclic Voltammetry".

15.7.3.7 getNumberOfCycles()

```
double AisCyclicVoltammetryElement::getNumberOfCycles ( )
```

get the value set for the number of cycles

Returns

the number of cycles set.

15.7.3.8 getSamplingInterval()

```
double AisCyclicVoltammetryElement::getSamplingInterval ( ) const
```

get how frequently we are sampling the data.

Returns

the data sampling interval value in seconds.

15.7.3.9 getSecondVoltageLimit()

```
double AisCyclicVoltammetryElement::getSecondVoltageLimit ( ) const
```

get the value set for the second voltage-limit

After starting from the start-voltage and reaching the first voltage-limit, the scan will go to the second voltage limit. The scan will continue to oscillate between the first and second voltage-limits according to the number of cycles.

Returns

the second voltage-limit value in volts.

15.7.3.10 getStartVoltage()

```
double AisCyclicVoltammetryElement::getStartVoltage ( ) const
```

get the value set for the start voltage

Returns

the value of the start voltage in volts

15.7.3.11 isAutoRange()

```
bool AisCyclicVoltammetryElement::isAutoRange ( ) const
```

tells whether the current range is set to auto-select or not.

Returns

true if the current range is set to auto-select and false if a rage has been selected.

15.7.3.12 isEndVoltageVsOCP()

```
bool AisCyclicVoltammetryElement::isEndVoltageVsOCP ( ) const
```

tells whether the end voltage is set with respect to the open circuit voltage or the reference terminal.

Returns

true if the end voltage is set with respect to the open-circuit voltage and false if set against the reference terminal.

Note

if no value was set, the default is false

15.7.3.13 isFirstVoltageLimitVsOCP()

```
bool AisCyclicVoltammetryElement::isFirstVoltageLimitVsOCP ( ) const
```

tells whether the first voltage-limit is set with respect to the open circuit voltage or not.

Returns

true if the first voltage-limit is set with respect to the open-circuit voltage and false if not.

Note

if no value was set, the default is false.

15.7.3.14 isSecondVoltageLimitVsOCP()

```
bool AisCyclicVoltammetryElement::isSecondVoltageLimitVsOCP ( ) const
```

tells whether the second voltage-limit is set with respect to the open circuit voltage or not.

Returns

true if the second voltage-limit is set with respect to the open-circuit voltage and false if not.

Note

if no value was set, the default is false.

15.7.3.15 isStartVoltageVsOCP()

```
bool AisCyclicVoltammetryElement::isStartVoltageVsOCP ( ) const
```

tells whether the start voltage is set with respect to the open circuit voltage or not.

Returns

true if the start voltage is set with respect to the open-circuit voltage and false if not.

15.7.3.16 setApproxMaxCurrent()

```
void AisCyclicVoltammetryElement::setApproxMaxCurrent (
             double approxMaxCurrent )
```

set maximum current expected, for manual current range selection.

The is an **optional** parameter. If nothing is set, the device will auto-select the current range.

Parameters

approxMaxCurrent	the value for the maximum current expected in Amps.

15.7.3.17 setAutoRange()

```
void AisCyclicVoltammetryElement::setAutoRange ( )
```

set to auto-select the current range.

This option is set by default. There is no need to call this function to auto-select if the range was not manually set.

15.7.3.18 setdEdt()

```
void AisCyclicVoltammetryElement::setdEdt (
            double dEdt )
```

set the value for the constant scan rate dE/dt.

Parameters

dEdt	the value set for the constant scan rate dE/dt in V/s.

15.7.3.19 setEndVoltage()

```
void AisCyclicVoltammetryElement::setEndVoltage (
            double endVoltage )
```

set the ending potential value.

This is the value of the voltage at which the experiment will stop. After the last cycle, the experiment will do one last sweep towards this value.

Parameters

endVoltage	the value to set for the ending potential in volts.

15.7.3.20 setEndVoltageVsOCP()

```
void AisCyclicVoltammetryElement::setEndVoltageVsOCP (
            bool endVoltageVsOCP )
```

set whether to reference the end voltage against the open-circuit voltage or the reference terminal.

The reference terminal is for you to connect to any reference point you like. Connect it to the working electrode to reference ground.

Parameters

endVoltageVsOCP	true to set the end voltage to be referenced against the open-circuit voltage and false if set against the reference terminal.

15.7.3.21 setFirstVoltageLimit()

```
void AisCyclicVoltammetryElement::setFirstVoltageLimit (
            double v1 )
```

set the first voltage-limit

After the starting voltage, the scan will go to the first voltage-limit. This could result in either upward scan first if the first voltage-limit is higher than the start voltage or downward scan first if the first voltage-limit is lower than the start voltage.

Parameters

v1	first voltage-limit value in volts

15.7.3.22 setFirstVoltageLimitVsOCP()

```
void AisCyclicVoltammetryElement::setFirstVoltageLimitVsOCP (
             bool firstVoltageLimitVsOCP )
```

set whether to reference the first voltage-limit against the open-circuit voltage or not.

The reference terminal is for you to connect to any reference point you like. Connect it to the working electrode to reference ground.

Parameters

firstVoltageLimitVsOCP	true to set the upper voltage to be referenced against the open-circuit voltage and false otherwise.

15.7.3.23 setNumberOfCycles()

```
void AisCyclicVoltammetryElement::setNumberOfCycles (
             int cycles )
```

set the number of cycles to oscillate between the first voltage-limit and the second voltage-limit.

Parameters

cycles	the number of cycles to set

15.7.3.24 setSamplingInterval()

```
void AisCyclicVoltammetryElement::setSamplingInterval (
             double sampInterval )
```

set how frequently we are sampling the data.

Parameters

sampInterval	the data sampling interval value in seconds.

15.7.3.25 setSecondVoltageLimit()

```
void AisCyclicVoltammetryElement::setSecondVoltageLimit (
            double v2 )
```

set the second voltage-limit

After starting from the start-voltage and reaching the first voltage-limit, the scan will go to the second voltage limit. The scan will continue to oscillate between the first and second voltage-limits according to the number of cycles.

Parameters

v2	the second voltage-limit value in volts

15.7.3.26 setSecondVoltageLimitVsOCP()

```
void AisCyclicVoltammetryElement::setSecondVoltageLimitVsOCP (
            bool secondVoltageLimitVsOCP )
```

set whether to reference the second voltage-limit against the open-circuit voltage or not.

The reference terminal is for you to connect to any reference point you like. Connect it to the working electrode to reference ground.

Parameters

secondVoltageLimitVsOCP	true to set the second voltage-limit to be referenced against the open-circuit voltage and false otherwise.

15.7.3.27 setStartVoltage()

```
void AisCyclicVoltammetryElement::setStartVoltage (
            double startVoltage )
```

set the value for the start voltage.

Parameters

startVoltage	the value of the start voltage in volts

15.7.3.28 setStartVoltageVsOCP()

```
void AisCyclicVoltammetryElement::setStartVoltageVsOCP (
            bool startVoltageVsOCP )
```

set whether to reference the start voltage against the open-circuit voltage or the reference terminal.

The reference terminal is for you to connect to any reference point you like. Connect it to the working electrode to reference ground.

Parameters

startVoltageVsOCP	true to if the start voltage is set to reference the open-circuit voltage and false if set against the reference terminal.

The documentation for this class was generated from the following file:

- AisCyclicVoltammetryElement.h

15.8 AisDCCurrentSweepElement Class Reference

this experiment performs a DC current sweep from the **starting current** to the **ending current** which progresses linearly according to the **scan rate**.

```
#include <AisDCCurrentSweepElement.h>
```

Inherits AisAbstractElement.

Public Member Functions

- AisDCCurrentSweepElement (double startCurrent, double endCurrent, double scanRate, double samplingInterval)

  the DC current sweep element.
- **AisDCCurrentSweepElement** (const AisDCCurrentSweepElement &)

  copy constructor for the AisDCCurrentSweepElement object.
- AisDCCurrentSweepElement & **operator=** (const AisDCCurrentSweepElement &)

  overload equal to operator for the AisDCCurrentSweepElement object.
- QString getName () const override

  get the name of the element.
- QStringList getCategory () const override

  get a list of applicable categories of the element.

- double getStartingCurrent () const

  get the value set for the starting current.
- void setStartingCurrent (double startingCurrent)

  set the value for the starting current.
- double getEndingCurrent () const

  get the value set for the ending current.
- void setEndingCurrent (double endingCurrent)

  set the value for the ending current.
- double getScanRate () const

  get the value set for the scan rate.
- void setScanRate (double scanRate)

  set the value for the current scan rate.
- double getSamplingInterval () const

  get how frequently we are sampling the data.
- void setSamplingInterval (double samplingInterval)

  set how frequently we are sampling the data.
- double getMaxVoltage () const

  get the value set for the maximum voltage. The experiment will end when it reaches this value.
- void setMaxVoltage (double maxVoltage)

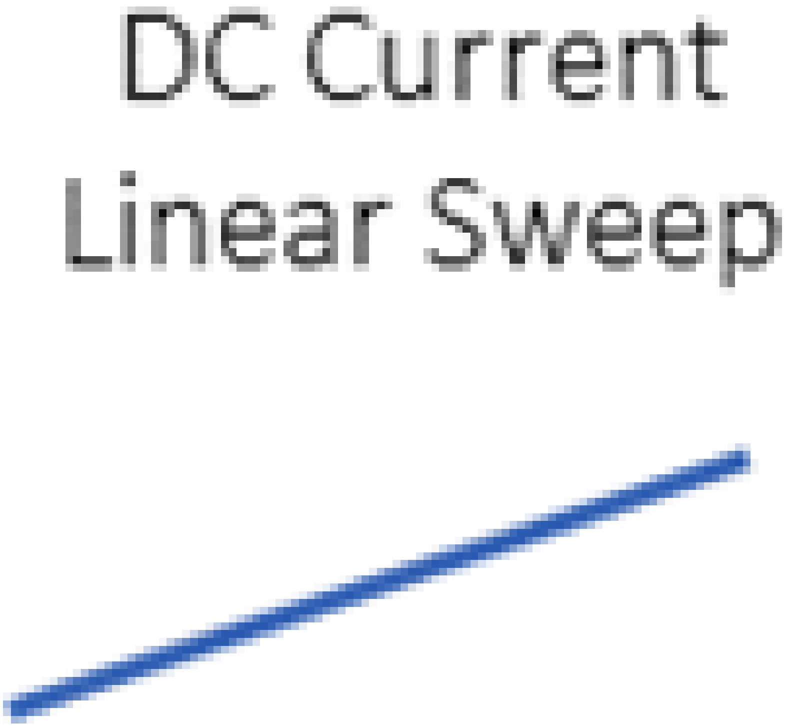
  set a maximum voltage to stop the experiment.
- double getMinVoltage () const

  get the value set minimum for the voltage in volts.
- void setMinVoltage (double minVoltage)

  set a minimum voltage to stop the experiment.

15.8.1 Detailed Description

this experiment performs a DC current sweep from the **starting current** to the **ending current** which progresses linearly according to the **scan rate**.



15.8.2 Constructor & Destructor Documentation

15.8.2.1 AisDCCurrentSweepElement()

```
AisDCCurrentSweepElement::AisDCCurrentSweepElement (
             double startCurrent,
             double endCurrent,
             double scanRate,
             double samplingInterval )  [explicit]
```

the DC current sweep element.

Parameters

startCurrent	the value for the starting current in Amps.
endCurrent	the value for the ending current in Amps.
scanRate	the value for the current scan rate in A/s.
samplingInterval	how frequently we are sampling the data.

15.8.3 Member Function Documentation

15.8.3.1 getCategory()

```
QStringList AisDCCurrentSweepElement::getCategory ( ) const  [override]
```

get a list of applicable categories of the element.

Returns

A list of applicable categories: ("Galvanostatic Control", "Basic Voltammetry").

get a list of applicable categories of the element.

Returns

A list of applicable categories: ("Galvanostatic Control", "Basic Voltammetry").

15.8.3.2 getEndingCurrent()

```
double AisDCCurrentSweepElement::getEndingCurrent ( ) const
```

get the value set for the ending current.

Returns

the value for the ending current in Amps.

15.8.3.3 getMaxVoltage()

```
double AisDCCurrentSweepElement::getMaxVoltage ( ) const
```

get the value set for the maximum voltage. The experiment will end when it reaches this value.

Returns

the value set for the maximum voltage.

Note

this is an optional parameter. If no value has been set, the default value is positive infinity

15.8.3.4 getMinVoltage()

```
double AisDCCurrentSweepElement::getMinVoltage ( ) const
```

get the value set minimum for the voltage in volts.

Returns

the value set for the minimum voltage in volts.

Note

this is an optional parameter. If no value has been set, the default value is negative infinity

15.8.3.5 getName()

```
QString AisDCCurrentSweepElement::getName ( ) const  [override]
```

get the name of the element.

Returns

The name of the element: "DC Current Linear Sweep".

15.8.3.6 getSamplingInterval()

```
double AisDCCurrentSweepElement::getSamplingInterval ( ) const
```

get how frequently we are sampling the data.

Returns

the data sampling interval value in seconds.

15.8.3.7 getScanRate()

```
double AisDCCurrentSweepElement::getScanRate ( ) const
```

get the value set for the scan rate.

Returns

the value set for the scan rate in A/s.

See also

setScanRate

15.8.3.8 getStartingCurrent()

```
double AisDCCurrentSweepElement::getStartingCurrent ( ) const
```

get the value set for the starting current.

Returns

the value set for the constant current in Amps.

15.8.3.9 setEndingCurrent()

```
void AisDCCurrentSweepElement::setEndingCurrent (
             double endingCurrent )
```

set the value for the ending current.

Parameters

endingCurrent	the value for the ending current in Amps

15.8.3.10 setMaxVoltage()

```
void AisDCCurrentSweepElement::setMaxVoltage (
             double maxVoltage )
```

set a maximum voltage to stop the experiment.

The is an **optional** condition. If nothing is set, then the experiment will not stop based on an upper-limit voltage value. If a maximum voltage is set, the experiment will continue to run as long as the measured voltage is below that value.

Parameters

maxVoltage	the maximum voltage value in volts at which the experiment will stop.

15.8.3.11 setMinVoltage()

```
void AisDCCurrentSweepElement::setMinVoltage (
             double minVoltage )
```

set a minimum voltage to stop the experiment.

The is an **optional** condition. If nothing is set, then the experiment will not stop based on an lower-limit voltage value. If a maximum voltage is set, the experiment will continue to run as long as the measured voltage is above that value.

Parameters

minVoltage	the minimum voltage value in volts at which the experiment will stop.

15.8.3.12 setSamplingInterval()

```
void AisDCCurrentSweepElement::setSamplingInterval (
             double samplingInterval )
```

set how frequently we are sampling the data.

Parameters

samplingInterval	the data sampling interval value in seconds.

15.8.3.13 setScanRate()

```
void AisDCCurrentSweepElement::setScanRate (
             double scanRate )
```

set the value for the current scan rate.

The scan rate represents the value of the discrete current step size in one second in the linear sweep.

Parameters

scanRate	the value to set for the scan rate.

15.8.3.14 setStartingCurrent()

```
void AisDCCurrentSweepElement::setStartingCurrent (
             double startingCurrent )
```

set the value for the starting current.

Parameters

startingCurrent	the value to set for the starting current in Amps

The documentation for this class was generated from the following file:

- AisDCCurrentSweepElement.h

15.9 AisDCData Struct Reference

a structure containing DC data information.

```
#include <AisDataPoints.h>
```

Public Attributes

- double **timestamp**

  the time at which the DC data arrived.
- double **workingElectrodeVoltage**

  the measured working electrode voltage in volts.
- double **counterElectrodeVoltage**

  the measured counter electrode voltage in volts.
- double **current**

  the measured electric current value in Amps
- double **temperature**

  the measured temperature in Celsius.

15.9.1 Detailed Description

a structure containing DC data information.

The documentation for this struct was generated from the following file:

- AisDataPoints.h

15.10 AisDCPotentialSweepElement Class Reference

this experiment performs a DC potential sweep from the **starting current** to the **ending current** which progresses linearly according to the **scan rate**.

```
#include <AisDCPotentialSweepElement.h>
```

Inherits AisAbstractElement.

Public Member Functions

- AisDCPotentialSweepElement (double startPotential, double endPotential, double scanRate, double samplingInterval)

  the potential sweep element constructor.
- **AisDCPotentialSweepElement** (const AisDCPotentialSweepElement &)

  copy constructor for the AisDCPotentialSweepElement object.
- AisDCPotentialSweepElement & **operator=** (const AisDCPotentialSweepElement &)

  overload equal to operator for the AisDCPotentialSweepElement object.
- QString getName () const override

  get the name of the element.
- QStringList getCategory () const override

  get a list of applicable categories of the element.
- double getStartingPot () const

  get the value set for the starting potential.
- void setStartingPot (double startingPotential)

  set the value for the starting potential.
- bool isStartVoltageVsOCP () const

  tells whether the starting potential is set against the open-circuit voltage or the reference terminal.
- void setStartVoltageVsOCP (bool startVoltageVsOCP)

  set whether to reference the starting potential against the open-circuit voltage or the reference terminal.
- double getEndingPot () const

  get the value set for the ending potential value.
- void setEndingPot (double endingPotential)

  set the ending potential value.
- bool isEndVoltageVsOCP () const

  tells whether the end voltage is set with respect to the open circuit voltage or the reference terminal.
- void setEndVoltageVsOCP (bool endVoltageVsOCP)

  set whether to reference the end voltage against the open-circuit voltage or the reference terminal.
- double getScanRate () const

  get the value set for the voltage scan rate.
- void setScanRate (double scanRate)

  set the value for the voltage scan rate.
- double getSamplingInterval () const

  get how frequently we are sampling the data.
- void setSamplingInterval (double samplingInterval)

  set how frequently we are sampling the data.
- bool isAutoRange () const

  tells whether the current range is set to auto-select or not.
- void setAutoRange ()

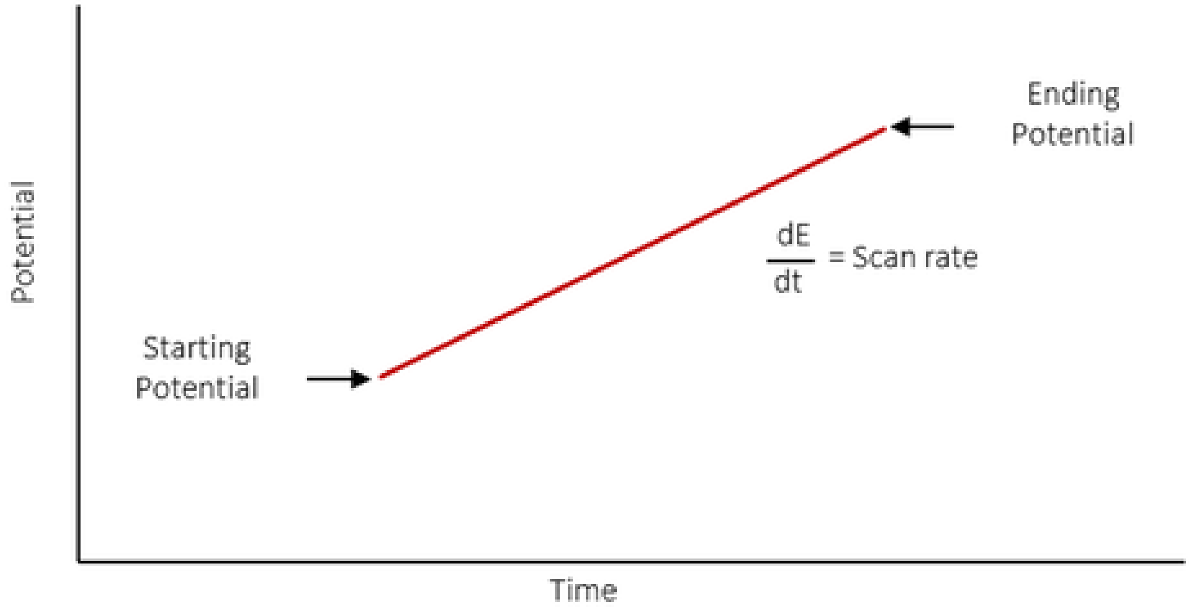
  set to auto-select the current range.
- double getApproxMaxCurrent () const

  get the value set for the expected maximum current.
- void setApproxMaxCurrent (double approxMaxCurrent)

  set maximum current expected, for manual current range selection.

