

Squidstat API User Manual

Generated by Admiral Instruments LLC

June 16, 2023

1 Squidstat API User Manual	1
2 The Basics of Running Experiments	3
2.0.1 Creating A Custom Experiment	3
2.0.2 Controlling The Experiment	4
2.0.2.1 Creating Control Flow Logic Specific To A Handler	5
2.0.2.2 Connecting Slots To Device-Tracker Signals	6
3 Advanced Control Flow	9
4 Manual Experiments	13
5 PythonExample	15
5.0.0.1 Introduction	15
5.0.0.2 Building a Custom Experiment with Python	15
6 FirmwareUpdate	17
6.0.0.1 Introduction	17
6.0.0.2 Building a Custom Experiment with Python	17
7 Remote access to Squidstat	19
7.0.0.1 Introduction	19
7.0.0.2 Server Implementation	19
7.0.0.3 Client Implementation	21
7.0.0.4 Full Example	22
8 Hierarchical Index	25
8.1 Class Hierarchy	25
9 Class Index	27
9.1 Class List	27
10 File Index	29
10.1 File List	29
11 Class Documentation	31
11.1 AisACData Struct Reference	31
11.1.1 Detailed Description	32
11.1.2 Member Data Documentation	32
11.1.2.1 numberOfCycles	32
11.2 AisCompRange Class Reference	32
11.2.1 Detailed Description	33
11.2.2 Constructor & Destructor Documentation	33
11.2.2.1 AisCompRange()	33
11.2.3 Member Function Documentation	33
11.2.3.1 getBandwidthIndex()	33

11.2.3.2 getCompRangeName()	34
11.2.3.3 getStabilityFactor()	34
11.2.3.4 setBandwidthIndex()	34
11.2.3.5 setCompRangeName()	34
11.2.3.6 setStabilityFactor()	35
11.3 AisConstantCurrentElement Class Reference	35
11.3.1 Detailed Description	37
11.3.2 Constructor & Destructor Documentation	37
11.3.2.1 AisConstantCurrentElement()	37
11.3.3 Member Function Documentation	37
11.3.3.1 getApproxMaxCurrent()	38
11.3.3.2 getApproxMaxVoltage()	38
11.3.3.3 getCategory()	38
11.3.3.4 getCurrent()	39
11.3.3.5 getMaxCapacity()	39
11.3.3.6 getMaxDuration()	39
11.3.3.7 getMaxVoltage()	39
11.3.3.8 getMinSamplingVoltageDifference()	40
11.3.3.9 getMinVoltage()	40
11.3.3.10 getName()	40
11.3.3.11 getSamplingInterval()	41
11.3.3.12 isAutoRange()	41
11.3.3.13 isAutoVoltageRange()	41
11.3.3.14 setApproxMaxCurrent()	41
11.3.3.15 setApproxMaxVoltage()	42
11.3.3.16 setAutoRange()	42
11.3.3.17 setAutoVoltageRange()	42
11.3.3.18 setCurrent()	42
11.3.3.19 setMaxCapacity()	43
11.3.3.20 setMaxDuration()	43
11.3.3.21 setMaxVoltage()	43
11.3.3.22 setMinSamplingVoltageDifference()	44
11.3.3.23 setMinVoltage()	44
11.3.3.24 setSamplingInterval()	44
11.4 AisConstantPotElement Class Reference	45
11.4.1 Detailed Description	46
11.4.2 Constructor & Destructor Documentation	46
11.4.2.1 AisConstantPotElement()	47
11.4.3 Member Function Documentation	47
11.4.3.1 getApproxMaxCurrent()	47
11.4.3.2 getCategory()	47
11.4.3.3 getMaxCapacity()	48