15.10.1 Detailed Description

this experiment performs a DC potential sweep from the **starting current** to the **ending current** which progresses linearly according to the **scan rate**.



15.10.2 Constructor & Destructor Documentation

15.10.2.1 AisDCPotentialSweepElement()

```
AisDCPotentialSweepElement::AisDCPotentialSweepElement (
            double startPotential,
            double endPotential,
            double scanRate,
            double samplingInterval )  [explicit]
```

the potential sweep element constructor.

Parameters

startPotential	the value of the starting potential in volts
endPotential	the value of the ending potential in volts
scanRate	the voltage scan rate in V/s
samplingInterval	how frequently we are sampling the data.

15.10.3 Member Function Documentation

15.10.3.1 getApproxMaxCurrent()

```
double AisDCPotentialSweepElement::getApproxMaxCurrent ( ) const
```

get the value set for the expected maximum current.

Returns

the value set for the expected maximum current in Amps.

Note

if nothing was manually set, the device will auto-select the current range and the return value will be positive infinity.

15.10.3.2 getCategory()

```
QStringList AisDCPotentialSweepElement::getCategory ( ) const  [override]
```

get a list of applicable categories of the element.

Returns

A list of applicable categories: ("Potentiostatic Control", "Basic Experiments").

get a list of applicable categories of the element.

Returns

A list of applicable categories: ("Potentiostatic Control", "Basic Experiments").

15.10.3.3 getEndingPot()

```
double AisDCPotentialSweepElement::getEndingPot ( ) const
```

get the value set for the ending potential value.

This is the value of the voltage at which the experiment will stop.

Returns

the value set for the ending voltage in volts.

15.10.3.4 getName()

```
QString AisDCPotentialSweepElement::getName ( ) const  [override]
```

get the name of the element.

Returns

The name of the element: "DC Potential Linear Sweep".

15.10.3.5 getSamplingInterval()

```
double AisDCPotentialSweepElement::getSamplingInterval ( ) const
```

get how frequently we are sampling the data.

Returns

the data sampling interval value in seconds.

15.10.3.6 getScanRate()

```
double AisDCPotentialSweepElement::getScanRate ( ) const
```

get the value set for the voltage scan rate.

Returns

the value set for the voltage scan rate in V/s

See also

setScanRate

15.10.3.7 getStartingPot()

```
double AisDCPotentialSweepElement::getStartingPot ( ) const
```

get the value set for the starting potential.

Returns

the value of the starting potential in volts.

15.10.3.8 isAutoRange()

```
bool AisDCPotentialSweepElement::isAutoRange ( ) const
```

tells whether the current range is set to auto-select or not.

Returns

true if the current range is set to auto-select and false if a rage has been selected.

15.10.3.9 isEndVoltageVsOCP()

```
bool AisDCPotentialSweepElement::isEndVoltageVsOCP ( ) const
```

tells whether the end voltage is set with respect to the open circuit voltage or the reference terminal.

Returns

true if the end voltage is set with respect to the open-circuit voltage and false if set against the reference terminal.

See also

setEndVoltageVsOCP

15.10.3.10 isStartVoltageVsOCP()

```
bool AisDCPotentialSweepElement::isStartVoltageVsOCP ( ) const
```

tells whether the starting potential is set against the open-circuit voltage or the reference terminal.

Returns

true if the starting potential is set against the open-circuit voltage and false if it is set against the reference terminal.

See also

setStartVoltageVsOCP

15.10.3.11 setApproxMaxCurrent()

```
void AisDCPotentialSweepElement::setApproxMaxCurrent (
             double approxMaxCurrent )
```

set maximum current expected, for manual current range selection.

The is an **optional** parameter. If nothing is set, the device will auto-select the current range.

Parameters

approxMaxCurrent	the value for the maximum current expected in Amps.

15.10.3.12 setAutoRange()

```
void AisDCPotentialSweepElement::setAutoRange ( )
```

set to auto-select the current range.

This option is set by default. There is no need to call this function to auto-select if the range was not manually set.

15.10.3.13 setEndingPot()

```
void AisDCPotentialSweepElement::setEndingPot (
             double endingPotential )
```

set the ending potential value.

This is the value of the voltage at which the experiment will stop.

Parameters

endingPotential	the value to set for the ending potential in volts.

15.10.3.14 setEndVoltageVsOCP()

```
void AisDCPotentialSweepElement::setEndVoltageVsOCP (
             bool endVoltageVsOCP )
```

set whether to reference the end voltage against the open-circuit voltage or the reference terminal.

The reference terminal is for you to connect to any reference point you like. Connect it to the working electrode to reference ground.

Parameters

endVoltageVsOCP	true to set the end voltage to be referenced against the open-circuit voltage and false if set against the reference terminal.

Note

by default, this is set to false.

15.10.3.15 setSamplingInterval()

```
void AisDCPotentialSweepElement::setSamplingInterval (
            double samplingInterval )
```

set how frequently we are sampling the data.

Parameters

samplingInterval	the data sampling interval value in seconds.

15.10.3.16 setScanRate()

```
void AisDCPotentialSweepElement::setScanRate (
            double scanRate )
```

set the value for the voltage scan rate.

The scan rate represents the value of the discrete voltage step size in one second in the linear sweep.

Parameters

scanRate	the value to set for the scan rate.

15.10.3.17 setStartingPot()

```
void AisDCPotentialSweepElement::setStartingPot (
            double startingPotential )
```

set the value for the starting potential.

Parameters

startingPotential	the value of the starting potential in volts

15.10.3.18 setStartVoltageVsOCP()

```
void AisDCPotentialSweepElement::setStartVoltageVsOCP (
            bool startVoltageVsOCP )
```

set whether to reference the starting potential against the open-circuit voltage or the reference terminal.

The reference terminal is for you to connect to any reference point you like. Connect it to the working electrode to reference ground.

Parameters

startVoltageVsOCP	true to if the starting potential is set to reference the open-circuit voltage and false if set against the reference terminal.

Note

by default, this is set to false.

The documentation for this class was generated from the following file:

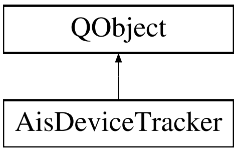
- AisDCPotentialSweepElement.h

15.11 AisDeviceTracker Class Reference

This class is used track device connections to the computer. It can establish connection with plugged-in devices. It also provides instrument handlers specific to each connected device which can provide control of the specific device like starting experiments.

```
#include <AisDeviceTracker.h>
```

Inheritance diagram for AisDeviceTracker:



Signals

- void newDeviceConnected (const QString &deviceName)

  a signal to be emitted whenever a new connection has been successfully established with a device.
- void deviceDisconnected (const QString &deviceName)

  a signal to be emitted whenever a device has been disconnected.
- void **firmwareUpdateNotification** (const QString &message)

Public Member Functions

- AisErrorCode connectToDeviceOnComPort (const QString &comPort)

  establish a connection with a device connected on a USB port.
- const AisInstrumentHandler & getInstrumentHandler (const QString &deviceName) const

  get an instrument handler to control a specific device.
- const std::list< QString > getConnectedDevices () const

  get a list of all the connected devices.
- int connectAllPluggedInDevices ()

  connect all devices physically plugged to the computer.
- AisErrorCode updateFirmwareOnComPort (QString comport) const

  update firmware on connected device at USB port.
- int updateFirmwareOnAllAvailableDevices ()

  request firmware update for all available devices.

Static Public Member Functions

- static AisDeviceTracker ∗ **Instance** ()

15.11.1 Detailed Description

This class is used track device connections to the computer. It can establish connection with plugged-in devices. It also provides instrument handlers specific to each connected device which can provide control of the specific device like starting experiments.

15.11.2 Member Function Documentation

15.11.2.1 connectAllPluggedInDevices()

```
int AisDeviceTracker::connectAllPluggedInDevices ( )
```

connect all devices physically plugged to the computer.

This will automatically detect all the communication ports that have devices plugged in and establish a connection with each.

Returns

the number of *new* devices that have successfully established a connection with the computer. If a device has already been connected before calling this function, it will not be counted in the return value.

Note

emits newDeviceConnected() signal with the device name for each successful connection.

15.11.2.2 connectToDeviceOnComPort()

```
AisErrorCode AisDeviceTracker::connectToDeviceOnComPort (
             const QString & comPort )
```

establish a connection with a device connected on a USB port.

Parameters

comPort	the communication port to connect through.

Returns

AisErrorCode::Success if a connection was established with the device through the given communication port. If not successful, possible returned errors are:

- AisErrorCode::Unknown
- AisErrorCode::FirmwareNotSupported
- AisErrorCode::ConnectionFailed

Note

emits newDeviceConnected() signal with the device name if establishing the connection was successful.

You need to specify the communication port specific to your computer. For example, on PC, you may find your port number through the 'device manager'. An example would be "COM15".

15.11.2.3 deviceDisconnected

```
void AisDeviceTracker::deviceDisconnected (
            const QString & deviceName )  [signal]
```

a signal to be emitted whenever a device has been disconnected.

Parameters

deviceName	the name of the newly disconnected device.

15.11.2.4 getConnectedDevices()

```
const std::list< QString > AisDeviceTracker::getConnectedDevices ( ) const
```

get a list of all the connected devices.

Returns

a list of all the connected devices.

15.11.2.5 getInstrumentHandler()

```
const AisInstrumentHandler & AisDeviceTracker::getInstrumentHandler (
            const QString & deviceName ) const
```

get an instrument handler to control a specific device.

Parameters

deviceName	the name of the connected device to get the instrument handler for.

Returns

the instrument handler that controls the specified device.

Note

You may get a list of the connected devices using getConnectedDevices(). Also, whenever a device has been connected by calling connectToDeviceOnComPort(), a signal is emitted with the device name. A signal and slot example is shown here.

See also

AisInstrumentHandler

connectToDeviceOnComPort()

getConnectedDevices()

15.11.2.6 newDeviceConnected

```
void AisDeviceTracker::newDeviceConnected (
            const QString & deviceName )  [signal]
```

a signal to be emitted whenever a new connection has been successfully established with a device.

Parameters

deviceName	the name of the newly connected device.

Note

this signal will be emitted for each newly connected device whenever either connectToDeviceOnComPort() or connectAllPluggedInDevices() successfully established connections.

15.11.2.7 updateFirmwareOnAllAvailableDevices()

```
int AisDeviceTracker::updateFirmwareOnAllAvailableDevices ( )
```

request firmware update for all available devices.

This will automatically detect devices not currently in use and update firmware if necessary.

Returns

the number of devices that have successfully requested for firmware update. If a device has already been updated firmware before calling this function, it will not be counted in the return value. If any error is generated while requesting firmware update, it will not be counted in the return value.

Note

emits firmwareUpdateNotification() signal will provide notification regarding firmware update of all devices.

You can update firmware when you reset the device physically through reset button.

See also

updateFirmwareOnComPort

15.11.2.8 updateFirmwareOnComPort()

```
AisErrorCode AisDeviceTracker::updateFirmwareOnComPort (
             QString comport ) const
```

update firmware on connected device at USB port.

Parameters

comPort	the communication port to connect through.

Returns

AisErrorCode::Success if firmware update successfully initiated through the given communication port. If not successful, possible returned errors are:

- AisErrorCode::FirmwareUptodate
- AisErrorCode::ConnectionFailed

Note

emits firmwareUpdateNotification() signal to provide firmware update progress.

You need to specify the communication port specific to your computer. For example, on PC, you may find your port number through the 'device manager'. An example would be "COM15".

The documentation for this class was generated from the following file:

- AisDeviceTracker.h

15.12 AisDiffPulseVoltammetryElement Class Reference

In this experiment, the working electrode holds at a **starting potential** during the **quiet time**. Then it applies a train of pulses superimposed on a staircase waveform, with a uniform **potential step** size. The potential continues to step until the **final potential** is reached.

```
#include <AisDiffPulseVoltammetryElement.h>
```

Inherits AisAbstractElement.

Public Member Functions

- AisDiffPulseVoltammetryElement (double startVoltage, double endVoltage, double voltageStep, double pulseHeight, double pulseWidth, double pulsePeriod)

  the differential pulse element constructor.
- **AisDiffPulseVoltammetryElement** (const AisDiffPulseVoltammetryElement &)

  copy constructor for the AisDiffPulseVoltammetryElement object.
- AisDiffPulseVoltammetryElement & **operator=** (const AisDiffPulseVoltammetryElement &)

  overload equal to operator for the AisDiffPulseVoltammetryElement object.
- QString getName () const override

  get the name of the element.
- QStringList getCategory () const override

  get a list of applicable categories of the element.
- double getStartVoltage () const

  get the value set for the start voltage.
- void setStartVoltage (double startVoltage)

  set the value for the start voltage.
- bool isStartVoltageVsOCP () const

  tells whether the starting potential is set against the open-circuit voltage or the reference terminal.
- void setStartVoltageVsOCP (bool startVoltageVsOCP)

  set whether to reference the starting potential against the open-circuit voltage or the reference terminal.
- double getEndVoltage () const

  get the value set for the ending potential value.
- void setEndVoltage (double endVoltage)

  set the ending potential value.
- bool isEndVoltageVsOCP () const

  tells whether the end voltage is set with respect to the open circuit voltage or the reference terminal.
- void setEndVoltageVsOCP (bool endVoltageVsOCP)

  set whether to reference the end voltage against the open-circuit voltage or the reference terminal.
- double getVStep () const

  get the value set for the potential step.
- void setVStep (double vStep)

  set the value for the potential step.
- double getPulseHeight () const

  get the value set for the pulse height.
- void setPulseHeight (double pulseHeight)

  set the value for the pulse height.
- double getPulseWidth () const

  get the value set for the pulse width.
- void setPulseWidth (double pulseWidth)

  set the value for the pulse width.
- double getPulsePeriod () const

  get the value set for the pulse period.
- void setPulsePeriod (double pulsePeriod)

  set the value for the pulse period.
- bool isAutoRange () const

  tells whether the current range is set to auto-select or not.
- void setAutoRange ()

  set to auto-select the current range.
- double getApproxMaxCurrent () const

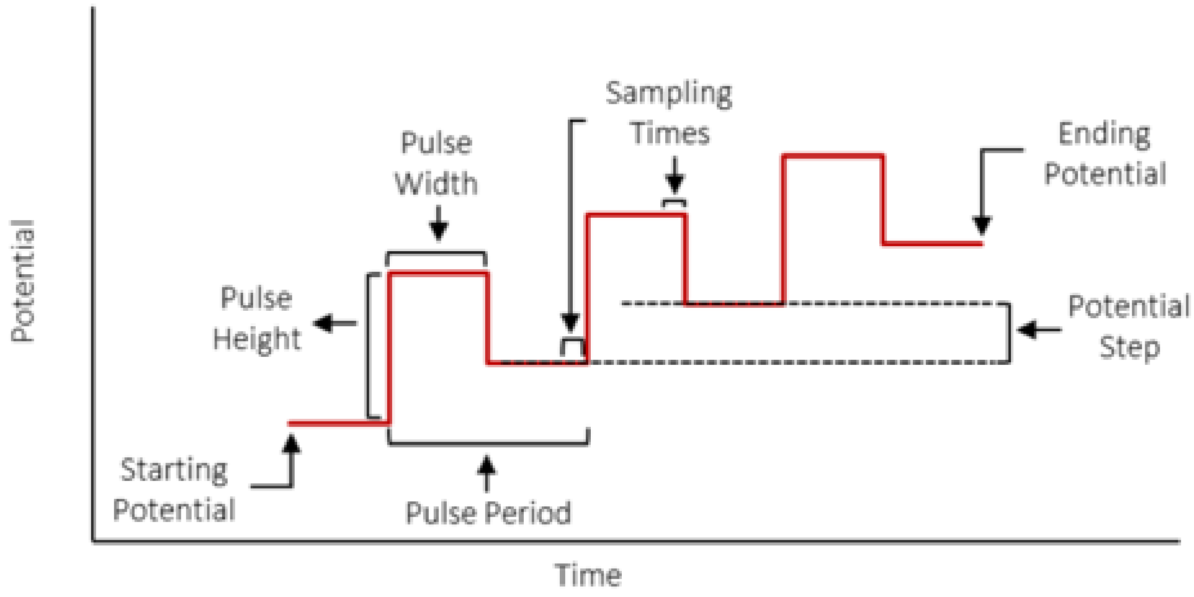
  get the value set for the expected maximum current.
- void setApproxMaxCurrent (double approxMaxCurrent)

  set maximum current expected, for manual current range selection.

15.12.1 Detailed Description

In this experiment, the working electrode holds at a **starting potential** during the **quiet time**. Then it applies a train of pulses superimposed on a staircase waveform, with a uniform **potential step** size. The potential continues to step until the **final potential** is reached.

The **pulse width** is the amount of time between the rising and falling edge of a pulse. The **pulse period** is the amount of time between the beginning of one pulse and the beginning of the next.



15.12.2 Constructor & Destructor Documentation

15.12.2.1 AisDiffPulseVoltammetryElement()

```
AisDiffPulseVoltammetryElement::AisDiffPulseVoltammetryElement (
              double startVoltage,
              double endVoltage,
              double voltageStep,
              double pulseHeight,
              double pulseWidth,
              double pulsePeriod )  [explicit]
```

the differential pulse element constructor.

Parameters

startVoltage	the value of the starting potential in volts
endVoltage	the value of the ending potential in volts
voltageStep	the value set for the voltage step in volts.
pulseHeight	the value for the pulse height in volts.
pulseWidth	the value for the pulse width in volts.
pulsePeriod	the value for the pulse period in volts.

15.12.3 Member Function Documentation

15.12.3.1 getApproxMaxCurrent()

```
double AisDiffPulseVoltammetryElement::getApproxMaxCurrent ( ) const
```

get the value set for the expected maximum current.

Returns

the value set for the expected maximum current in Amps.

Note

if nothing was manually set, the device will auto-select the current range and the return value will be positive infinity.

15.12.3.2 getCategory()

```
QStringList AisDiffPulseVoltammetryElement::getCategory ( ) const  [override]
```

get a list of applicable categories of the element.

Returns

A list of applicable categories: ("Potentiostatic Control", "Basic Voltammetry", "Pulse Voltammetry").

get a list of applicable categories of the element.

Returns

A list of applicable categories: ("Potentiostatic Control", "Basic Voltammetry", "Pulse Voltammetry").

15.12.3.3 getEndVoltage()

```
double AisDiffPulseVoltammetryElement::getEndVoltage ( ) const
```

get the value set for the ending potential value.

This is the value of the voltage at which the experiment will stop.

Returns

the value set for the ending voltage in volts.

15.12.3.4 getName()

```
QString AisDiffPulseVoltammetryElement::getName ( ) const  [override]
```

get the name of the element.

Returns

The name of the element: "Differential Pulse Potential Voltammetry".

15.12.3.5 getPulseHeight()

```
double AisDiffPulseVoltammetryElement::getPulseHeight ( ) const
```

get the value set for the pulse height.

Returns

the value set for the pulse height in volts.

See also

setPulseHeight

15.12.3.6 getPulsePeriod()

```
double AisDiffPulseVoltammetryElement::getPulsePeriod ( ) const
```

get the value set for the pulse period.

Returns

the value set for the pulse period in seconds.

See also

setPulsePeriod

15.12.3.7 getPulseWidth()

```
double AisDiffPulseVoltammetryElement::getPulseWidth ( ) const
```

get the value set for the pulse width.

Returns

the value set for the pulse width in seconds.

See also

setPulseWidth

15.12.3.8 getStartVoltage()

```
double AisDiffPulseVoltammetryElement::getStartVoltage ( ) const
```

get the value set for the start voltage.

Returns

the value of the start voltage in volts.

15.12.3.9 getVStep()

```
double AisDiffPulseVoltammetryElement::getVStep ( ) const
```

get the value set for the potential step.

Returns

the value set for the potential step in volts.

See also

setVStep

15.12.3.10 isAutoRange()

```
bool AisDiffPulseVoltammetryElement::isAutoRange ( ) const
```

tells whether the current range is set to auto-select or not.

Returns

true if the current range is set to auto-select and false if a rage has been selected.

15.12.3.11 isEndVoltageVsOCP()

```
bool AisDiffPulseVoltammetryElement::isEndVoltageVsOCP ( ) const
```

tells whether the end voltage is set with respect to the open circuit voltage or the reference terminal.

Returns

true if the end voltage is set with respect to the open-circuit voltage and false if set against the reference terminal.

See also

setEndVoltageVsOCP

15.12.3.12 isStartVoltageVsOCP()

```
bool AisDiffPulseVoltammetryElement::isStartVoltageVsOCP ( ) const
```

tells whether the starting potential is set against the open-circuit voltage or the reference terminal.

Returns

true if the starting potential is set against the open-circuit voltage and false if it is set against the reference terminal.

See also

setStartVoltageVsOCP

15.12.3.13 setApproxMaxCurrent()

```
void AisDiffPulseVoltammetryElement::setApproxMaxCurrent (
             double approxMaxCurrent )
```

set maximum current expected, for manual current range selection.

The is an **optional** parameter. If nothing is set, the device will auto-select the current range.

Parameters

approxMaxCurrent	the value for the maximum current expected in Amps.

15.12.3.14 setAutoRange()

```
void AisDiffPulseVoltammetryElement::setAutoRange ( )
```

set to auto-select the current range.

This option is set by default. There is no need to call this function to auto-select if the range was not manually set.

15.12.3.15 setEndVoltage()

```
void AisDiffPulseVoltammetryElement::setEndVoltage (
             double endVoltage )
```

set the ending potential value.

This is the value of the voltage at which the experiment will stop.

Parameters

endVoltage	the value to set for the ending voltage in volts.

15.12.3.16 setEndVoltageVsOCP()

```
void AisDiffPulseVoltammetryElement::setEndVoltageVsOCP (
             bool endVoltageVsOCP )
```

set whether to reference the end voltage against the open-circuit voltage or the reference terminal.

The reference terminal is for you to connect to any reference point you like. Connect it to the working electrode to reference ground.

Parameters

endVoltageVsOCP	true to set the end voltage to be referenced against the open-circuit voltage and false if set against the reference terminal.

Note

by default, this is set to false.

15.12.3.17 setPulseHeight()

```
void AisDiffPulseVoltammetryElement::setPulseHeight (
            double pulseHeight )
```

set the value for the pulse height.

For the first pulse, the pulse height is added to the starting potential. For the next pulse, the pulse height is added to the potential voltage and the potential step. In general, the pulse height is added to the potential step and the starting voltage of the last pulse.

Parameters

pulseHeight	the value to set for the pulse height in volts.

15.12.3.18 setPulsePeriod()

```
void AisDiffPulseVoltammetryElement::setPulsePeriod (
            double pulsePeriod )
```

set the value for the pulse period.

The pulse period is the time spent between the starts of two consecutive pulses.

Parameters

pulsePeriod	the value to set for the pulse period in seconds.

15.12.3.19 setPulseWidth()

```
void AisDiffPulseVoltammetryElement::setPulseWidth (
            double pulseWidth )
```

set the value for the pulse width.

The pulse width is the value in seconds for the time spent at the same voltage set for the pulse height.

Parameters

pulseWidth	the value to set for the pulse width in seconds.

See also

setPulseHeight

15.12.3.20 setStartVoltage()

```
void AisDiffPulseVoltammetryElement::setStartVoltage (
            double startVoltage )
```

set the value for the start voltage.

Parameters

startVoltage	the value of the start voltage in volts

15.12.3.21 setStartVoltageVsOCP()

```
void AisDiffPulseVoltammetryElement::setStartVoltageVsOCP (
            bool startVoltageVsOCP )
```

set whether to reference the starting potential against the open-circuit voltage or the reference terminal.

The reference terminal is for you to connect to any reference point you like. Connect it to the working electrode to reference ground.

Parameters

startVoltageVsOCP	true to if the starting potential is set to reference the open-circuit voltage and false if set against the reference terminal.

Note

by default, this is set to false.

15.12.3.22 setVStep()

```
void AisDiffPulseVoltammetryElement::setVStep (
            double vStep )
```

set the value for the potential step.

The potential step is the difference between the starting potential of two consecutive pulses.

Parameters

vStep	the value to set for the potential step in volts.

The documentation for this class was generated from the following file:

- AisDiffPulseVoltammetryElement.h

15.13 AisEISGalvanostaticElement Class Reference

This experiment records the complex impedance of the experimental cell in galvanostatic mode, starting from the **start frequency** and sweeping through towards the **end frequency**, with a fixed number of frequency **steps per decade**.

```
#include <AisEISGalvanostaticElement.h>
```

Inherits AisAbstractElement.

Public Member Functions

- AisEISGalvanostaticElement (double startFrequency, double endFrequency, double stepsPerDecade, double currentBias, double currentAamplitude)

  the EIS galvanostatic element constructor.
- **AisEISGalvanostaticElement** (const AisEISGalvanostaticElement &)

  copy constructor for the AisEISGalvanostaticElement object.
- AisEISGalvanostaticElement & **operator=** (const AisEISGalvanostaticElement &)

  overload equal to operator for the AisEISGalvanostaticElement object.
- QString getName () const override

  get the name of the element.
- QStringList getCategory () const override

  get a list of applicable categories of the element.
- double getStartFreq () const

  get the value set for the current starting frequency
- void setStartFreq (double startFreq)

  set the value for the current starting frequency.
- double getEndFreq () const

  the value set for the current ending frequency.
- void setEndFreq (double endFreq)

  set the value for the current end frequency.
- double getStepsPerDecade () const

  get the value set for the current frequency steps per decade.
- void setStepsPerDecade (double stepsPerDecade)

  set the number of the current frequency steps per decade.
- double getBiasCurrent () const

  get the value set for the DC bias (DC offset).
- void setBiasCurrent (double biasCurrent)

  set the value for the DC bias (DC offset).
- double getAmplitude () const

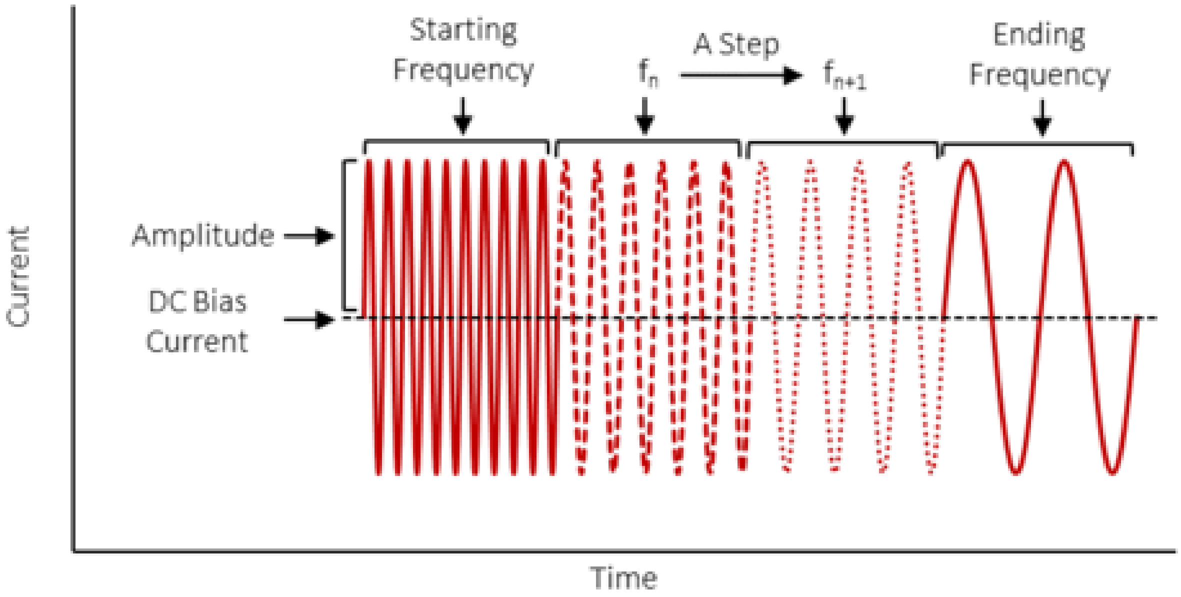
  the value to set for the AC current amplitude.
- void setAmplitude (double amplitude)

  set the value for the AC current amplitude.

15.13.1 Detailed Description

This experiment records the complex impedance of the experimental cell in galvanostatic mode, starting from the **start frequency** and sweeping through towards the **end frequency**, with a fixed number of frequency **steps per decade**.

Important parameters include the **DC bias** and the **AC excitation amplitude**.



15.13.2 Constructor & Destructor Documentation

15.13.2.1 AisEISGalvanostaticElement()

```
AisEISGalvanostaticElement::AisEISGalvanostaticElement (
            double startFrequency,
            double endFrequency,
            double stepsPerDecade,
            double currentBias,
            double currentAamplitude )  [explicit]
```

the EIS galvanostatic element constructor.

Parameters

startFrequency	the value for the current starting frequency
endFrequency	the value for the current ending frequency
stepsPerDecade	the value for the current frequency steps per decade.
currentBias	the value for the DC bias (DC offset).
currentAamplitude	the AC current amplitude.

15.13.3 Member Function Documentation

15.13.3.1 getAmplitude()

```
double AisEISGalvanostaticElement::getAmplitude ( ) const
```

the value to set for the AC current amplitude.

Returns

the value set for the AC current amplitude in Amps.

15.13.3.2 getBiasCurrent()

```
double AisEISGalvanostaticElement::getBiasCurrent ( ) const
```

get the value set for the DC bias (DC offset).

Returns

the value set for the DC bias in Amps.

15.13.3.3 getCategory()

```
QStringList AisEISGalvanostaticElement::getCategory ( ) const  [override]
```

get a list of applicable categories of the element.

Returns

A list of applicable categories: ("Galvanostatic Control", "Impedance Methods", "Basic Experiments").

get a list of applicable categories of the element.

Returns

A list of applicable categories: ("Galvanostatic Control", "Impedance Methods", "Basic Experiments").

15.13.3.4 getEndFreq()

```
double AisEISGalvanostaticElement::getEndFreq ( ) const
```

the value set for the current ending frequency.

Returns

the value set for the current end frequency in Hz

15.13.3.5 getName()

```
QString AisEISGalvanostaticElement::getName ( ) const  [override]
```

get the name of the element.

Returns

The name of the element: "Galvanostatic EIS".

15.13.3.6 getStartFreq()

```
double AisEISGalvanostaticElement::getStartFreq ( ) const
```

get the value set for the current starting frequency

Returns

the value set for the current start frequency in Hz?

15.13.3.7 getStepsPerDecade()

```
double AisEISGalvanostaticElement::getStepsPerDecade ( ) const
```

get the value set for the current frequency steps per decade.

Returns

the value set for the current frequency steps per decade. This is unit-less.

15.13.3.8 setAmplitude()

```
void AisEISGalvanostaticElement::setAmplitude (
             double amplitude )
```

set the value for the AC current amplitude.

Parameters

amplitude	the value to set for the AC current amplitude in Amps.

15.13.3.9 setBiasCurrent()

```
void AisEISGalvanostaticElement::setBiasCurrent (
             double biasCurrent )
```

set the value for the DC bias (DC offset).

Parameters

biasCurrent	the value to set for the DC bias in Amps.

15.13.3.10 setEndFreq()

```
void AisEISGalvanostaticElement::setEndFreq (
             double endFreq )
```

set the value for the current end frequency.

Parameters

endFreq	the value to set for the current end frequency in Hz

15.13.3.11 setStartFreq()

```
void AisEISGalvanostaticElement::setStartFreq (
             double startFreq )
```

set the value for the current starting frequency.

Parameters

startFreq	the value to set the current starting frequency in Hz

15.13.3.12 setStepsPerDecade()

```
void AisEISGalvanostaticElement::setStepsPerDecade (
            double stepsPerDecade )
```

set the number of the current frequency steps per decade.

Parameters

stepsPerDecade	the value to set for the number of steps per decade.

The documentation for this class was generated from the following file:

- AisEISGalvanostaticElement.h

15.14 AisEISPotentiostaticElement Class Reference

This experiment records the complex impedance of the experimental cell in potentiostatic mode, starting from the **start frequency** and sweeping through towards the **end frequency**, with a fixed number of frequency **steps per decade**.

```
#include <AisEISPotentiostaticElement.h>
```

Inherits AisAbstractElement.

Public Member Functions

- AisEISPotentiostaticElement (double startFrequency, double endFrequency, double stepsPerDecade, double voltageBias, double voltageAamplitude)

  the EIS potentiostatic element
- **AisEISPotentiostaticElement** (const AisEISPotentiostaticElement &)

  copy constructor for the AisEISPotentiostaticElement object.
- AisEISPotentiostaticElement & **operator=** (const AisEISPotentiostaticElement &)

  overload equal to operator for the AisEISPotentiostaticElement object.
- QString getName () const override

  get the name of the element.
- QStringList getCategory () const override

  get a list of applicable categories of the element.
- double getStartFreq () const

  get the value set for the voltage starting frequency
- void setStartFreq (double startFreq)

  set the value for the voltage starting frequency.
- double getEndFreq () const

  the value set for the voltage ending frequency.
- void setEndFreq (double endFreq)

  set the value for the voltage end frequency.
- double getStepsPerDecade () const

  get the value set for the voltage frequency steps per decade.
- void setStepsPerDecade (double stepsPerDecade)

set the number of the voltage frequency steps per decade.

- double getBiasVoltage () const

  get the value set for the DC bias (DC offset).
- void setBiasVoltage (double biasVoltage)

  set the value for the DC bias (DC offset).
- bool isBiasVoltageVsOCP () const

  tells whether the DC-bias voltage is referenced against the open-circuit voltage or the reference cable.
- void setBiasVoltageVsOCP (bool biasVsOCP)

  set whether to reference the DC-bias voltage against the open-circuit voltage or the reference terminal.
- double getAmplitude () const

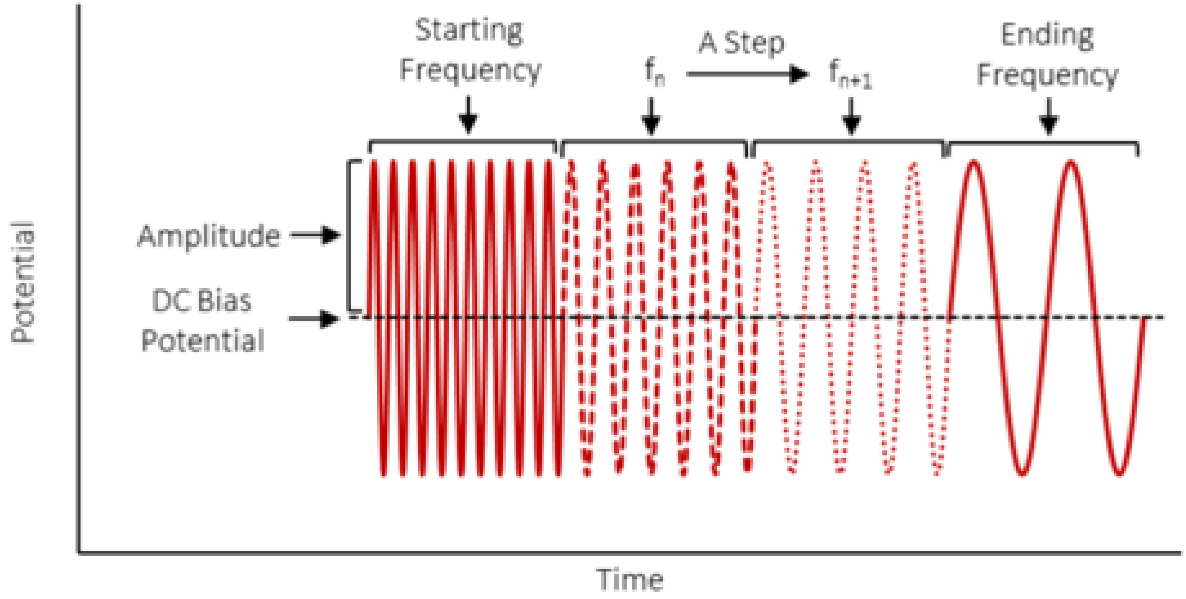
  the value to set for the AC voltage amplitude.
- void setAmplitude (double amplitude)

  set the value for the AC voltage amplitude.

15.14.1 Detailed Description

This experiment records the complex impedance of the experimental cell in potentiostatic mode, starting from the **start frequency** and sweeping through towards the **end frequency**, with a fixed number of frequency **steps per decade**.

Important parameters include the **DC bias** and the **AC excitation amplitude**.



15.14.2 Constructor & Destructor Documentation

15.14.2.1 AisEISPotentiostaticElement()

```
AisEISPotentiostaticElement::AisEISPotentiostaticElement (
             double startFrequency,
             double endFrequency,
             double stepsPerDecade,
             double voltageBias,
             double voltageAamplitude )  [explicit]
```

the EIS potentiostatic element

Parameters

startFrequency	the value for the voltage starting frequency
endFrequency	the value for the voltage ending frequency
stepsPerDecade	the value for the voltage frequency steps per decade.
voltageBias	the value for the DC bias (DC offset).
voltageAamplitude	the AC voltage amplitude.

15.14.3 Member Function Documentation

15.14.3.1 getAmplitude()

```
double AisEISPotentiostaticElement::getAmplitude ( ) const
```

the value to set for the AC voltage amplitude.

Returns

the value set for the AC voltage amplitude in volts.

15.14.3.2 getBiasVoltage()

```
double AisEISPotentiostaticElement::getBiasVoltage ( ) const
```

get the value set for the DC bias (DC offset).

Returns

the value set for the DC bias in volts.

15.14.3.3 getCategory()

```
QStringList AisEISPotentiostaticElement::getCategory ( ) const  [override]
```

get a list of applicable categories of the element.

Returns

A list of applicable categories: ("Potentiostatic Control", "Impedance Methods", "Basic Experiments").

15.14.3.4 getEndFreq()

```
double AisEISPotentiostaticElement::getEndFreq ( ) const
```

the value set for the voltage ending frequency.

Returns

the value set for the voltage end frequency in Hz

15.14.3.5 getName()

```
QString AisEISPotentiostaticElement::getName ( ) const  [override]
```

get the name of the element.

Returns

The name of the element: "Potentiostatic EIS".