11.4.3.4 getMaxCurrent()	48
11.4.3.5 getMaxDuration()	48
11.4.3.6 getMinCurrent()	49
11.4.3.7 getMindIdt()	49
11.4.3.8 getName()	49
11.4.3.9 getPotential()	50
11.4.3.10 getSamplingInterval()	50
11.4.3.11 isAutoRange()	50
11.4.3.12 isVoltageVsOCP()	50
11.4.3.13 setApproxMaxCurrent()	50
11.4.3.14 setAutoRange()	51
11.4.3.15 setMaxCapacity()	51
11.4.3.16 setMaxCurrent()	51
11.4.3.17 setMaxDuration()	52
11.4.3.18 setMinCurrent()	52
11.4.3.19 setMindIdt()	52
11.4.3.20 setPotential()	53
11.4.3.21 setSamplingInterval()	53
11.4.3.22 setVoltageVsOCP()	53
11.5 AisConstantPowerElement Class Reference	53
11.5.1 Detailed Description	55
11.5.2 Constructor & Destructor Documentation	55
11.5.2.1 AisConstantPowerElement()	55
11.5.3 Member Function Documentation	56
11.5.3.1 getCategory()	56
11.5.3.2 getMaxCapacity()	56
11.5.3.3 getMaxDuration()	56
11.5.3.4 getMaxVoltage()	57
11.5.3.5 getMinVoltage()	57
11.5.3.6 getName()	57
11.5.3.7 getPower()	58
11.5.3.8 getSamplingInterval()	58
11.5.3.9 isCharge()	58
11.5.3.10 setCharge()	58
11.5.3.11 setMaxCapacity()	59
11.5.3.12 setMaxDuration()	59
11.5.3.13 setMaxVoltage()	59
11.5.3.14 setMinVoltage()	60
11.5.3.15 setPower()	60
11.5.3.16 setSamplingInterval()	60
11.6 AisConstantResistanceElement Class Reference	61
11.6.1 Detailed Description	62

11.6.2 Constructor & Destructor Documentation	62
11.6.2.1 AisConstantResistanceElement()	62
11.6.3 Member Function Documentation	62
11.6.3.1 getCategory()	63
11.6.3.2 getMaxCapacity()	63
11.6.3.3 getMaxDuration()	63
11.6.3.4 getMinVoltage()	63
11.6.3.5 getName()	64
11.6.3.6 getResistance()	64
11.6.3.7 getSamplingInterval()	64
11.6.3.8 setMaxCapacity()	64
11.6.3.9 setMaxDuration()	65
11.6.3.10 setMinVoltage()	65
11.6.3.11 setResistance()	65
11.6.3.12 setSamplingInterval()	66
11.7 AisCyclicVoltammetryElement Class Reference	66
11.7.1 Detailed Description	67
11.7.2 Constructor & Destructor Documentation	68
11.7.2.1 AisCyclicVoltammetryElement()	68
11.7.3 Member Function Documentation	68
11.7.3.1 getApproxMaxCurrent()	68
11.7.3.2 getCategory()	69
11.7.3.3 getdEdt()	69
11.7.3.4 getEndVoltage()	69
11.7.3.5 getFirstVoltageLimit()	69
11.7.3.6 getName()	70
11.7.3.7 getNumberOfCycles()	70
11.7.3.8 getSamplingInterval()	70
11.7.3.9 getSecondVoltageLimit()	70
11.7.3.10 getStartVoltage()	71
11.7.3.11 isAutoRange()	71
11.7.3.12 isEndVoltageVsOCP()	71
11.7.3.13 isFirstVoltageLimitVsOCP()	71
11.7.3.14 isSecondVoltageLimitVsOCP()	72
11.7.3.15 isStartVoltageVsOCP()	72
11.7.3.16 setApproxMaxCurrent()	72
11.7.3.17 setAutoRange()	72
11.7.3.18 setdEdt()	73
11.7.3.19 setEndVoltage()	73
11.7.3.20 setEndVoltageVsOCP()	73
11.7.3.21 setFirstVoltageLimit()	73
11.7.3.22 setFirstVoltageLimitVsOCP()	74