15.14.3.6 getStartFreq()

```
double AisEISPotentiostaticElement::getStartFreq ( ) const
```

get the value set for the voltage starting frequency

Returns

the value set for the start frequency in Hz

15.14.3.7 getStepsPerDecade()

```
double AisEISPotentiostaticElement::getStepsPerDecade ( ) const
```

get the value set for the voltage frequency steps per decade.

Returns

the value set for the frequency steps per decade. This is unit-less.

15.14.3.8 isBiasVoltageVsOCP()

```
bool AisEISPotentiostaticElement::isBiasVoltageVsOCP ( ) const
```

tells whether the DC-bias voltage is referenced against the open-circuit voltage or the reference cable.

The reference terminal is for you to connect to any reference point you like. Connect it to the working electrode to reference ground.

Returns

true if the DC-bias voltage is set against the open-circuit voltage and false if it is set against the reference terminal.

15.14.3.9 setAmplitude()

```
void AisEISPotentiostaticElement::setAmplitude (
            double amplitude )
```

set the value for the AC voltage amplitude.

Parameters

amplitude	the value to set for the AC voltage amplitude in volts.

15.14.3.10 setBiasVoltage()

```
void AisEISPotentiostaticElement::setBiasVoltage (
            double biasVoltage )
```

set the value for the DC bias (DC offset).

Parameters

biasVoltage	the value to set for the DC bias in volts.

15.14.3.11 setBiasVoltageVsOCP()

```
void AisEISPotentiostaticElement::setBiasVoltageVsOCP (
            bool biasVsOCP )
```

set whether to reference the DC-bias voltage against the open-circuit voltage or the reference terminal.

Parameters

biasVsOCP	true to if the DC-bias voltage is set to reference the open-circuit voltage and false if set against the reference terminal.

15.14.3.12 setEndFreq()

```
void AisEISPotentiostaticElement::setEndFreq (
            double endFreq )
```

set the value for the voltage end frequency.

Parameters

endFreq	the value to set for the voltage end frequency in Hz

15.14.3.13 setStartFreq()

```
void AisEISPotentiostaticElement::setStartFreq (
            double startFreq )
```

set the value for the voltage starting frequency.

Parameters

startFreq	the value to set the starting frequency Hz

15.14.3.14 setStepsPerDecade()

```
void AisEISPotentiostaticElement::setStepsPerDecade (
            double stepsPerDecade )
```

set the number of the voltage frequency steps per decade.

Parameters

stepsPerDecade	the value to set for the number of steps per decade.

The documentation for this class was generated from the following file:

- AisEISPotentiostaticElement.h

15.15 AisErrorCode Class Reference

This class contains the possible error codes returned to the user when working with the API.

```
#include <AisErrorCode.h>
```

Public Types

- enum ErrorCode : uint8_t {
  Unknown = 255 , Success = 0 , ConnectionFailed = 1 , FirmwareNotSupported = 2 ,
  FirmwareFileNotFound = 3 , **FirmwareUptodate** = 4 , InvalidChannel = 10 , BusyChannel = 11 ,
  DeviceNotFound = 13 , ManualExperimentNotRunning = 51 , ExperimentNotUploaded = 52 ,
  ExperimentIsEmpty = 53 ,
  InvalidParameters = 54 , ChannelNotBusy = 55 , DeviceCommunicationFailed = 100 , FailedToSetManualModeCurrentRange = 101 ,
  FailedToSetManualModeConstantVoltage = 102 , FailedToPauseExperiment = 103 , FailedToResumeExperiment = 104 , FailedToStopExperiment = 105 ,
  FailedToUploadExperiment = 106 , ExperimentAlreadyPaused = 107 , ExperimentAlreadyRun = 108 }

Public Member Functions

- **AisErrorCode** (ErrorCode error)
- **AisErrorCode** (ErrorCode error, QString message)
- QString message () const

  a function to get a message explaining the error.
- int value () const

  a function to get the error code.
- **operator ErrorCode** () const

15.15.1 Detailed Description

This class contains the possible error codes returned to the user when working with the API.

If a function has an AisErrorCode return type, then it needs to be checked for possible failures. The object of this class returned will contain an error code that can be accessed by calling value() member function and an error message that can be accessed by calling

See also

message.

15.15.2 Member Enumeration Documentation

15.15.2.1 ErrorCode

```
enum AisErrorCode::ErrorCode : uint8_t
```

Enumerator

Unknown	indicates that the command failed for an unknown reason.
Success	indicates success.
ConnectionFailed	indicates failure connecting the plugged in device when calling AisDeviceTracker::connectToDeviceOnComPort.
FirmwareNotSupported	indicates failure connecting the plugged in device when calling AisDeviceTracker::connectToDeviceOnComPort because firmware update require.
InvalidChannel	indicates that the given channel number is not valid.
BusyChannel	indicates that failure was due to the channel being busy.
DeviceNotFound	indicates that no device was detected to be connected.
ManualExperimentNotRunning	indicates that the given command applies when there is a manual experiment running on the channel but there is none.
ExperimentNotUploaded	indicates that the given command applies when an experiment has already been uploaded to the channel but there is none.
ExperimentIsEmpty	indicates that the given experiment has no elements. It need to contain at least one.
InvalidParameters	indicates that a given parameter is invalid. For example, it is out of the allowed range.
ChannelNotBusy	indicates that the given command applies when there is an experiment running or paused on the channel but there is none.
DeviceCommunicationFailed	indicates that there was failure in communication with the device.
FailedToSetManualModeCurrentRange	indicates failure of setting manual Mode current range for possible communication failure with the device.
FailedToSetManualModeConstantVoltage	indicates failure of setting manual Mode constant voltage for possible communication failure with the device.
FailedToPauseExperiment	indicates that pausing the experiment failed because either there is no running experiment or for possible communication failure with the device.

Enumerator

FailedToResumeExperiment	indicates that resuming the experiment failed because either there is no paused experiment or for possible communication failure with the device.
FailedToStopExperiment	indicates that stopping the experiment failed because either there is no running or paused experiment or for possible communication failure with the device.
FailedToUploadExperiment	indicates failure to communicate with the device to upload the experiment.
ExperimentAlreadyPaused	indicates that pausing the experiment failed because experiment is already pause.
ExperimentAlreadyRun	indicates that resuming the experiment failed because experiment is already running. it is not in pause state.

15.15.3 Member Function Documentation

15.15.3.1 message()

```
QString AisErrorCode::message ( ) const
```

a function to get a message explaining the error.

Returns

a message that explains the error.

15.15.3.2 value()

```
int AisErrorCode::value ( ) const
```

a function to get the error code.

Returns

the error code

The documentation for this class was generated from the following file:

- AisErrorCode.h

15.16 AisExperiment Class Reference

this class is used to create custom experiments. A custom experiment has a container of contains one or more elements. Once you create elements are set their parameters, you can add them to the container

```
#include <AisExperiment.h>
```

Public Member Functions

- **AisExperiment** ()
  this is the default constructor for the custom experiment.
- AisExperiment (const AisExperiment &exp)
  this is the copy constructor for the custom experiment.
- void operator= (const AisExperiment &exp)
  the assignment operator for the custom experiment.
- QString getExperimentName () const
  get the name of the custom experiment.
- QString getDescription () const
  get a brief description of the custom experiment.
- QStringList getCategory () const
  get the category for the custom experiment.
- void setExperimentName (QString name)
  set a name for the custom experiment.
- void setDescription (QString description)
  set a description for the experiment.
- bool appendElement (AisAbstractElement &element, uint repeat=1)
  append an element to the custom experiment.
- bool appendSubExperiment (const AisExperiment &subExp, uint repeat=1)
  append a sub experiment to this/(the calling) custom experiment.

Friends

- class **AisInstrumentHandler**

15.16.1 Detailed Description

this class is used to create custom experiments. A custom experiment has a container of contains one or more elements. Once you create elements are set their parameters, you can add them to the container

Note

> we call the basic experiments -that are used to build more complex custom experiments- elements. In contexts where both elements and custom experiments are used, elements will be referred to as elements to make the digestion. In other contexts, elements may also be referred to as experiments as they may indeed be used as experiments.

15.16.2 Constructor & Destructor Documentation

15.16.2.1 AisExperiment()

```
AisExperiment::AisExperiment (
                const AisExperiment & exp )  [explicit]
```

this is the copy constructor for the custom experiment.

Parameters

exp	the custom experiment to copy from.

15.16.3 Member Function Documentation

15.16.3.1 appendElement()

```
bool AisExperiment::appendElement (
            AisAbstractElement & element,
            uint repeat = 1 )
```

append an element to the custom experiment.

Parameters

element	an elemental experiment to be appended to this/(the calling) custom experiment.
repeat	the number of times this element is to be repeated. This is an optional parameter and is defaulted to equal 1 when not set.

Returns

true if appending the element was successful and false otherwise.

Note

although an element is an experiment, in the context of custom experiments, it is referred to as an element to make a distinction between the two. In other contexts where such distinction is not needed, an element may still be referred to as an experiment.

15.16.3.2 appendSubExperiment()

```
bool AisExperiment::appendSubExperiment (
            const AisExperiment & subExp,
            uint repeat = 1 )
```

append a sub experiment to this/(the calling) custom experiment.

Parameters

subExp	a sub experiment to be appended to this/(the calling) custom experiment.
repeat	the number of times this sub experiment is to be repeated. This is an optional parameter and is defaulted to equal 1 when not set.

Returns

true if appending the sub experiment was successful and false otherwise.

15.16.3.3 getCategory()

```
QStringList AisExperiment::getCategory ( ) const
```

get the category for the custom experiment.

Returns

the category set for the custom experiment. If no category has been set, the default category returned is ("Custom").

15.16.3.4 getDescription()

```
QString AisExperiment::getDescription ( ) const
```

get a brief description of the custom experiment.

Returns

the description set for the custom experiment. If no description has been set, the default description returned is "Not Defined".

15.16.3.5 getExperimentName()

```
QString AisExperiment::getExperimentName ( ) const
```

get the name of the custom experiment.

Returns

the name set for the custom experiment. If no name has been set, the default name returned is "Custom Experiment"

15.16.3.6 operator=()

```
void AisExperiment::operator= (
             const AisExperiment & exp )
```

the assignment operator for the custom experiment.

Parameters

exp	the custom experiment to copy from.

15.16.3.7 setDescription()

```
void AisExperiment::setDescription (
            QString description )
```

set a description for the experiment.

Parameters

description	the description to be set for the custom experiment.

15.16.3.8 setExperimentName()

```
void AisExperiment::setExperimentName (
            QString name )
```

set a name for the custom experiment.

Parameters

name	the name to be set for the custom experiment.

The documentation for this class was generated from the following file:

- AisExperiment.h

15.17 AisExperimentNode Struct Reference

a structure containing some information regarding the running element.

```
#include <AisDataPoints.h>
```

Public Attributes

- QString **stepName**

  This is the name of the current element running.
- int **stepNumber**

  this number is the order of the element within the custom experiment.
- int **substepNumber**

  this number is the order of the step within the element.

15.17.1 Detailed Description

a structure containing some information regarding the running element.

The documentation for this struct was generated from the following file:

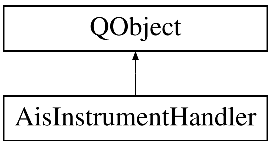
- AisDataPoints.h

15.18 AisInstrumentHandler Class Reference

this class provides control of the device including starting, pausing, resuming and stopping an experiment on a channel as well as reading the data and other controls of the device.

```
#include <AisInstrumentHandler.h>
```

Inheritance diagram for AisInstrumentHandler:



Signals

- void deviceDisconnected ()

  a signal that is emitted if the device associated with this handler has been disconnected.
- void groundFloatStateChanged (bool grounded)

  a signal that is emitted when the floating ground connection state has changed.
- void experimentNewElementStarting (uint8_t channel, const AisExperimentNode &stepInfo)

  a signal that is emitted whenever a new elemental experiment has started.
- void activeDCDataReady (uint8_t channel, const AisDCData &DCData)

  a signal that is emitted whenever new DC data for an active experiment are ready.
- void idleDCDataReady (uint8_t channel, const AisDCData &DCData)

  a signal that is emitted whenever new DC data are ready when the device is in an idle state.
- void recoveryDCDataReady (uint8_t channel, const AisDCData &DCData)

  a signal that is emitted whenever new DC recovery data are ready.
- void activeACDataReady (uint8_t channel, const AisACData &ACData)

  a signal that is emitted whenever new AC data for an active experiment are ready.
- void recoveryACDataReady (uint8_t channel, const AisACData &ACData)

  a signal that is emitted whenever new AC recovery data are ready.
- void experimentStopped (uint8_t channel)

  a signal that is emitted whenever an experiment was stopped manually or has completed.
- void experimentPaused (uint8_t channel)

  a signal that is emitted whenever an experiment was paused.
- void experimentResumed (uint8_t channel)

  a signal that is emitted whenever an experiment was resumed.
- void recoverDataErased (bool successful)

  a signal that is emitted whenever data erase process is completed.

Public Member Functions

- AisErrorCode uploadExperimentToChannel (uint8_t channel, std::shared_ptr< AisExperiment > experiment) const

  upload an already created custom experiment to a specific channel on the device.
- AisErrorCode uploadExperimentToChannel (uint8_t channel, const AisExperiment &experiment) const

  upload an already created custom experiment to a specific channel on the device.
- AisErrorCode startUploadedExperiment (uint8_t channel) const

  start the previously uploaded experiment on the specific channel.
- AisErrorCode skipExperimentStep (uint8_t channel) const

  skip the current experiment step and proceed to the next.
- AisErrorCode pauseExperiment (uint8_t channel) const

  pause a running experiment on the channel.
- AisErrorCode resumeExperiment (uint8_t channel) const

  resume a paused experiment on the channel.
- AisErrorCode stopExperiment (uint8_t channel) const

  stop a running or a paused experiment on the channel.
- double getExperimentUTCStartTime (uint8_t channel) const

  get UTC time for the start of the experiment in seconds.
- AisErrorCode setIRComp (uint8_t channel, double uncompensatedResistance, double compensationLevel) const

  set IR compensation.
- AisErrorCode setCompRange (uint8_t channel, const AisCompRange &compRange) const

  set a compensation range with stability factor and bandwidth index.
- int8_t setLinkedChannels (std::vector< uint8_t > channels) const

  connect several channels together in parallel mode.
- int8_t setBipolarLinkedChannels (std::vector< uint8_t > channels) const

  connect two channels together in bipolar mode.
- bool hasBipolarMode (uint8_t channel) const

  tells whether the given channel is bipolar mode
- std::vector< uint8_t > getLinkedChannels (uint8_t channel) const

  get a list of channels linked to the given channel.
- bool isChannelBusy (uint8_t channel) const

  tells whether the given channel is busy or not.
- bool isChannelPaused (uint8_t channel) const

  tells whether the given channel has a paused experiment or not.
- std::vector< uint8_t > getFreeChannels () const

  get a list of the currently free channels.
- int getNumberOfChannels () const

  get the number of all the channels on this device.
- AisErrorCode eraseRecoverData () const

  delete the recover data from device.
- AisErrorCode startManualExperiment (uint8_t channel) const

  start a manual experiment.
- AisErrorCode setManualModeSamplingInterval (uint8_t channel, double value) const

  set an interval for sampling the data.
- AisErrorCode setManualModeOCP (uint8_t channel) const

  set open-circuit potential mode.
- AisErrorCode setManualModeConstantVoltage (uint8_t channel, double value) const

  set constant voltage for the manual experiment.
- AisErrorCode setManualModeConstantVoltage (uint8_t channel, double value, int currentRangeIndex) const

set constant voltage for the manual experiment and also set a manual current range.

- AisErrorCode setManualModeConstantCurrent (uint8_t channel, double value) const

  set constant current for the manual experiment.

- std::vector< std::pair< double, double > > getManualModeCurrentRangeList (uint8_t channel) const

  get a list of the applicable current ranges to the given channel specific to your device.

15.18.1 Detailed Description

this class provides control of the device including starting, pausing, resuming and stopping an experiment on a channel as well as reading the data and other controls of the device.

You may get an instrument handler instance of this class by calling AisDeviceTracker::getInstrumentHandler where you can get the device name either by calling AisDeviceTracker::getConnectedDevices or whenever the signal newDeviceConnected() is emitted.

15.18.2 Member Function Documentation

15.18.2.1 activeACDataReady

```
void AisInstrumentHandler::activeACDataReady (
            uint8_t channel,
            const AisACData & ACData )  [signal]
```

a signal that is emitted whenever new AC data for an active experiment are ready.

Parameters

channel	the channel number from which the AC data arrived.
ACData	the AC data that just arrived.

15.18.2.2 activeDCDataReady

```
void AisInstrumentHandler::activeDCDataReady (
            uint8_t channel,
            const AisDCData & DCData )  [signal]
```

a signal that is emitted whenever new DC data for an active experiment are ready.

Parameters

channel	the channel number from which the DC data arrived.
DCData	the DC data that just arrived.

15.18.2.3 deviceDisconnected

```
void AisInstrumentHandler::deviceDisconnected ( )  [signal]
```

a signal that is emitted if the device associated with this handler has been disconnected.

15.18.2.4 eraseRecoverData()

```
AisErrorCode AisInstrumentHandler::eraseRecoverData ( ) const
```

delete the recover data from device.

Returns

AisErrorCode::Success if request is sucessfully send for delete the data. If not successful, possible returned errors are:

- AisErrorCode::DeviceNotFound
- AisErrorCode::DeviceCommunicationFailed

15.18.2.5 experimentNewElementStarting

```
void AisInstrumentHandler::experimentNewElementStarting (
            uint8_t channel,
            const AisExperimentNode & stepInfo )  [signal]
```

a signal that is emitted whenever a new elemental experiment has started.

Parameters

channel	the channel number on which the experiment was started.
stepInfo	information regarding the current step.

See also

AisExperimentNode

15.18.2.6 experimentPaused

```
void AisInstrumentHandler::experimentPaused (
            uint8_t channel )  [signal]
```

a signal that is emitted whenever an experiment was paused.

Parameters

channel	the channel on which the experiment was paused.

15.18.2.7 experimentResumed

```
void AisInstrumentHandler::experimentResumed (
            uint8_t channel )  [signal]
```

a signal that is emitted whenever an experiment was resumed.

Parameters

channel	the channel on which the experiment was resumed.

15.18.2.8 experimentStopped

```
void AisInstrumentHandler::experimentStopped (
            uint8_t channel )  [signal]
```

a signal that is emitted whenever an experiment was stopped manually or has completed.

Parameters

channel	the channel on which the experiment has stopped.

15.18.2.9 getExperimentUTCStartTime()

```
double AisInstrumentHandler::getExperimentUTCStartTime (
            uint8_t channel ) const
```

get UTC time for the start of the experiment in seconds.

This will give the time in seconds between the origin of UTC time and the start of the experiment aka Unix Epoch.

Parameters

channel	the channel for which to get the start time of the experiment.

Returns

the Unix Epoch up to the start of the experiment in seconds.

15.18.2.10 getFreeChannels()

```
std::vector< uint8_t > AisInstrumentHandler::getFreeChannels ( ) const
```

get a list of the currently free channels.

Returns

a list of the currently free channels. If all channels are busy, an empty list is returned.

15.18.2.11 getLinkedChannels()

```
std::vector< uint8_t > AisInstrumentHandler::getLinkedChannels (
            uint8_t channel ) const
```

get a list of channels linked to the given channel.

Parameters

channel	a valid channel number to find which other channels are linked to it.

Returns

a list of channels linked to the channel parameter.

15.18.2.12 getManualModeCurrentRangeList()

```
std::vector< std::pair< double, double > > AisInstrumentHandler::getManualModeCurrentRange↩
List (
            uint8_t channel ) const
```

get a list of the applicable current ranges to the given channel specific to your device.

The list is indexed, with each index containing a range with minimum and maximum current for the range. You can pass the index of the desired current range to setManualModeConstantVoltage.

Parameters

channel	a valid channel number for which to check the current range.

Returns

a list of the of the applicable current ranges to the given channel specific to your device.

15.18.2.13 getNumberOfChannels()

```
int AisInstrumentHandler::getNumberOfChannels ( ) const
```

get the number of all the channels on this device.

Returns

the number of channels on the connected device. If no device found, -1 will be returned.

15.18.2.14 groundFloatStateChanged

```
void AisInstrumentHandler::groundFloatStateChanged (
             bool grounded )  [signal]
```

a signal that is emitted when the floating ground connection state has changed.

Parameters

grounded	true if there is a connection to ground and false if the ground has disconnected.

15.18.2.15 hasBipolarMode()

```
bool AisInstrumentHandler::hasBipolarMode (
             uint8_t channel ) const
```

tells whether the given channel is bipolar mode

Parameters

channel	the channel number to check if it is bipolar mode

Returns

true only if given a valid channel number that has bipolar mode.

15.18.2.16 idleDCDataReady

```
void AisInstrumentHandler::idleDCDataReady (
            uint8_t channel,
            const AisDCData & DCData )  [signal]
```

a signal that is emitted whenever new DC data are ready when the device is in an idle state.

A manual experiment displays real time values. These values are displayed even if the channel does not have an experiment running on it.

Parameters

channel	the channel number from which the DC data arrived.
DCData	the DC data that just arrived.

15.18.2.17 isChannelBusy()

```
bool AisInstrumentHandler::isChannelBusy (
            uint8_t channel ) const
```

tells whether the given channel is busy or not.

Parameters

channel	the channel number to check if it is busy or not.

Returns

true only if given a valid channel number that has either a running or a paused experiment.

15.18.2.18 isChannelPaused()

```
bool AisInstrumentHandler::isChannelPaused (
            uint8_t channel ) const
```

tells whether the given channel has a paused experiment or not.

Parameters

channel	the channel number to check if it has a paused experiment.

Returns

true only if given a valid channel number that has an experiment that has been paused.

15.18.2.19 pauseExperiment()

```
AisErrorCode AisInstrumentHandler::pauseExperiment (
             uint8_t channel ) const
```

pause a running experiment on the channel.

Parameters

channel	the channel number to pause the experiment on.

Returns

true if an experiment was successfully paused on the channel and false otherwise. If not successful, possible returned errors are:

- AisErrorCode::FailedToPauseExperiment
- AisErrorCode::DeviceNotFound
- AisErrorCode::InvalidChannel
- AisErrorCode::ChannelNotBusy

This will return AisErrorCode::Success only if there is currently a running experiment on a valid channel on a connected device.

15.18.2.20 recoverDataErased

```
void AisInstrumentHandler::recoverDataErased (
             bool successful )  [signal]
```

a signal that is emitted whenever data erase process is completed.

Parameters

successful	is true on erased correctly, and false on data is not erased.

15.18.2.21 recoveryACDataReady

```
void AisInstrumentHandler::recoveryACDataReady (
             uint8_t channel,
             const AisACData & ACData )  [signal]
```

a signal that is emitted whenever new AC recovery data are ready.

Parameters

channel	the channel number from which the AC data are recovered from.
ACData	the AC data that just arrived.

15.18.2.22 recoveryDCDataReady

```
void AisInstrumentHandler::recoveryDCDataReady (
             uint8_t channel,
             const AisDCData & DCData )  [signal]
```

a signal that is emitted whenever new DC recovery data are ready.

Parameters

channel	the channel number from which the DC data are recovered from.
DCData	the DC data that just arrived.

15.18.2.23 resumeExperiment()

```
AisErrorCode AisInstrumentHandler::resumeExperiment (
             uint8_t channel ) const
```

resume a paused experiment on the channel.

Parameters

channel	the channel number to resume the experiment on.

Returns

AisErrorCode::Success if an experiment was successfully resumed on the channel. If not successful, possible returned errors are:

- AisErrorCode::FailedToResumeExperiment
- AisErrorCode::DeviceNotFound
- AisErrorCode::InvalidChannel
- AisErrorCode::ChannelNotBusy

This will return AisErrorCode::Success only if there is currently a paused experiment on a valid channel on a connected device.

15.18.2.24 setBipolarLinkedChannels()

```
int8_t AisInstrumentHandler::setBipolarLinkedChannels (
            std::vector< uint8_t > channels ) const
```

connect two channels together in bipolar mode.

You may combine two channels to expand the voltage range to include negative voltages. Note that this is only applicable to the cycler model. For 4 channel Cycler models, you can combine channels 1 and 2 or channels 3 and 4. You cannot use any other channel combinations.

Parameters

channels	a list of two channels to be oprate in bipolar mode.

Returns

the master channel out of the given list of two channels. The master channel is your interface to upload an experiment to and then control it. If not successful set in bipolar mode, possible returned errors as -1.

Note

this functionality is only applicable to the cycler model.

15.18.2.25 setCompRange()

```
AisErrorCode AisInstrumentHandler::setCompRange (
            uint8_t channel,
            const AisCompRange & compRange ) const
```

set a compensation range with stability factor and bandwidth index.

Parameters

channel	the channel for which to set the compensation range.
compRange	an object of type compRange that is initialized with a stability factor (0-10) and a bandwidth index (0-10).

Returns

AisErrorCode::Success if setting the IR compensation was successful. If not successful, possible returned errors are:

- AisErrorCode::DeviceNotFound
- AisErrorCode::InvalidChannel
- AisErrorCode::InvalidParameters

See also

AisCompRange

15.18.2.26 setIRComp()

```
AisErrorCode AisInstrumentHandler::setIRComp (
            uint8_t channel,
            double uncompensatedResistance,
            double compensationLevel ) const
```

set IR compensation.

Parameters

channel	the channel for which to set the IR compensation.
uncompensatedResistance	the value of the uncompensated resistance in Ohms.
compensationLevel	the compensation percentage (0%-100%). This is unit-less.

Returns

AisErrorCode::Success if setting the IR compensation was successful. If not successful, possible returned errors are:

- AisErrorCode::DeviceNotFound
- AisErrorCode::InvalidChannel
- AisErrorCode::InvalidParameters

15.18.2.27 setLinkedChannels()

```
int8_t AisInstrumentHandler::setLinkedChannels (
            std::vector< uint8_t > channels ) const
```

connect several channels together in parallel mode.

You may connect a list of channels so you can get a higher combined output current of all channels. Note that this is only applicable to the cycler model.

Parameters

channels	a list of channels to be linked.

Returns

the master channel out of the given list of channels. The master channel is your interface to upload an experiment to and then control it.

Note

this functionality is only applicable to the cycler model.

15.18.2.28 setManualModeConstantCurrent()

```
AisErrorCode AisInstrumentHandler::setManualModeConstantCurrent (
             uint8_t channel,
             double value ) const
```

set constant current for the manual experiment.

Parameters

channel	a valid channel number to set a constant voltage for.
value	the value to set the constant current in Amps.

Returns

AisErrorCode::Success if setting the constant current was successful. If not successful, possible returned errors are:

- AisErrorCode::ManualExperimentNotRunning
- AisErrorCode::DeviceNotFound
- AisErrorCode::InvalidChannel
- AisErrorCode::DeviceCommunicationFailed

15.18.2.29 setManualModeConstantVoltage() [1/2]

```
AisErrorCode AisInstrumentHandler::setManualModeConstantVoltage (
             uint8_t channel,
             double value ) const
```

set constant voltage for the manual experiment.

Parameters

channel	a valid channel number to set a constant voltage for.
value	the value to set the constant voltage in volts.

Returns

AisErrorCode::Success if setting the constant voltage was successful. If not successful, possible returned errors are:

- AisErrorCode::FailedToSetManualModeConstantVoltage

- AisErrorCode::ManualExperimentNotRunning
- AisErrorCode::DeviceNotFound
- AisErrorCode::InvalidChannel

15.18.2.30 setManualModeConstantVoltage() [2/2]

```
AisErrorCode AisInstrumentHandler::setManualModeConstantVoltage (
            uint8_t channel,
            double value,
            int currentRangeIndex ) const
```

set constant voltage for the manual experiment and also set a manual current range.

Parameters

channel	a valid channel number to set a constant voltage for.
value	the value to set the constant voltage in volts.
currentRangeIndex	the index of the desired current range.

Returns

> AisErrorCode::Success if setting the constant voltage was successful. You can get a list of the available ranges for your model by calling getManualModeCurrentRangeList. If not successful, possible returned errors are:
>
> - AisErrorCode::FailedToSetManualModeConstantVoltage
> - AisErrorCode::FailedToSetManualModeCurrentRange
> - AisErrorCode::ManualExperimentNotRunning
> - AisErrorCode::DeviceNotFound
> - AisErrorCode::InvalidChannel

15.18.2.31 setManualModeOCP()

```
AisErrorCode AisInstrumentHandler::setManualModeOCP (
            uint8_t channel ) const
```

set open-circuit potential mode.

To apply the set potential or current, leave the open circuit potential mode off. This operation is reversed automatically when calling either setManualModeConstantVoltage() or setManualModeConstantCurrent()

Parameters

channel	a valid channel number to set open circuit mode on.

Returns

AisErrorCode::Success if turning on the open circuit mode was successful. If not successful, possible returned errors are:

- AisErrorCode::ManualExperimentNotRunning
- AisErrorCode::DeviceNotFound
- AisErrorCode::InvalidChannel
- AisErrorCode::DeviceCommunicationFailed

15.18.2.32 setManualModeSamplingInterval()

```
AisErrorCode AisInstrumentHandler::setManualModeSamplingInterval (
             uint8_t channel,
             double value ) const
```

set an interval for sampling the data.

Parameters

channel	the channel to set the sampling interval for.
value	the value for the sampling interval in seconds.

Returns

AisErrorCode::Success if the operation was set successfully. If not successful, possible returned errors are:

- AisErrorCode::DeviceNotFound
- AisErrorCode::Unknown
- AisErrorCode::InvalidChannel

15.18.2.33 skipExperimentStep()

```
AisErrorCode AisInstrumentHandler::skipExperimentStep (
             uint8_t channel ) const
```

skip the current experiment step and proceed to the next.

When running an element that has several steps like going from CC to CV, then skipping the step goes to the next step within the element. When having several elements in the custom experiment and the current element has one step or we are at the last step within the element, then skipping the step results in going to the next element. If this is the final step of the final element, the experiment will stop.

Parameters

channel	a valid channel number with an experiment to skip the step.

Returns

AisErrorCode::Success the experiment step was successfully skipped If not successful, possible returned errors are:

- AisErrorCode::DeviceNotFound
- AisErrorCode::InvalidChannel
- AisErrorCode::ChannelNotBusy
- AisErrorCode::DeviceCommunicationFailed

15.18.2.34 startManualExperiment()

```
AisErrorCode AisInstrumentHandler::startManualExperiment (
              uint8_t channel ) const
```

start a manual experiment.

With manual experiments, users can turn on any connected channel and toggle between open circuit mode and voltage or current setpoints that can be changed in real-time and run for indefinite periods.

Parameters

channel	a valid channel number to run the manual experiment on.

Returns

AisErrorCode::Success if the manual experiment was successfully started. If not successful, possible returned errors are:

- AisErrorCode::FailedManualModeStartExperiment
- AisErrorCode::DeviceNotFound
- AisErrorCode::InvalidChannel
- AisErrorCode::BusyChannel

15.18.2.35 startUploadedExperiment()

```
AisErrorCode AisInstrumentHandler::startUploadedExperiment (
              uint8_t channel ) const
```

start the previously uploaded experiment on the specific channel.

Parameters

channel	the channel number to start the experiment on.

Returns

AisErrorCode::Success if the experiment was successfully started on the channel. If not successful, possible returned errors are:

- AisErrorCode::DeviceCommunicationFailed
- AisErrorCode::ExperimentNotUploaded
- AisErrorCode::DeviceNotFound
- AisErrorCode::InvalidChannel
- AisErrorCode::BusyChannel

See also

uploadExperimentToChannel

isChannelBusy

15.18.2.36 stopExperiment()

```
AisErrorCode AisInstrumentHandler::stopExperiment (
                uint8_t channel ) const
```

stop a running or a paused experiment on the channel.

Parameters

channel	the channel number to stop the experiment on.

Returns

AisErrorCode::Success if an experiment was successfully stopped on the channel. If not successful, possible returned errors are:

- AisErrorCode::FailedToStopExperiment
- AisErrorCode::DeviceNotFound
- AisErrorCode::InvalidChannel

This will only return AisErrorCode::Success if there is currently a running or a paused experiment on a valid channel on a connected device.

15.18.2.37 uploadExperimentToChannel() [1/2]

```
AisErrorCode AisInstrumentHandler::uploadExperimentToChannel (
                uint8_t channel,
                const AisExperiment & experiment ) const
```

upload an already created custom experiment to a specific channel on the device.

Any running experiment is run on a specific device on a specific channel. This function uploads an experiment to a channel so that you may start, pause, resume and stop the experiment. All of these four control functionalities and others require a channel number to control the experiment. Therefore, if we have several channels, we need to keep track of which experiment is on which channel.

Parameters

channel	the channel number to upload the experiment to.
experiment	the custom experiment to be uploaded to the channel.

Returns

AisErrorCode::Success if the experiment was successfully uploaded to the channel. If not successful, possible returned errors are:

- AisErrorCode::FailedToUploadExperiment
- AisErrorCode::ExperimentIsEmpty
- AisErrorCode::DeviceNotFound
- AisErrorCode::InvalidChannel
- AisErrorCode::BusyChannel
- AisErrorCode::InvalidParameters

This returns AisErrorCode::Success only when given a valid channel number that is not busy on a connected device.

See also

isChannelBusy

15.18.2.38 uploadExperimentToChannel() [2/2]

```
AisErrorCode AisInstrumentHandler::uploadExperimentToChannel (
            uint8_t channel,
            std::shared_ptr< AisExperiment > experiment ) const
```

upload an already created custom experiment to a specific channel on the device.

Any running experiment is run on a specific device on a specific channel. This function uploads an experiment to a channel so that you may start, pause, resume and stop the experiment. All of these four control functionalities and others require a channel number to control the experiment. Therefore, if we have several channels, we need to keep track of which experiment is on which channel.

Parameters

channel	the channel number to upload the experiment to.
experiment	the custom experiment to be uploaded to the channel.

Returns

AisErrorCode::Success if the experiment was successfully uploaded to the channel. If not successful, possible returned errors are:

- AisErrorCode::FailedToUploadExperiment
- AisErrorCode::ExperimentIsEmpty

- AisErrorCode::DeviceNotFound
- AisErrorCode::InvalidChannel
- AisErrorCode::BusyChannel
- AisErrorCode::InvalidParameters

This returns AisErrorCode::Success only when given a valid channel number that is not busy on a connected device.

See also

isChannelBusy

The documentation for this class was generated from the following file:

- AisInstrumentHandler.h

15.19 AisNormalPulseVoltammetryElement Class Reference

This experiment holds the working electrode at a **baseline potential** during the **quiet time**, then applies a train of pulses, which increase in amplitude until the **final potential** is reached.

```
#include <AisNormalPulseVoltammetryElement.h>
```

Inherits AisAbstractElement.

Public Member Functions

- AisNormalPulseVoltammetryElement (double startVoltage, double endVoltage, double voltageStep, double pulseWidth, double pulsePeriod)

  the normal-pulse-voltammetry element constructor
- **AisNormalPulseVoltammetryElement** (const AisNormalPulseVoltammetryElement &)

  copy constructor for the AisNormalPulseVoltammetryElement object.
- AisNormalPulseVoltammetryElement & **operator=** (const AisNormalPulseVoltammetryElement &)

  overload equal to operator for the AisNormalPulseVoltammetryElement object.
- QString getName () const override

  get the name of the element.
- QStringList getCategory () const override

  get a list of applicable categories of the element.
- double getStartVoltage () const

  get the value set for the start voltage.
- void setStartVoltage (double startVoltage)

  set the value for the start voltage.
- bool isStartVoltageVsOCP () const

  tells whether the start voltage is set against the open-circuit voltage or the reference terminal.
- void setStartVoltageVsOCP (bool startVoltageVsOCP)

  set whether to reference the start voltage against the open-circuit voltage or the reference terminal.
- double getEndVoltage () const

  get the value set for the ending potential value.
- void setEndVoltage (double endVoltage)

set the ending potential value.

- bool isEndVoltageVsOCP () const

  tells whether the end voltage is set with respect to the open circuit voltage or the reference terminal.
- void setEndVoltageVsOCP (bool endVoltageVsOcp)

  set whether to reference the end voltage against the open-circuit voltage or the reference terminal.
- double getVStep () const

  get the value set for the voltage step.
- void setVStep (double vStep)

  set the value for the voltage step.
- double getPulseWidth () const

  get the value for the voltage pulse width.
- void setPulseWidth (double pulseWidth)

  set the value for the voltage pulse width.
- double getPulsePeriod () const

  get the value set for the pulse period.
- void setPulsePeriod (double pulsePeriod)

  set the value for the pulse period.
- bool isAutoRange () const

  tells whether the current range is set to auto-select or not.
- void setAutoRange ()

  set to auto-select the current range.
- double getApproxMaxCurrent () const

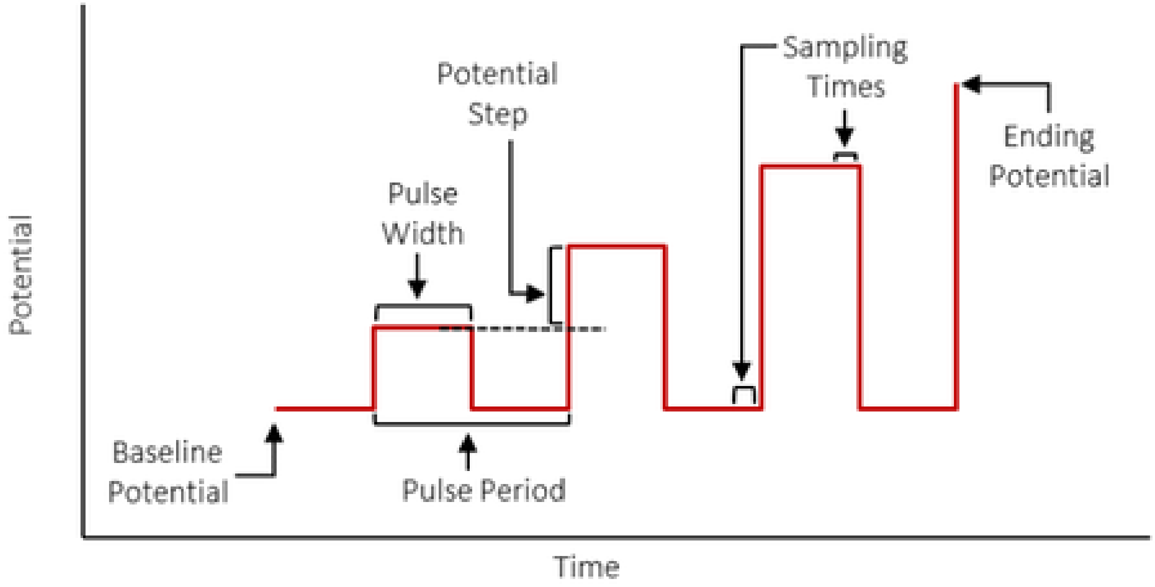
  get the value set for the expected maximum current.
- void setApproxMaxCurrent (double approxMaxCurrent)

  set maximum current expected, for manual current range selection.