11.7.3.23 setNumberOfCycles()	74
11.7.3.24 setSamplingInterval()	74
11.7.3.25 setSecondVoltageLimit()	75
11.7.3.26 setSecondVoltageLimitVsOCP()	75
11.7.3.27 setStartVoltage()	75
11.7.3.28 setStartVoltageVsOCP()	76
11.8 AisDCCurrentSweepElement Class Reference	76
11.8.1 Detailed Description	77
11.8.2 Constructor & Destructor Documentation	78
11.8.2.1 AisDCCurrentSweepElement()	78
11.8.3 Member Function Documentation	79
11.8.3.1 getCategory()	79
11.8.3.2 getEndingCurrent()	79
11.8.3.3 getMaxVoltage()	79
11.8.3.4 getMinVoltage()	80
11.8.3.5 getName()	80
11.8.3.6 getSamplingInterval()	80
11.8.3.7 getScanRate()	80
11.8.3.8 getStartingCurrent()	81
11.8.3.9 setEndingCurrent()	81
11.8.3.10 setMaxVoltage()	81
11.8.3.11 setMinVoltage()	81
11.8.3.12 setSamplingInterval()	82
11.8.3.13 setScanRate()	82
11.8.3.14 setStartingCurrent()	82
11.9 AisDCData Struct Reference	83
11.9.1 Detailed Description	83
11.10 AisDCPotentialSweepElement Class Reference	83
11.10.1 Detailed Description	85
11.10.2 Constructor & Destructor Documentation	85
11.10.2.1 AisDCPotentialSweepElement()	85
11.10.3 Member Function Documentation	86
11.10.3.1 getApproxMaxCurrent()	86
11.10.3.2 getCategory()	86
11.10.3.3 getEndingPot()	86
11.10.3.4 getName()	87
11.10.3.5 getSamplingInterval()	87
11.10.3.6 getScanRate()	87
11.10.3.7 getStartingPot()	87
11.10.3.8 isAutoRange()	88
11.10.3.9 isEndVoltageVsOCP()	88
11.10.3.10 isStartVoltageVsOCP()	88

11.10.3.11 setApproxMaxCurrent()	88
11.10.3.12 setAutoRange()	89
11.10.3.13 setEndingPot()	89
11.10.3.14 setEndVoltageVsOCP()	89
11.10.3.15 setSamplingInterval()	90
11.10.3.16 setScanRate()	90
11.10.3.17 setStartingPot()	90
11.10.3.18 setStartVoltageVsOCP()	90
11.11 AisDeviceTracker Class Reference	91
11.11.1 Detailed Description	92
11.11.2 Member Function Documentation	92
11.11.2.1 connectAllPluggedInDevices()	92
11.11.2.2 connectToDeviceOnComPort()	92
11.11.2.3 deviceDisconnected	93
11.11.2.4 getConnectedDevices()	93
11.11.2.5 getInstrumentHandler()	93
11.11.2.6 newDeviceConnected	94
11.11.2.7 updateFirmwareOnAllAvailableDevices()	94
11.11.2.8 updateFirmwareOnComPort()	95
11.12 AisDiffPulseVoltammetryElement Class Reference	95
11.12.1 Detailed Description	97
11.12.2 Constructor & Destructor Documentation	97
11.12.2.1 AisDiffPulseVoltammetryElement()	97
11.12.3 Member Function Documentation	98
11.12.3.1 getApproxMaxCurrent()	98
11.12.3.2 getCategory()	98
11.12.3.3 getEndVoltage()	98
11.12.3.4 getName()	99
11.12.3.5 getPulseHeight()	99
11.12.3.6 getPulsePeriod()	99
11.12.3.7 getPulseWidth()	100
11.12.3.8 getStartVoltage()	100
11.12.3.9 getVStep()	100
11.12.3.10 isAutoRange()	101
11.12.3.11 isEndVoltageVsOCP()	101
11.12.3.12 isStartVoltageVsOCP()	101
11.12.3.13 setApproxMaxCurrent()	101
11.12.3.14 setAutoRange()	102
11.12.3.15 setEndVoltage()	102
11.12.3.16 setEndVoltageVsOCP()	102
11.12.3.17 setPulseHeight()	103
11.12.3.18 setPulsePeriod()	103