15.19.1 Detailed Description

This experiment holds the working electrode at a **baseline potential** during the **quiet time**, then applies a train of pulses, which increase in amplitude until the **final potential** is reached.

The **potential step** is the magnitude of this incremental increase. The **pulse width** is the amount of time between the rising and falling edge of a pulse. The **pulse period** is the amount of time between the beginning of one pulse and the beginning of the next.



15.19.2 Constructor & Destructor Documentation

15.19.2.1 AisNormalPulseVoltammetryElement()

```
AisNormalPulseVoltammetryElement::AisNormalPulseVoltammetryElement (
            double startVoltage,
            double endVoltage,
            double voltageStep,
            double pulseWidth,
            double pulsePeriod )  [explicit]
```

the normal-pulse-voltammetry element constructor

Parameters

startVoltage	the value of the starting potential in volts
endVoltage	the value of the ending potential in volts
voltageStep	the value set for the voltage step in volts.
pulseWidth	the value for the pulse width in volts.
pulsePeriod	the value for the pulse period in volts.

15.19.3 Member Function Documentation

15.19.3.1 getApproxMaxCurrent()

```
double AisNormalPulseVoltammetryElement::getApproxMaxCurrent ( ) const
```

get the value set for the expected maximum current.

Returns

the value set for the expected maximum current in Amps.

Note

if nothing was manually set, the device will auto-select the current range and the return value will be positive infinity.

15.19.3.2 getCategory()

```
QStringList AisNormalPulseVoltammetryElement::getCategory ( ) const  [override]
```

get a list of applicable categories of the element.

Returns

A list of applicable categories: ("Potentiostatic Control", "Basic Voltammetry", "Pulse Voltammetry").

15.19.3.3 getEndVoltage()

```
double AisNormalPulseVoltammetryElement::getEndVoltage ( ) const
```

get the value set for the ending potential value.

This is the value of the voltage at which the experiment will stop.

Returns

the value set for the ending voltage in volts.

15.19.3.4 getName()

```
QString AisNormalPulseVoltammetryElement::getName ( ) const  [override]
```

get the name of the element.

Returns

The name of the element: "Normal Pulse Potential Voltammetry".

15.19.3.5 getPulsePeriod()

```
double AisNormalPulseVoltammetryElement::getPulsePeriod ( ) const
```

get the value set for the pulse period.

Returns

the value set for the pulse period in seconds.

See also

setPulsePeriod

15.19.3.6 getPulseWidth()

```
double AisNormalPulseVoltammetryElement::getPulseWidth ( ) const
```

get the value for the voltage pulse width.

Returns

the value for the voltage pulse width in seconds.

See also

setPulseWidth

15.19.3.7 getStartVoltage()

```
double AisNormalPulseVoltammetryElement::getStartVoltage ( ) const
```

get the value set for the start voltage.

Returns

the value of the start voltage in volts.

15.19.3.8 getVStep()

```
double AisNormalPulseVoltammetryElement::getVStep ( ) const
```

get the value set for the voltage step.

Returns

the value set for the voltage step in volts.

See also

setVStep

15.19.3.9 isAutoRange()

```
bool AisNormalPulseVoltammetryElement::isAutoRange ( ) const
```

tells whether the current range is set to auto-select or not.

Returns

true if the current range is set to auto-select and false if a rage has been selected.

15.19.3.10 isEndVoltageVsOCP()

```
bool AisNormalPulseVoltammetryElement::isEndVoltageVsOCP ( ) const
```

tells whether the end voltage is set with respect to the open circuit voltage or the reference terminal.

Returns

true if the end voltage is set with respect to the open-circuit voltage and false if set against the reference terminal.

Note

if nothing is set, the default is false.

15.19.3.11 isStartVoltageVsOCP()

```
bool AisNormalPulseVoltammetryElement::isStartVoltageVsOCP ( ) const
```

tells whether the start voltage is set against the open-circuit voltage or the reference terminal.

Returns

true if the start voltage is set against the open-circuit voltage and false if it is set against the reference terminal.

Note

if nothing is set, the default is false.

See also

setStartVoltageVsOCP

15.19.3.12 setApproxMaxCurrent()

```
void AisNormalPulseVoltammetryElement::setApproxMaxCurrent (
             double approxMaxCurrent )
```

set maximum current expected, for manual current range selection.

The is an **optional** parameter. If nothing is set, the device will auto-select the current range.

Parameters

approxMaxCurrent	the value for the maximum current expected in Amps.

15.19.3.13 setAutoRange()

```
void AisNormalPulseVoltammetryElement::setAutoRange ( )
```

set to auto-select the current range.

This option is set by default. There is no need to call this function to auto-select if the range was not manually set.

15.19.3.14 setEndVoltage()

```
void AisNormalPulseVoltammetryElement::setEndVoltage (
             double endVoltage )
```

set the ending potential value.

This is the value of the voltage at which the experiment will stop.

Parameters

endVoltage	the value to set for the ending potential in volts.

15.19.3.15 setEndVoltageVsOCP()

```
void AisNormalPulseVoltammetryElement::setEndVoltageVsOCP (
            bool endVoltageVsOcp )
```

set whether to reference the end voltage against the open-circuit voltage or the reference terminal.

The reference terminal is for you to connect to any reference point you like. Connect it to the working electrode to reference ground.

Parameters

endVoltageVsOcp	true to set the end voltage to be referenced against the open-circuit voltage and false if set against the reference terminal.

Note

by default, this is set to false.

15.19.3.16 setPulsePeriod()

```
void AisNormalPulseVoltammetryElement::setPulsePeriod (
            double pulsePeriod )
```

set the value for the pulse period.

The pulse period is the time spent between the starts of two consecutive pulses.

Parameters

pulsePeriod	the value to set for the pulse period in seconds.

15.19.3.17 setPulseWidth()

```
void AisNormalPulseVoltammetryElement::setPulseWidth (
            double pulseWidth )
```

set the value for the voltage pulse width.

The pulse width is the value in seconds for the time spent at the same voltage set for the pulse height.

Parameters

pulseWidth	the value to set for the pulse width in seconds.

15.19.3.18 setStartVoltage()

```
void AisNormalPulseVoltammetryElement::setStartVoltage (
            double startVoltage )
```

set the value for the start voltage.

Parameters

startVoltage	the value of the start voltage in volts

15.19.3.19 setStartVoltageVsOCP()

```
void AisNormalPulseVoltammetryElement::setStartVoltageVsOCP (
            bool startVoltageVsOCP )
```

set whether to reference the start voltage against the open-circuit voltage or the reference terminal.

The reference terminal is for you to connect to any reference point you like. Connect it to the working electrode to reference ground.

Parameters

startVoltageVsOCP	true to if the start voltage is set to reference the open-circuit voltage and false if set against the reference terminal.

Note

by default, this is set to false.

15.19.3.20 setVStep()

```
void AisNormalPulseVoltammetryElement::setVStep (
            double vStep )
```

set the value for the voltage step.

The voltage step is the voltage difference between the heights of two consecutive pulses.

Parameters

vStep	the value for the voltage step in volts.

The documentation for this class was generated from the following file:

- AisNormalPulseVoltammetryElement.h

15.20 AisOpenCircuitElement Class Reference

This experiment observes the **open circuit potential** of the working electrode for a specific period of time.

```
#include <AisOpenCircuitElement.h>
```

Inherits AisAbstractElement.

Public Member Functions

- AisOpenCircuitElement (double duration, double samplingInterval)

  the open-circuit element constructor.
- **AisOpenCircuitElement** (const AisOpenCircuitElement &)

  copy constructor for the AisOpenCircuitElement object.
- AisOpenCircuitElement & **operator=** (const AisOpenCircuitElement &)

  overload equal to operator for the AisOpenCircuitElement object.
- QString getName () const override

  get the name of the element.
- QStringList getCategory () const override

  get a list of applicable categories of the element.
- double getSamplingInterval () const

  get how frequently we are sampling the data.
- void setSamplingInterval (double samplingInterval)

  set how frequently we are sampling the data.
- double getMaxDuration () const

  get the value set for the duration of the experiment.
- void setMaxDuration (double maxDuration)

  set the value set for the duration of the experiment.
- double getMaxVoltage () const

  get the value set for the maximum voltage. The experiment will end when it reaches this value.
- void setMaxVoltage (double maxVoltage)

  set a maximum voltage to stop the experiment.
- double getMinVoltage () const

  get the value set minimum for the voltage in volts.
- void setMinVoltage (double minVoltage)

  set a minimum voltage to stop the experiment.
- double getMindVdt () const

  get the value set for the minimum voltage rate of change with respect to time (minimum dV/dt).
- void setMindVdt (double mindVdt)

set the minimum value for the voltage rate of change with respect to time (minimum dV/dt).

- bool isAutoVoltageRange () const

  tells whether the voltage range is set to auto-select or not.
- void setAutoVoltageRange ()

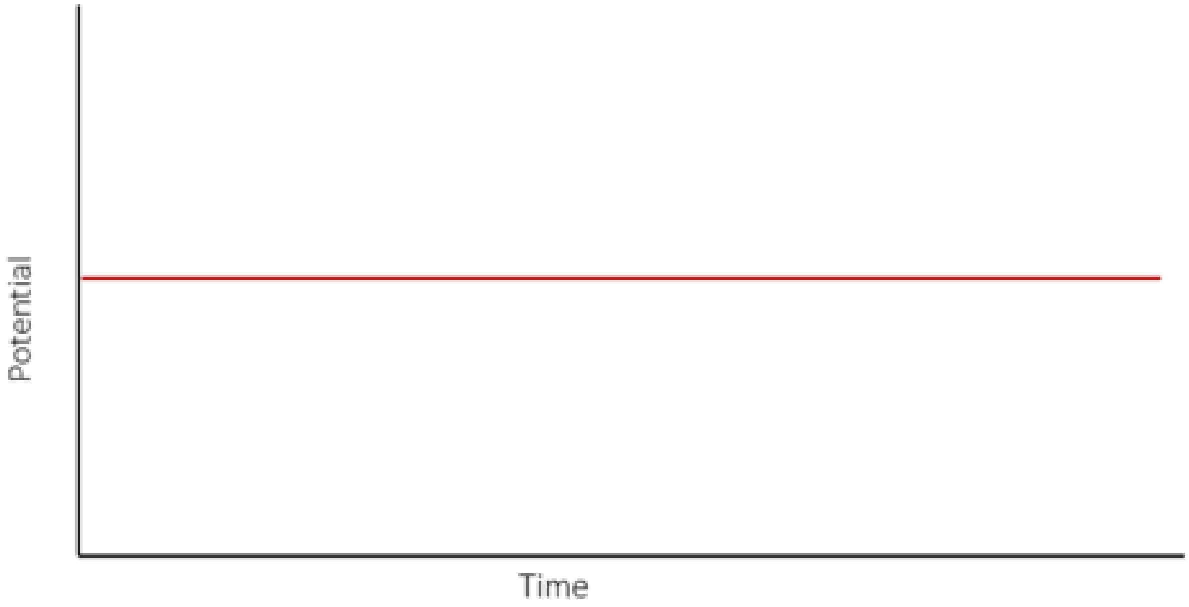
  set to auto-select the voltage range.
- double getApproxMaxVoltage () const

  get the value set for the expected maximum voltage.
- void setApproxMaxVoltage (double approxMaxVoltage)

  set maximum voltage expected, for manual voltage range selection.

15.20.1 Detailed Description

This experiment observes the **open circuit potential** of the working electrode for a specific period of time.



15.20.2 Constructor & Destructor Documentation

15.20.2.1 AisOpenCircuitElement()

```
AisOpenCircuitElement::AisOpenCircuitElement (
             double duration,
             double samplingInterval )  [explicit]
```

the open-circuit element constructor.

Parameters

duration	the maximum duration for the experiment in seconds.
samplingInterval	the data sampling interval value in seconds.

15.20.3 Member Function Documentation

15.20.3.1 getApproxMaxVoltage()

```
double AisOpenCircuitElement::getApproxMaxVoltage ( ) const
```

get the value set for the expected maximum voltage.

Returns

the value set for the expected maximum Voltage in volt.

Note

if nothing was manually set, the device will auto-select the voltage range and the return value will be positive infinity.

15.20.3.2 getCategory()

```
QStringList AisOpenCircuitElement::getCategory ( ) const  [override]
```

get a list of applicable categories of the element.

Returns

A list of applicable categories: ("Basic Experiments").

15.20.3.3 getMaxDuration()

```
double AisOpenCircuitElement::getMaxDuration ( ) const
```

get the value set for the duration of the experiment.

Returns

the value set for the duration of the experiment in seconds.

15.20.3.4 getMaxVoltage()

```
double AisOpenCircuitElement::getMaxVoltage ( ) const
```

get the value set for the maximum voltage. The experiment will end when it reaches this value.

Returns

the value set for the maximum voltage.

Note

this is an optional parameter. If no value has been set, the default value is positive infinity.

15.20.3.5 getMindVdt()

```
double AisOpenCircuitElement::getMindVdt ( ) const
```

get the value set for the minimum voltage rate of change with respect to time (minimum dV/dt).

Returns

the value set for the minimum voltage rate of change with respect to time (minimum dV/dt).

Note

this is an optional parameter. If no value has been set, the default value is zero

15.20.3.6 getMinVoltage()

```
double AisOpenCircuitElement::getMinVoltage ( ) const
```

get the value set minimum for the voltage in volts.

Returns

the value set for the minimum voltage in volts.

Note

this is an optional parameter. If no value has been set, the default value is negative infinity.

15.20.3.7 getName()

```
QString AisOpenCircuitElement::getName ( ) const  [override]
```

get the name of the element.

Returns

The name of the element: "Open Circuit Potential".

15.20.3.8 getSamplingInterval()

```
double AisOpenCircuitElement::getSamplingInterval ( ) const
```

get how frequently we are sampling the data.

Returns

the data sampling interval value in seconds.

15.20.3.9 isAutoVoltageRange()

```
bool AisOpenCircuitElement::isAutoVoltageRange ( ) const
```

tells whether the voltage range is set to auto-select or not.

Returns

true if the voltage range is set to auto-select and false if a range has been selected.

15.20.3.10 setApproxMaxVoltage()

```
void AisOpenCircuitElement::setApproxMaxVoltage (
             double approxMaxVoltage )
```

set maximum voltage expected, for manual voltage range selection.

The is an **optional** parameter. If nothing is set, the device will auto-select the voltage range.

Parameters

approxMaxVoltage	the value for the maximum current expected in V.

15.20.3.11 setAutoVoltageRange()

```
void AisOpenCircuitElement::setAutoVoltageRange ( )
```

set to auto-select the voltage range.

This option is set by default. There is no need to call this function to auto-select if the range was not manually set.

15.20.3.12 setMaxDuration()

```
void AisOpenCircuitElement::setMaxDuration (
            double maxDuration )
```

set the value set for the duration of the experiment.

Parameters

maxDuration	the value to set for the duration of the experiment in seconds.

15.20.3.13 setMaxVoltage()

```
void AisOpenCircuitElement::setMaxVoltage (
            double maxVoltage )
```

set a maximum voltage to stop the experiment.

The is an **optional** condition. If nothing is set, then the experiment will not stop based on an upper-limit voltage value. If a maximum voltage is set, the experiment will continue to run as long as the measured voltage is below that value.

Parameters

maxVoltage	the maximum voltage value in volts at which the experiment will stop.

15.20.3.14 setMindVdt()

```
void AisOpenCircuitElement::setMindVdt (
            double mindVdt )
```

set the minimum value for the voltage rate of change with respect to time (minimum dV/dt).

The is an **optional** condition. If nothing is set, then the experiment will not stop based on an lower-limit rate of change value. If a minimum value is set, the experiment will continue to run as long as the rage of change is above that value.

Parameters

mindVdt	the minimum value for the voltage rate of change with respect to time (minimum dV/dt).

15.20.3.15 setMinVoltage()

```
void AisOpenCircuitElement::setMinVoltage (
            double minVoltage )
```

set a minimum voltage to stop the experiment.

The is an **optional** condition. If nothing is set, then the experiment will not stop based on an lower-limit voltage value. If a maximum voltage is set, the experiment will continue to run as long as the measured voltage is above that value.

Parameters

minVoltage	the minimum voltage value in volts at which the experiment will stop.

15.20.3.16 setSamplingInterval()

```
void AisOpenCircuitElement::setSamplingInterval (
            double samplingInterval )
```

set how frequently we are sampling the data.

Parameters

samplingInterval	the data sampling interval value in seconds.

The documentation for this class was generated from the following file:

- AisOpenCircuitElement.h

15.21 AisSquareWaveVoltammetryElement Class Reference

This experiment holds the working electrode at the **starting potential** during the **quiet time**. Then it applies a train of square pulses superimposed on a staircase waveform with a uniform **potential step** magnitude.

```
#include <AisSquareWaveVoltammetryElement.h>
```

Inherits AisAbstractElement.

Public Member Functions

- AisSquareWaveVoltammetryElement (double startVoltage, double endVoltage, double voltageStep, double pulseAmp, double pulseFrequency)

    the square wave element constructor
- **AisSquareWaveVoltammetryElement** (const AisSquareWaveVoltammetryElement &)

    copy constructor for the AisSquareWaveVoltammetryElement object.
- AisSquareWaveVoltammetryElement & **operator=** (const AisSquareWaveVoltammetryElement &)

    overload equal to operator for the AisSquareWaveVoltammetryElement object.
- QString getName () const override

    get the name of the element.
- QStringList getCategory () const override

    get a list of applicable categories of the element.
- double getStartVoltage () const

    get the value set for the start voltage.
- void setStartVoltage (double startVoltage)

    set the value for the start voltage.
- bool isStartVoltageVsOCP () const

    tells whether the start voltage is set against the open-circuit voltage or the reference terminal.
- void setStartVoltageVsOCP (bool startVoltageVsOcp)

    set whether to reference the start voltage against the open-circuit voltage or the reference terminal.
- double getEndVoltage () const

    get the value set for the ending potential value.
- void setEndVoltage (double endVoltage)

    set the ending potential value.
- bool isEndVoltageVsOCP () const

    tells whether the end voltage is set with respect to the open circuit voltage or the reference terminal.
- void setEndVoltageVsOCP (bool endVoltageVsOcp)

    set whether to reference the end voltage against the open-circuit voltage or the reference terminal.
- double getVStep () const

    get the value set for the voltage step.
- void setVStep (double vStep)

    set the value for the voltage step.
- double getPulseAmp () const

    get the value set for the pulse amplitude.
- void setPulseAmp (double pulseAmp)

    set the value for the pulse amplitude.
- double getPulseFreq () const

    get the value set for the pulse frequency.
- void setPulseFreq (double pulseFreq)

    set the value for the pulse frequency.
- bool isAutoRange () const

    tells whether the current range is set to auto-select or not.
- void setAutoRange ()

    set to auto-select the current range.
- double getApproxMaxCurrent () const

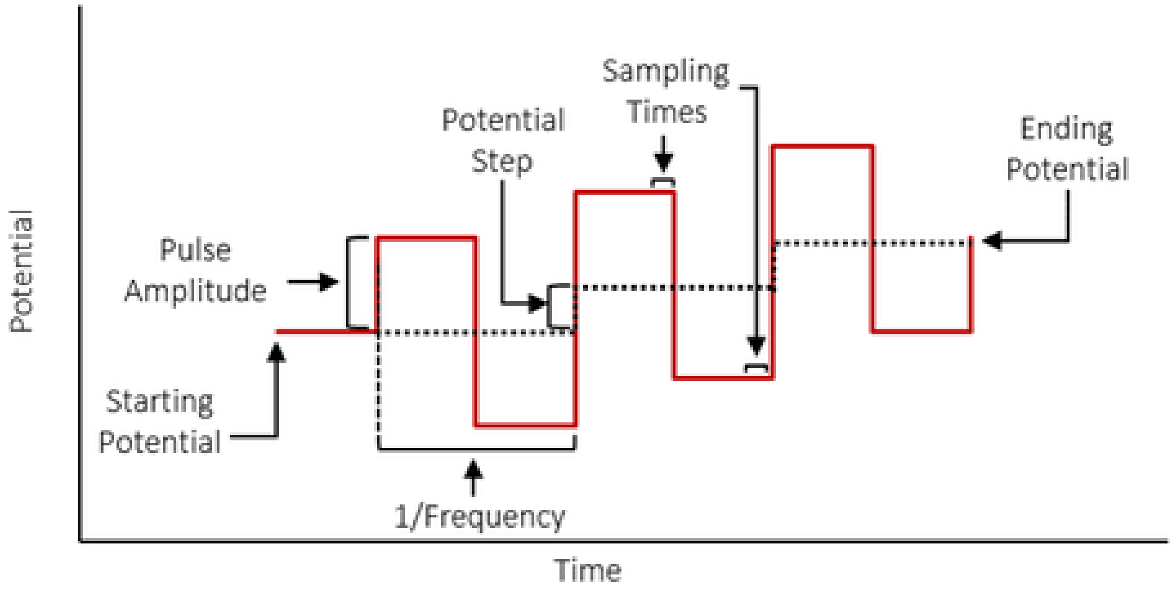
    get the value set for the expected maximum current.
- void setApproxMaxCurrent (double approxMaxCurrent)

    set maximum current expected, for manual current range selection.