11.12.3.19 setPulseWidth()	103
11.12.3.20 setStartVoltage()	104
11.12.3.21 setStartVoltageVsOCP()	104
11.12.3.22 setVStep()	104
11.13 AisEISGalvanostaticElement Class Reference	105
11.13.1 Detailed Description	106
11.13.2 Constructor & Destructor Documentation	106
11.13.2.1 AisEISGalvanostaticElement()	106
11.13.3 Member Function Documentation	107
11.13.3.1 getAmplitude()	107
11.13.3.2 getBiasCurrent()	107
11.13.3.3 getCategory()	107
11.13.3.4 getEndFreq()	108
11.13.3.5 getName()	108
11.13.3.6 getStartFreq()	108
11.13.3.7 getStepsPerDecade()	108
11.13.3.8 setAmplitude()	108
11.13.3.9 setBiasCurrent()	109
11.13.3.10 setEndFreq()	109
11.13.3.11 setStartFreq()	109
11.13.3.12 setStepsPerDecade()	110
11.14 AisEISPotentiostaticElement Class Reference	110
11.14.1 Detailed Description	111
11.14.2 Constructor & Destructor Documentation	111
11.14.2.1 AisEISPotentiostaticElement()	112
11.14.3 Member Function Documentation	112
11.14.3.1 getAmplitude()	112
11.14.3.2 getBiasVoltage()	112
11.14.3.3 getCategory()	113
11.14.3.4 getEndFreq()	113
11.14.3.5 getName()	113
11.14.3.6 getStartFreq()	113
11.14.3.7 getStepsPerDecade()	114
11.14.3.8 isBiasVoltageVsOCP()	114
11.14.3.9 setAmplitude()	114
11.14.3.10 setBiasVoltage()	114
11.14.3.11 setBiasVoltageVsOCP()	115
11.14.3.12 setEndFreq()	115
11.14.3.13 setStartFreq()	115
11.14.3.14 setStepsPerDecade()	116
11.15 AisErrorCode Class Reference	116
11.15.1 Detailed Description	117

11.15.2 Member Enumeration Documentation	117
11.15.2.1 ErrorCode	117
11.15.3 Member Function Documentation	118
11.15.3.1 message()	118
11.15.3.2 value()	118
11.16 AisExperiment Class Reference	119
11.16.1 Detailed Description	119
11.16.2 Constructor & Destructor Documentation	119
11.16.2.1 AisExperiment()	119
11.16.3 Member Function Documentation	120
11.16.3.1 appendElement()	120
11.16.3.2 appendSubExperiment()	120
11.16.3.3 getCategory()	121
11.16.3.4 getDescription()	121
11.16.3.5 getExperimentName()	121
11.16.3.6 operator=()	121
11.16.3.7 setDescription()	122
11.16.3.8 setExperimentName()	122
11.17 AisExperimentNode Struct Reference	122
11.17.1 Detailed Description	123
11.18 AisInstrumentHandler Class Reference	123
11.18.1 Detailed Description	125
11.18.2 Member Function Documentation	125
11.18.2.1 activeACDataReady	125
11.18.2.2 activeDCDataReady	125
11.18.2.3 deviceDisconnected	125
11.18.2.4 eraseRecoverData()	126
11.18.2.5 experimentNewElementStarting	126
11.18.2.6 experimentPaused	126
11.18.2.7 experimentResumed	127
11.18.2.8 experimentStopped	127
11.18.2.9 getExperimentUTCStartTime()	127
11.18.2.10 getFreeChannels()	127
11.18.2.11 getLinkedChannels()	128
11.18.2.12 getManualModeCurrentRangeList()	128
11.18.2.13 getNumberOfChannels()	128
11.18.2.14 groundFloatStateChanged	129
11.18.2.15 idleDCDataReady	129
11.18.2.16 isChannelBusy()	129
11.18.2.17 isChannelPaused()	130
11.18.2.18 pauseExperiment()	130
11.18.2.19 recoverDataErased	131

11.18.2.20 recoveryACDataReady	131
11.18.2.21 recoveryDCDataReady	131
11.18.2.22 resumeExperiment()	131
11.18.2.23 setCompRange()	132
11.18.2.24 setIRComp()	133
11.18.2.25 setLinkedChannels()	134
11.18.2.26 setManualModeConstantCurrent()	134
11.18.2.27 setManualModeConstantVoltage() [1/2]	135
11.18.2.28 setManualModeConstantVoltage() [2/2]	135
11.18.2.29 setManualModeOCP()	136
11.18.2.30 setManualModeSamplingInterval()	136
11.18.2.31 skipExperimentStep()	137
11.18.2.32 startManualExperiment()	137
11.18.2.33 startUploadedExperiment()	138
11.18.2.34 stopExperiment()	138
11.18.2.35 uploadExperimentToChannel() [1/2]	139
11.18.2.36 uploadExperimentToChannel() [2/2]	140
11.19 AisNormalPulseVoltammetryElement Class Reference	140
11.19.1 Detailed Description	142
11.19.2 Constructor & Destructor Documentation	142
11.19.2.1 AisNormalPulseVoltammetryElement()	142
11.19.3 Member Function Documentation	143
11.19.3.1 getApproxMaxCurrent()	143
11.19.3.2 getCategory()	143
11.19.3.3 getEndVoltage()	143
11.19.3.4 getName()	144
11.19.3.5 getPulsePeriod()	144
11.19.3.6 getPulseWidth()	144
11.19.3.7 getStartVoltage()	145
11.19.3.8 getVStep()	145
11.19.3.9 isAutoRange()	145
11.19.3.10 isEndVoltageVsOCP()	145
11.19.3.11 isStartVoltageVsOCP()	146
11.19.3.12 setApproxMaxCurrent()	146
11.19.3.13 setAutoRange()	146
11.19.3.14 setEndVoltage()	146
11.19.3.15 setEndVoltageVsOCP()	147
11.19.3.16 setPulsePeriod()	147
11.19.3.17 setPulseWidth()	147
11.19.3.18 setStartVoltage()	148
11.19.3.19 setStartVoltageVsOCP()	148
11.19.3.20 setVStep()	148