15.21.1 Detailed Description

This experiment holds the working electrode at the **starting potential** during the **quiet time**. Then it applies a train of square pulses superimposed on a staircase waveform with a uniform **potential step** magnitude.

The potential continues to step until the **final potential** is reached. Each square pulse consists of a forward pulse and a reverse pulse of equal in **amplitude** but opposite in direction. **Frequency** is the inverse of the total duration of a square pulse. Current responses are sampled at two points, one at the end of the forward pulse (if) and another at the end of the reverse pulse (ir). The difference in current sampled at these two points is plotted against the potential of the corresponding staircase tread.



15.21.2 Constructor & Destructor Documentation

15.21.2.1 AisSquareWaveVoltammetryElement()

```
AisSquareWaveVoltammetryElement::AisSquareWaveVoltammetryElement (
            double startVoltage,
            double endVoltage,
            double voltageStep,
            double pulseAmp,
            double pulseFrequency )  [explicit]
```

the square wave element constructor

Parameters

startVoltage	the value of the starting potential in volts
endVoltage	the value of the ending potential in volts
voltageStep	the value set for the voltage step in volts.
pulseAmp	the value for the pulse amplitude in volts.
pulseFrequency	the value for the pulse frequency in Hz.

15.21.3 Member Function Documentation

15.21.3.1 getApproxMaxCurrent()

```
double AisSquareWaveVoltammetryElement::getApproxMaxCurrent ( ) const
```

get the value set for the expected maximum current.

Returns

the value set for the expected maximum current in Amps.

Note

if nothing was manually set, the device will auto-select the current range and the return value will be positive infinity.

15.21.3.2 getCategory()

```
QStringList AisSquareWaveVoltammetryElement::getCategory ( ) const  [override]
```

get a list of applicable categories of the element.

Returns

A list of applicable categories: ("Potentiostatic Control", "Pulse Voltammetry").

15.21.3.3 getEndVoltage()

```
double AisSquareWaveVoltammetryElement::getEndVoltage ( ) const
```

get the value set for the ending potential value.

This is the value of the voltage at which the experiment will stop.

Returns

the value set for the ending voltage in volts.

15.21.3.4 getName()

```
QString AisSquareWaveVoltammetryElement::getName ( ) const  [override]
```

get the name of the element.

Returns

The name of the element: "Square Wave Potential Voltammetry".

15.21.3.5 getPulseAmp()

```
double AisSquareWaveVoltammetryElement::getPulseAmp ( ) const
```

get the value set for the pulse amplitude.

Returns

the value set for the pulse amplitude in volts.

See also

setPulseAmp

15.21.3.6 getPulseFreq()

```
double AisSquareWaveVoltammetryElement::getPulseFreq ( ) const
```

get the value set for the pulse frequency.

Returns

the value set for the frequency in Hz.

15.21.3.7 getStartVoltage()

```
double AisSquareWaveVoltammetryElement::getStartVoltage ( ) const
```

get the value set for the start voltage.

Returns

the value of the start voltage in volts.

15.21.3.8 getVStep()

```
double AisSquareWaveVoltammetryElement::getVStep ( ) const
```

get the value set for the voltage step.

Returns

the value set for the voltage step in volts.

See also

setVStep

15.21.3.9 isAutoRange()

```
bool AisSquareWaveVoltammetryElement::isAutoRange ( ) const
```

tells whether the current range is set to auto-select or not.

Returns

true if the current range is set to auto-select and false if a rage has been selected.

15.21.3.10 isEndVoltageVsOCP()

```
bool AisSquareWaveVoltammetryElement::isEndVoltageVsOCP ( ) const
```

tells whether the end voltage is set with respect to the open circuit voltage or the reference terminal.

Returns

true if the end voltage is set with respect to the open-circuit voltage and false if set against the reference terminal.

Note

if nothing is set, the default is false.

15.21.3.11 isStartVoltageVsOCP()

```
bool AisSquareWaveVoltammetryElement::isStartVoltageVsOCP ( ) const
```

tells whether the start voltage is set against the open-circuit voltage or the reference terminal.

Returns

true if the start voltage is set against the open-circuit voltage and false if it is set against the reference terminal.

Note

if nothing is set, the default is false.

See also

setStartVoltageVsOCP

15.21.3.12 setApproxMaxCurrent()

```
void AisSquareWaveVoltammetryElement::setApproxMaxCurrent (
            double approxMaxCurrent )
```

set maximum current expected, for manual current range selection.

The is an **optional** parameter. If nothing is set, the device will auto-select the current range.

Parameters

approxMaxCurrent	the value for the maximum current expected in Amps.

15.21.3.13 setAutoRange()

```
void AisSquareWaveVoltammetryElement::setAutoRange ( )
```

set to auto-select the current range.

This option is set by default. There is no need to call this function to auto-select if the range was not manually set.

15.21.3.14 setEndVoltage()

```
void AisSquareWaveVoltammetryElement::setEndVoltage (
            double endVoltage )
```

set the ending potential value.

This is the value of the voltage at which the experiment will stop.

Parameters

endVoltage	the value to set for the ending potential in volts.

15.21.3.15 setEndVoltageVsOCP()

```
void AisSquareWaveVoltammetryElement::setEndVoltageVsOCP (
            bool endVoltageVsOcp )
```

set whether to reference the end voltage against the open-circuit voltage or the reference terminal.

The reference terminal is for you to connect to any reference point you like. Connect it to the working electrode to reference ground.

Parameters

endVoltageVsOcp	true to set the end voltage to be referenced against the open-circuit voltage and false if set against the reference terminal.

Note

by default, this is set to false.

15.21.3.16 setPulseAmp()

```
void AisSquareWaveVoltammetryElement::setPulseAmp (
            double pulseAmp )
```

set the value for the pulse amplitude.

The voltage pulse goes up in hight by the given amplitude in addition to the starting potential (of the previous pulse). It then goes back down twice the amplitude to end up one amplitude below the starting potential (of the previous pulse).

Parameters

pulseAmp	the value to set for the pulse amplitude in volts.

15.21.3.17 setPulseFreq()

```
void AisSquareWaveVoltammetryElement::setPulseFreq (
            double pulseFreq )
```

set the value for the pulse frequency.

Parameters

pulseFreq	the value to set for the pulse frequency in Hz.

15.21.3.18 setStartVoltage()

```
void AisSquareWaveVoltammetryElement::setStartVoltage (
            double startVoltage )
```

set the value for the start voltage.

Parameters

startVoltage	the value of the start voltage in volts

15.21.3.19 setStartVoltageVsOCP()

```
void AisSquareWaveVoltammetryElement::setStartVoltageVsOCP (
            bool startVoltageVsOcp )
```

set whether to reference the start voltage against the open-circuit voltage or the reference terminal.

The reference terminal is for you to connect to any reference point you like. Connect it to the working electrode to reference ground.

Parameters

startVoltageVsOcp	true to if the start voltage is set to reference the open-circuit voltage and false if set against the reference terminal.

Note

by default, this is set to false.

15.21.3.20 setVStep()

```
void AisSquareWaveVoltammetryElement::setVStep (
            double vStep )
```

set the value for the voltage step.

The voltage step is added to the value of the starting potential of the previous pulse to start the new pulse.

Parameters

vStep	the value for the voltage step in volts.

The documentation for this class was generated from the following file:

- AisSquareWaveVoltammetryElement.h

Chapter 16

File Documentation

16.1 AisCompRange.h

```
#ifndef SQUIDSTATLIBRARY_AISCOMPRANGE_H
#define SQUIDSTATLIBRARY_AISCOMPRANGE_H

#include "AisSquidstatGlobal.h"
#include <QString>
#include <memory>

class AisCompRangePrivate;

class SQUIDSTATLIBRARY_EXPORT AisCompRange final {
public:
    explicit AisCompRange(const QString& compRangeName, uint8_t bandwidthIndex, uint8_t stabilityFactor);

    AisCompRange(const AisCompRange&);


    uint8_t getBandwidthIndex() const;

    void setBandwidthIndex(uint8_t index);

    uint8_t getStabilityFactor() const;

    void setStabilityFactor(uint8_t factor);

    void setCompRangeName(const QString& compRangeName);

    const QString& getCompRangeName() const;

private:
    std::shared_ptr<AisCompRangePrivate> m_data;
};

#endif
```

16.2 AisDataPoints.h

```
#ifndef SQUIDSTATLIBRARY_AISDATAPOINTS_H
#define SQUIDSTATLIBRARY_AISDATAPOINTS_H

struct AisDCData {

    double timestamp;

    double workingElectrodeVoltage;

    double counterElectrodeVoltage;

    double current;

    double temperature;
};

struct AisACData {

    double timestamp;

    double frequency;

    double absoluteImpedance;

    double realImpedance;

    double imagImpedance;

    double phaseAngle;

    double totalHarmonicDistortion;

    double numberOfCycles;

    double workingElectrodeDCVoltage;

    double DCCurrent;

    double currentAmplitude;

    double voltageAmplitude;
};

struct AisExperimentNode {

    QString stepName;

    int stepNumber;

    int substepNumber;
};

#endif //SQUIDSTATLIBRARY_AISDATAPOINTS_H
```

16.3 AisDeviceTracker.h

```
#ifndef SQUIDSTATLIBRARY_AISDEVICETRACKER_H
#define SQUIDSTATLIBRARY_AISDEVICETRACKER_H

#include "AisErrorCode.h"
#include "AisSquidstatGlobal.h"
#include <QObject>
#include <memory>



class AisDeviceTrackerPrivate;
class AisInstrumentHandler;

class SQUIDSTATLIBRARY_EXPORT AisDeviceTracker final :  public QObject
{
    Q_OBJECT
public:
    ~AisDeviceTracker() override;
    static AisDeviceTracker *Instance();

    AisErrorCode connectToDeviceOnComPort(const QString &comPort);

    const AisInstrumentHandler &getInstrumentHandler(const QString &deviceName) const;

    const std::list<QString> getConnectedDevices() const;

    int connectAllPluggedInDevices();

    AisErrorCode updateFirmwareOnComPort(QString comport) const;

    int updateFirmwareOnAllAvailableDevices();

signals:
    void newDeviceConnected(const QString &deviceName);

    void deviceDisconnected(const QString &deviceName);

    void firmwareUpdateNotification(const QString& message);

private:
    AisDeviceTracker();
    AisDeviceTracker(const AisDeviceTracker &);
    void operator=(const AisDeviceTracker &);

    std::unique_ptr<AisDeviceTrackerPrivate> m_data;


};

#endif
```

16.4 AisErrorCode.h

```

#ifndef AIS_ERROR_CODE_H
#define AIS_ERROR_CODE_H

#include "AisSquidstatGlobal.h"
#include <qstring.h>

class SQUIDSTATLIBRARY_EXPORT AisErrorCode {

public:
    enum ErrorCode :  uint8_t {
        Unknown = 255,
        Success = 0,
        ConnectionFailed = 1,
        FirmwareNotSupported = 2,
        FirmwareFileNotFound = 3,
        FirmwareUptodate = 4,

        InvalidChannel = 10,
        BusyChannel = 11,
        DeviceNotFound = 13,
        ManualExperimentNotRunning = 51,
        ExperimentNotUploaded = 52,
        ExperimentIsEmpty = 53,
        InvalidParameters = 54,
        ChannelNotBusy = 55,
        DeviceCommunicationFailed = 100,
        FailedToSetManualModeCurrentRange = 101,
        FailedToSetManualModeConstantVoltage = 102,
        FailedToPauseExperiment = 103,
        FailedToResumeExperiment = 104,
        FailedToStopExperiment = 105,
        FailedToUploadExperiment = 106,
        ExperimentAlreadyPaused = 107,
        ExperimentAlreadyRun = 108,
    };

    AisErrorCode();
    AisErrorCode(ErrorCode error);
    AisErrorCode(ErrorCode error, QString message);

    QString message() const;

    int value() const;

    operator ErrorCode() const;

private:
    ErrorCode code;
    QString errorMessage;
};

#endif // !  AIS_ERROR_CODE_H
```

16.5 AisExperiment.h

```
#ifndef SQUIDSTATLIBRARY_AISCUSTOMEXPERIMENTRUNNER_H
#define SQUIDSTATLIBRARY_AISCUSTOMEXPERIMENTRUNNER_H

#include "AisSquidstatGlobal.h"
#include "experiments/builder_elements/AisAbstractElement.h"
#include <QString>

class CustomExperimentRunner;
class AisExperimentPrivate;

class SQUIDSTATLIBRARY_EXPORT AisExperiment final {
public:
    explicit AisExperiment();

    explicit AisExperiment(const AisExperiment& exp);

    void operator=(const AisExperiment& exp);

    ~AisExperiment();

    QString getExperimentName() const;

    QString getDescription() const;

    QStringList getCategory() const;

    void setExperimentName(QString name);

    void setDescription(QString description);

    bool appendElement(AisAbstractElement& element, uint repeat = 1);

    bool appendSubExperiment(const AisExperiment& subExp, uint repeat = 1);

private:
    friend class AisInstrumentHandler;
    std::shared_ptr<AisExperimentPrivate> m_data;
};

#endif //SQUIDSTATLIBRARY_AISCUSTOMEXPERIMENTRUNNER_H
```

16.6 AisInstrumentHandler.h

```
#ifndef SQUIDSTATLIBRARY_AISINSTRUMENTHANDLER_H
#define SQUIDSTATLIBRARY_AISINSTRUMENTHANDLER_H

#include <ctime>

#include <QObject>

#include "AisCompRange.h"
#include "AisDataPoints.h"
#include "AisErrorCode.h"
#include "AisSquidstatGlobal.h"

class AisInstrumentHandlerPrivate;
class AisExperiment;

class SQUIDSTATLIBRARY_EXPORT AisInstrumentHandler final :  public QObject {
    Q_OBJECT
    AisInstrumentHandlerPrivate* m_data;
public:
    explicit AisInstrumentHandler(AisInstrumentHandlerPrivate* privateData);
    ~AisInstrumentHandler();

    AisErrorCode uploadExperimentToChannel(uint8_t channel, std::shared_ptr<AisExperiment> experiment) const;

    AisErrorCode uploadExperimentToChannel(uint8_t channel, const AisExperiment& experiment) const;

    AisErrorCode startUploadedExperiment(uint8_t channel) const;

    AisErrorCode skipExperimentStep(uint8_t channel) const;

    AisErrorCode pauseExperiment(uint8_t channel) const;

    AisErrorCode resumeExperiment(uint8_t channel) const;

    AisErrorCode stopExperiment(uint8_t channel) const;

    double getExperimentUTCStartTime(uint8_t channel) const;

    AisErrorCode setIRComp(uint8_t channel, double uncompensatedResistance, double compensationLevel) const;

    AisErrorCode setCompRange(uint8_t channel, const AisCompRange& compRange) const;

    int8_t setLinkedChannels(std::vector<uint8_t> channels) const;

    int8_t setBipolarLinkedChannels(std::vector<uint8_t> channels) const;


    bool hasBipolarMode(uint8_t channel) const;

    std::vector<uint8_t> getLinkedChannels(uint8_t channel) const;

    bool isChannelBusy(uint8_t channel) const;

    bool isChannelPaused(uint8_t channel) const;

    std::vector<uint8_t> getFreeChannels() const;

    int getNumberOfChannels() const;

    AisErrorCode eraseRecoverData() const;

    AisErrorCode startManualExperiment(uint8_t channel) const;

    AisErrorCode setManualModeSamplingInterval(uint8_t channel, double value) const;

    AisErrorCode setManualModeOCP(uint8_t channel) const;

    AisErrorCode setManualModeConstantVoltage(uint8_t channel, double value) const;

    AisErrorCode setManualModeConstantVoltage(uint8_t channel, double value, int currentRangeIndex)
  const;

    AisErrorCode setManualModeConstantCurrent(uint8_t channel, double value) const;

    std::vector<std::pair<double, double» getManualModeCurrentRangeList(uint8_t channel) const;

signals:

    void deviceDisconnected();

    void groundFloatStateChanged(bool grounded);

    void experimentNewElementStarting(uint8_t channel, const AisExperimentNode& stepInfo);

    void activeDCDataReady(uint8_t channel, const AisDCData& DCData);

    void idleDCDataReady(uint8_t channel, const AisDCData& DCData);

    void recoveryDCDataReady(uint8_t channel, const AisDCData& DCData);

    void activeACDataReady(uint8_t channel, const AisACData& ACData);

    void recoveryACDataReady(uint8_t channel, const AisACData& ACData);

    void experimentStopped(uint8_t channel);

    void experimentPaused(uint8_t channel);

    void experimentResumed(uint8_t channel);

    void recoverDataErased(bool successful);

private slots:
    void onActiveExperimentNodeBeginning(uint8_t channel, const AisExperimentNode&);
    void onRecoveryExperimentNodeBeginning(uint8_t channel, const AisExperimentNode&);
    void onDeviceDisconnected();

private:
    void connectWithOperatorSignals();
};

#endif //SQUIDSTATLIBRARY_AISINSTRUMENTHANDLER_H
```

16.7 AisSquidstatGlobal.h

```
#pragma once

#include <QtGlobal>


#ifndef BUILD_STATIC
#if defined(SQUIDSTATLIBRARY_LIB)
#define SQUIDSTATLIBRARY_EXPORT Q_DECL_EXPORT
#else
#define SQUIDSTATLIBRARY_EXPORT Q_DECL_IMPORT
#endif
#else
#define SQUIDSTATLIBRARY_EXPORT
#endif
```

16.8 AisAbstractElement.h

```
#ifndef SQUIDSTATLIBRARY_AISABSTRACTELEMENT_H
```

```
#define SQUIDSTATLIBRARY_AISABSTRACTELEMENT_H

#include "AisSquidstatGlobal.h"
#include <QString>
#include <memory>

class AbstractBuilderElement;

class SQUIDSTATLIBRARY_EXPORT AisAbstractElement {
public:
    virtual ~AisAbstractElement();

    virtual QString getName() const = 0;

    virtual QStringList getCategory() const = 0;

protected:
    std::shared_ptr<AbstractBuilderElement> m_data;
    friend class AisExperiment;
};

#endif //SQUIDSTATLIBRARY_AISABSTRACTELEMENT_H
```