11.20 AisOpenCircuitElement Class Reference	149
11.20.1 Detailed Description	150
11.20.2 Constructor & Destructor Documentation	150
11.20.2.1 AisOpenCircuitElement()	150
11.20.3 Member Function Documentation	151
11.20.3.1 getApproxMaxVoltage()	151
11.20.3.2 getCategory()	151
11.20.3.3 getMaxDuration()	151
11.20.3.4 getMaxVoltage()	152
11.20.3.5 getMindVdt()	152
11.20.3.6 getMinVoltage()	152
11.20.3.7 getName()	153
11.20.3.8 getSamplingInterval()	153
11.20.3.9 isAutoVoltageRange()	153
11.20.3.10 setApproxMaxVoltage()	153
11.20.3.11 setAutoVoltageRange()	154
11.20.3.12 setMaxDuration()	154
11.20.3.13 setMaxVoltage()	154
11.20.3.14 setMindVdt()	154
11.20.3.15 setMinVoltage()	155
11.20.3.16 setSamplingInterval()	155
11.21 AisSquareWaveVoltammetryElement Class Reference	155
11.21.1 Detailed Description	157
11.21.2 Constructor & Destructor Documentation	157
11.21.2.1 AisSquareWaveVoltammetryElement()	157
11.21.3 Member Function Documentation	158
11.21.3.1 getApproxMaxCurrent()	158
11.21.3.2 getCategory()	158
11.21.3.3 getEndVoltage()	159
11.21.3.4 getName()	159
11.21.3.5 getPulseAmp()	159
11.21.3.6 getPulseFreq()	159
11.21.3.7 getStartVoltage()	160
11.21.3.8 getVStep()	160
11.21.3.9 isAutoRange()	160
11.21.3.10 isEndVoltageVsOCP()	160
11.21.3.11 isStartVoltageVsOCP()	161
11.21.3.12 setApproxMaxCurrent()	161
11.21.3.13 setAutoRange()	161
11.21.3.14 setEndVoltage()	161
11.21.3.15 setEndVoltageVsOCP()	162
11.21.3.16 setPulseAmp()	162

11.21.3.17 setPulseFreq()	162
11.21.3.18 setStartVoltage()	164
11.21.3.19 setStartVoltageVsOCP()	164
11.21.3.20 setVStep()	164
12 File Documentation	167
12.1 AisCompRange.h	167
12.2 AisDataPoints.h	167
12.3 AisDeviceTracker.h	168
12.4 AisErrorCode.h	169
12.5 AisExperiment.h	169
12.6 AisInstrumentHandler.h	170
12.7 AisSquidstatGlobal.h	171
12.8 AisAbstractElement.h	171
12.9 AisConstantCurrentElement.h	172
12.10 AisConstantPotElement.h	173
12.11 AisConstantPowerElement.h	174
12.12 AisConstantResistanceElement.h	174
12.13 AisCyclicVoltammetryElement.h	175
12.14 AisDCCurrentSweepElement.h	176
12.15 AisDCPotentialSweepElement.h	177
12.16 AisDiffPulseVoltammetryElement.h	177
12.17 AisEISGalvanostaticElement.h	178
12.18 AisEISPotentiostaticElement.h	179
12.19 AisNormalPulseVoltammetryElement.h	179
12.20 AisOpenCircuitElement.h	180
12.21 AisSquareWaveVoltammetryElement.h	181
Index	183

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

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.