16.9 AisConstantCurrentElement.h

```
#pragma once

#include "AisAbstractElement.h"
#include "AisSquidstatGlobal.h"
#include <QString>

class ConstantCurrentAdvElement;

class SQUIDSTATLIBRARY_EXPORT AisConstantCurrentElement final :  public AisAbstractElement {
public:
    explicit AisConstantCurrentElement(
        double current,
        double samplingInterval,
        double duration);
    explicit AisConstantCurrentElement(const AisConstantCurrentElement&);
    AisConstantCurrentElement& operator=(const AisConstantCurrentElement&);

    ~AisConstantCurrentElement() override;

    QString getName() const override;

    QStringList getCategory() const override;

    double getCurrent() const;

    void setCurrent(double current);

    double getSamplingInterval() const;

    void setSamplingInterval(double samplingInterval);

    double getMinSamplingVoltageDifference() const;

    void setMinSamplingVoltageDifference(double minVoltageDifference);

    double getMaxVoltage() const;

    void setMaxVoltage(double maxVoltage);

    double getMinVoltage() const;

    void setMinVoltage(double minVoltage);

    double getMaxDuration() const;

    void setMaxDuration(double maxDuration);

    double getMaxCapacity() const;

    void setMaxCapacity(double maxCapacity);

    bool isAutoRange() const;

    void setAutoRange();

    double getApproxMaxCurrent() const;

    void setApproxMaxCurrent(double approxMaxCurrent);

    bool isAutoVoltageRange() const;

    void setAutoVoltageRange();


    double getApproxMaxVoltage() const;

    void setApproxMaxVoltage(double approxMaxVoltage);

private:
    std::shared_ptr<ConstantCurrentAdvElement> m_dataDerived;
};
```

16.10 AisConstantPotElement.h

```
#pragma once

#include "AisAbstractElement.h"
#include "AisSquidstatGlobal.h"
#include <QString>

class ConstantPotAdvElement;

class SQUIDSTATLIBRARY_EXPORT AisConstantPotElement final :  public AisAbstractElement {
public:
    explicit AisConstantPotElement(
        double voltage,
        double samplingInterval,
        double duration);
    explicit AisConstantPotElement(const AisConstantPotElement&);

    AisConstantPotElement& operator=(const AisConstantPotElement&);

    ~AisConstantPotElement() override;

    QString getName() const override;

    QStringList getCategory() const override;

    double getPotential() const;

    void setPotential(double potential);

    bool isVoltageVsOCP() const;

    void setVoltageVsOCP(bool vsOCP);

    double getSamplingInterval() const;

    void setSamplingInterval(double samplingInterval);

    double getMaxDuration() const;

    void setMaxDuration(double maxDuration);

    double getMaxCurrent() const;

    void setMaxCurrent(double maxCurrent);

    double getMinCurrent() const;

    void setMinCurrent(double minCurrent);

    double getMaxCapacity() const;

    void setMaxCapacity(double maxCapacity);

    double getMindIdt() const;

    void setMindIdt(double mindIdt);

    bool isAutoRange() const;

    void setAutoRange();

    double getApproxMaxCurrent() const;

    void setApproxMaxCurrent(double approxMaxCurrent);

private:
    std::shared_ptr<ConstantPotAdvElement> m_dataDerived;
};
```

16.11 AisConstantPowerElement.h

```
#pragma once

#include "AisAbstractElement.h"
#include "AisSquidstatGlobal.h"
#include <QString>

class ConstantPowerElement;

class SQUIDSTATLIBRARY_EXPORT AisConstantPowerElement final :  public AisAbstractElement {
public:
    explicit AisConstantPowerElement(
        bool isCharge,
        double power,
        double duration,
        double smaplingInterval);

    explicit AisConstantPowerElement(const AisConstantPowerElement&);

    AisConstantPowerElement& operator=(const AisConstantPowerElement&);

    ~AisConstantPowerElement() override;

    QString getName() const override;

    QStringList getCategory() const override;

    bool isCharge() const;

    void setCharge(bool isCharge);

    double getPower() const;

    void setPower(double power);

    double getSamplingInterval() const;

    void setSamplingInterval(double samplingInterval);

    double getMaxVoltage() const;

    void setMaxVoltage(double maxVoltage);

    double getMinVoltage() const;

    void setMinVoltage(double minVoltage);

    double getMaxDuration() const;

    void setMaxDuration(double maxDuration);

    double getMaxCapacity() const;

    void setMaxCapacity(double maxCapacity);

private:
    std::shared_ptr<ConstantPowerElement> m_dataDerived;
};
```

16.12 AisConstantResistanceElement.h

```
#pragma once

#include "AisAbstractElement.h"
#include "AisSquidstatGlobal.h"
#include <QString>

class ConstantResistanceElement;

class SQUIDSTATLIBRARY_EXPORT AisConstantResistanceElement final :  public AisAbstractElement {
public:
    explicit AisConstantResistanceElement(
        double resistance,
        double duration,
        double samplingInterval);
    explicit AisConstantResistanceElement(const AisConstantResistanceElement&);
    AisConstantResistanceElement& operator=(const AisConstantResistanceElement&);

    ~AisConstantResistanceElement() override;

    QString getName() const override;

    QStringList getCategory() const override;

    double getResistance() const;

    void setResistance(double resistance);

    double getSamplingInterval() const;

    void setSamplingInterval(double samplingInterval);

    double getMinVoltage() const;

    void setMinVoltage(double minVoltage);

    double getMaxDuration() const;

    void setMaxDuration(double maxDuration);

    double getMaxCapacity() const;

    void setMaxCapacity(double maxCapacity);

private:
    std::shared_ptr<ConstantResistanceElement> m_dataDerived;
};
```

16.13 AisCyclicVoltammetryElement.h

```
#pragma once

#include "AisAbstractElement.h"
#include "AisSquidstatGlobal.h"
#include <QString>

class CyclicVoltammetryElement;

class SQUIDSTATLIBRARY_EXPORT AisCyclicVoltammetryElement final :  public AisAbstractElement {
public:
    explicit AisCyclicVoltammetryElement(
        double startVoltage,
        double firstVoltageLimit,
        double secondVoltageLimit,
        double endVoltage,
        double dEdt,
        double samplingInterval);

    explicit AisCyclicVoltammetryElement(const AisCyclicVoltammetryElement&);

    AisCyclicVoltammetryElement& operator=(const AisCyclicVoltammetryElement&);

    ~AisCyclicVoltammetryElement() override;

    QString getName() const override;

    QStringList getCategory() const override;

    double getStartVoltage() const;

    void setStartVoltage(double startVoltage);

    bool isStartVoltageVsOCP() const;

    void setStartVoltageVsOCP(bool startVoltageVsOCP);

    double getFirstVoltageLimit() const;

    void setFirstVoltageLimit(double v1);

    bool isFirstVoltageLimitVsOCP() const;

    void setFirstVoltageLimitVsOCP(bool firstVoltageLimitVsOCP);

    double getSecondVoltageLimit() const;

    void setSecondVoltageLimit(double v2);

    bool isSecondVoltageLimitVsOCP() const;

    void setSecondVoltageLimitVsOCP(bool secondVoltageLimitVsOCP);

    double getNumberOfCycles();

    void setNumberOfCycles(int cycles);
```

```cpp

    double getEndVoltage() const;

    void setEndVoltage(double endVoltage);

    bool isEndVoltageVsOCP() const;

    void setEndVoltageVsOCP(bool endVoltageVsOCP);

    double getdEdt() const;

    void setdEdt(double dEdt);

    double getSamplingInterval() const;

    void setSamplingInterval(double sampInterval);

    bool isAutoRange() const;

    void setAutoRange();

    double getApproxMaxCurrent() const;

    void setApproxMaxCurrent(double approxMaxCurrent);

private:
    std::shared_ptr<CyclicVoltammetryElement> m_dataDerived;
};
```

16.14 AisDCCurrentSweepElement.h

```cpp
#pragma once

#include "AisAbstractElement.h"
#include "AisSquidstatGlobal.h"
#include <QString>

class DCCurrentSweepElement;

class SQUIDSTATLIBRARY_EXPORT AisDCCurrentSweepElement :  public AisAbstractElement {
public:
    explicit AisDCCurrentSweepElement(
        double startCurrent,
        double endCurrent,
        double scanRate,
        double samplingInterval

    );
    explicit AisDCCurrentSweepElement(const AisDCCurrentSweepElement&);
    AisDCCurrentSweepElement& operator=(const AisDCCurrentSweepElement&);

    ~AisDCCurrentSweepElement() override;

    QString getName() const override;

    QStringList getCategory() const override;

    double getStartingCurrent() const;

    void setStartingCurrent(double startingCurrent);

    double getEndingCurrent() const;

    void setEndingCurrent(double endingCurrent);

    double getScanRate() const;

    void setScanRate(double scanRate);

    double getSamplingInterval() const;

    void setSamplingInterval(double samplingInterval);

    double getMaxVoltage() const;

    void setMaxVoltage(double maxVoltage);

    double getMinVoltage() const;

    void setMinVoltage(double minVoltage);

private:
    std::shared_ptr<DCCurrentSweepElement> m_dataDerived;
};
```

16.15 AisDCPotentialSweepElement.h

```
#pragma once

#include "AisAbstractElement.h"
#include "AisSquidstatGlobal.h"
#include <QString>

class DCPotentialSweepElement;

class SQUIDSTATLIBRARY_EXPORT AisDCPotentialSweepElement final :  public AisAbstractElement {
public:
    explicit AisDCPotentialSweepElement(
        double startPotential,
        double endPotential,
        double scanRate,
        double samplingInterval);
    explicit AisDCPotentialSweepElement(const AisDCPotentialSweepElement&);
    AisDCPotentialSweepElement& operator=(const AisDCPotentialSweepElement&);

    ~AisDCPotentialSweepElement() override;

    QString getName() const override;
    QStringList getCategory() const override;

    double getStartingPot() const;

    void setStartingPot(double startingPotential);

    bool isStartVoltageVsOCP() const;

    void setStartVoltageVsOCP(bool startVoltageVsOCP);

    double getEndingPot() const;

    void setEndingPot(double endingPotential);

    bool isEndVoltageVsOCP() const;

    void setEndVoltageVsOCP(bool endVoltageVsOCP);

    double getScanRate() const;

    void setScanRate(double scanRate);

    double getSamplingInterval() const;

    void setSamplingInterval(double samplingInterval);

    bool isAutoRange() const;

    void setAutoRange();

    double getApproxMaxCurrent() const;

    void setApproxMaxCurrent(double approxMaxCurrent);

private:
    std::shared_ptr<DCPotentialSweepElement> m_dataDerived;
};
```

16.16 AisDiffPulseVoltammetryElement.h

```
#pragma once

#include "AisAbstractElement.h"
#include "AisSquidstatGlobal.h"
#include <QString>

class DiffPulseVoltammetryElement;

class SQUIDSTATLIBRARY_EXPORT AisDiffPulseVoltammetryElement final :  public AisAbstractElement {
public:
    explicit AisDiffPulseVoltammetryElement(
        double startVoltage,
        double endVoltage,
        double voltageStep,
        double pulseHeight,
        double pulseWidth,
        double pulsePeriod);
    explicit AisDiffPulseVoltammetryElement(const AisDiffPulseVoltammetryElement&);
    AisDiffPulseVoltammetryElement& operator=(const AisDiffPulseVoltammetryElement&);

    ~AisDiffPulseVoltammetryElement() override;

    QString getName() const override;

    QStringList getCategory() const override;

    double getStartVoltage() const;

    void setStartVoltage(double startVoltage);

    bool isStartVoltageVsOCP() const;

    void setStartVoltageVsOCP(bool startVoltageVsOCP);

    double getEndVoltage() const;

    void setEndVoltage(double endVoltage);

    bool isEndVoltageVsOCP() const;

    void setEndVoltageVsOCP(bool endVoltageVsOCP);

    double getVStep() const;

    void setVStep(double vStep);

    double getPulseHeight() const;

    void setPulseHeight(double pulseHeight);

    double getPulseWidth() const;

    void setPulseWidth(double pulseWidth);

    double getPulsePeriod() const;

    void setPulsePeriod(double pulsePeriod);

    bool isAutoRange() const;

    void setAutoRange();

    double getApproxMaxCurrent() const;

    void setApproxMaxCurrent(double approxMaxCurrent);

private:
    std::shared_ptr<DiffPulseVoltammetryElement> m_dataDerived;
};
```

16.17 AisEISGalvanostaticElement.h

```
#pragma once

#include "AisAbstractElement.h"
#include "AisSquidstatGlobal.h"
#include <QString>

class EISGalvanostaticElement;

class SQUIDSTATLIBRARY_EXPORT AisEISGalvanostaticElement final :  public AisAbstractElement {
public:
    explicit AisEISGalvanostaticElement(
        double startFrequency,
        double endFrequency,
        double stepsPerDecade,
        double currentBias,
        double currentAamplitude);
    explicit AisEISGalvanostaticElement(const AisEISGalvanostaticElement&);
    AisEISGalvanostaticElement& operator=(const AisEISGalvanostaticElement&);

    ~AisEISGalvanostaticElement() override;

    QString getName() const override;

    QStringList getCategory() const override;

    double getStartFreq() const;

    void setStartFreq(double startFreq);

    double getEndFreq() const;

    void setEndFreq(double endFreq);

    double getStepsPerDecade() const;

    void setStepsPerDecade(double stepsPerDecade);

    double getBiasCurrent() const;

    void setBiasCurrent(double biasCurrent);

    double getAmplitude() const;

    void setAmplitude(double amplitude);

private:
    std::shared_ptr<EISGalvanostaticElement> m_dataDerived;
};
```

16.18 AisEISPotentiostaticElement.h

```
#pragma once

#include "AisAbstractElement.h"
#include "AisSquidstatGlobal.h"
#include <QString>

class EISPotentiostaticElement;

class SQUIDSTATLIBRARY_EXPORT AisEISPotentiostaticElement final :  public AisAbstractElement {
public:
    explicit AisEISPotentiostaticElement(
        double startFrequency,
        double endFrequency,
        double stepsPerDecade,
        double voltageBias,
        double voltageAamplitude);
    explicit AisEISPotentiostaticElement(const AisEISPotentiostaticElement&);
    AisEISPotentiostaticElement& operator=(const AisEISPotentiostaticElement&);

    ~AisEISPotentiostaticElement() override;

    QString getName() const override;

    QStringList getCategory() const override;

    double getStartFreq() const;

    void setStartFreq(double startFreq);

    double getEndFreq() const;

    void setEndFreq(double endFreq);

    double getStepsPerDecade() const;

    void setStepsPerDecade(double stepsPerDecade);

    double getBiasVoltage() const;

    void setBiasVoltage(double biasVoltage);

    bool isBiasVoltageVsOCP() const;

    void setBiasVoltageVsOCP(bool biasVsOCP);

    double getAmplitude() const;

    void setAmplitude(double amplitude);

private:
    std::shared_ptr<EISPotentiostaticElement> m_dataDerived;
};
```

16.19 AisNormalPulseVoltammetryElement.h

```
#pragma once

#include "AisAbstractElement.h"
#include "AisSquidstatGlobal.h"
```

```
#include <QString>

class NormalPulseVoltammetryElement;

class SQUIDSTATLIBRARY_EXPORT AisNormalPulseVoltammetryElement final :  public AisAbstractElement {
public:
    explicit AisNormalPulseVoltammetryElement(
        double startVoltage,
        double endVoltage,
        double voltageStep,
        double pulseWidth,
        double pulsePeriod);
    explicit AisNormalPulseVoltammetryElement(const AisNormalPulseVoltammetryElement&);
    AisNormalPulseVoltammetryElement& operator=(const AisNormalPulseVoltammetryElement&);

    ~AisNormalPulseVoltammetryElement() override;

    QString getName() const override;

    QStringList getCategory() const override;

    double getStartVoltage() const;

    void setStartVoltage(double startVoltage);

    bool isStartVoltageVsOCP() const;

    void setStartVoltageVsOCP(bool startVoltageVsOCP);

    double getEndVoltage() const;

    void setEndVoltage(double endVoltage);

    bool isEndVoltageVsOCP() const;

    void setEndVoltageVsOCP(bool endVoltageVsOcp);

    double getVStep() const;

    void setVStep(double vStep);

    double getPulseWidth() const;

    void setPulseWidth(double pulseWidth);

    double getPulsePeriod() const;

    void setPulsePeriod(double pulsePeriod);

    bool isAutoRange() const;

    void setAutoRange();

    double getApproxMaxCurrent() const;

    void setApproxMaxCurrent(double approxMaxCurrent);

private:
    std::shared_ptr<NormalPulseVoltammetryElement> m_dataDerived;
};
```

16.20 AisOpenCircuitElement.h

```
#pragma once

#include "AisAbstractElement.h"
#include "AisSquidstatGlobal.h"
#include <QString>

class OpenCircuitElement;

class SQUIDSTATLIBRARY_EXPORT AisOpenCircuitElement final :  public AisAbstractElement {
public:
    explicit AisOpenCircuitElement(
        double duration,
        double samplingInterval);
    explicit AisOpenCircuitElement(const AisOpenCircuitElement&);
    AisOpenCircuitElement& operator=(const AisOpenCircuitElement&);

    ~AisOpenCircuitElement() override;

    QString getName() const override;

    QStringList getCategory() const override;

    double getSamplingInterval() const;

    void setSamplingInterval(double samplingInterval);

    double getMaxDuration() const;

    void setMaxDuration(double maxDuration);

    double getMaxVoltage() const;

    void setMaxVoltage(double maxVoltage);

    double getMinVoltage() const;

    void setMinVoltage(double minVoltage);

    double getMindVdt() const;

    void setMindVdt(double mindVdt);

    bool isAutoVoltageRange() const;

    void setAutoVoltageRange();

    double getApproxMaxVoltage() const;

    void setApproxMaxVoltage(double approxMaxVoltage);

private:
    std::shared_ptr<OpenCircuitElement> m_dataDerived;
};
```

16.21 AisSquareWaveVoltammetryElement.h

```
#pragma once

#include "AisAbstractElement.h"
#include "AisSquidstatGlobal.h"
#include <QString>

class SquareWaveVoltammetryElement;

class SQUIDSTATLIBRARY_EXPORT AisSquareWaveVoltammetryElement final :  public AisAbstractElement {
public:
    explicit AisSquareWaveVoltammetryElement(
        double startVoltage,
        double endVoltage,
        double voltageStep,
        double pulseAmp,
        double pulseFrequency);

    explicit AisSquareWaveVoltammetryElement(const AisSquareWaveVoltammetryElement&);
    AisSquareWaveVoltammetryElement& operator=(const AisSquareWaveVoltammetryElement&);

    ~AisSquareWaveVoltammetryElement() override;

    QString getName() const override;

    QStringList getCategory() const override;

    double getStartVoltage() const;

    void setStartVoltage(double startVoltage);

    bool isStartVoltageVsOCP() const;

    void setStartVoltageVsOCP(bool startVoltageVsOcp);

    double getEndVoltage() const;

    void setEndVoltage(double endVoltage);

    bool isEndVoltageVsOCP() const;

    void setEndVoltageVsOCP(bool endVoltageVsOcp);

    double getVStep() const;

    void setVStep(double vStep);

    double getPulseAmp() const;

    void setPulseAmp(double pulseAmp);

    double getPulseFreq() const;

    void setPulseFreq(double pulseFreq);

    bool isAutoRange() const;

    void setAutoRange();

    double getApproxMaxCurrent() const;

    void setApproxMaxCurrent(double approxMaxCurrent);

private:
    std::shared_ptr<SquareWaveVoltammetryElement> m_dataDerived;
};
```

Index