2.0.1 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.

2.0.2 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.

2.0.2.1 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>

2.0.2.2 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.

2.0.2.2.1 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);

```

2.0.2.2.2 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 3

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 4

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 5

PythonExample

5.0.0.1 Introduction

The following example will help illustrate the use of the Squidstatlibrary with Python3.7+. The necessary Python library files are located inside the `pythonWrapper` directory.

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

5.0.0.2 Building a Custom Experiment with Python

5.0.0.2.1 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
```

5.0.0.2.2 Import experiment modules depeneding on the requirement. `from SquidstatPyLibrary import`

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

5.0.0.2.3 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)
```

5.0.0.2.4 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("Acel102");
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_())
```

5.0.0.2.5 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("Acell102");
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 6

FirmwareUpdate

6.0.0.1 Introduction

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

We will go through an example.

6.0.0.2 Building a Custom Experiment with Python

6.0.0.2.1 Import all the required basic class from SquidstatLibrary, and Qt Library. #include

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

6.0.0.2.2 Connect the Notification signal. QObject::connect(tracker, &AisDeviceTracker::firmwareUpdateNotification, &a, [=](const QString& message) { qInfo() << message; });

6.0.0.2.3 request to update the firmware to all connected device. auto numberOfDevice = tracker->updateFirmwareOnAllAvailableDevices(); if (numberOfDevice == 0) { qInfo() << "Firmware update is not start in any of device"; } else { qInfo() << "Firmware update start in " << numberOfDevice << "device.";

6.0.0.2.4 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();
    }
}
```

6.0.0.2.5 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 numberOfDevice = tracker->updateFirmwareOnAllAvailableDevices();
        if (numberOfDevice == 0) {
            qInfo() « "Firmware update is not start in any of device";
        } else {
            qInfo() « "Firmware update start in " « numberOfDevice « "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 7

Remote access to Squidstat

7.0.0.1 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.

7.0.0.2 Server Implementation

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

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.

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

These 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.

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

`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.

```
# 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))
```

`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.

```
# 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_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.

```
# 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_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.

```
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())

```

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

```

# 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")

```

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.

```

# 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()

```

7.0.0.3 Client Implementation

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

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

```

SERVER_HOST = "localhost"
SERVER_PORT = 12345

```

`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.

```

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)

```

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

```

try:
    client_socket.connect((SERVER_HOST, SERVER_PORT))

```

```
except Exception as ex:
    print("Unable to establish connection to server:\n%s" % ex)
    exit()
```

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

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

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.

```
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
```

7.0.0.4 Full Example

```
7.0.0.4.1 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 {SQUIDCOMPRT}...")
tracker = AisDeviceTracker.Instance()
tracker.newDeviceConnected.connect(lambda deviceName: print("Device is Connected: %s" % deviceName))
error = tracker.connectToDeviceOnComPort(SQUIDCOMPRT)
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_()

```

7.0.0.4.2 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 interrupt_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}')
interrupt_thread = threading.Thread(target=interrupt_listener)
interrupt_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 8

Hierarchical Index

8.1 Class Hierarchy

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

AisACData	31
AisCompRange	32
AisConstantCurrentElement	35
AisConstantPotElement	45
AisConstantPowerElement	53
AisConstantResistanceElement	61
AisCyclicVoltammetryElement	66
AisDCCurrentSweepElement	76
AisDCData	83
AisDCPotentialSweepElement	83
AisDiffPulseVoltammetryElement	95
AisEISGalvanostaticElement	105
AisEISPotentiostaticElement	110
AisErrorCode	116
AisExperiment	119
AisExperimentNode	122
AisNormalPulseVoltammetryElement	140
AisOpenCircuitElement	149
AisSquareWaveVoltammetryElement	155
QObject	
AisDeviceTracker	91
AisInstrumentHandler	123

Chapter 9

Class Index

9.1 Class List

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

AisACData	Structure containing AC data information	31
AisCompRange	This class has advanced options controlling the device stability including the bandwidth index and the stability factor	32
AisConstantCurrentElement	Experiment that simulates a constant current flow with more advance options for stopping the experiment. 35	
AisConstantPotElement	Experiment that simulates a constant applied voltage. 45	
AisConstantPowerElement	This experiment simulates a constant power, charge or discharge". 53	
AisConstantResistanceElement	This element/experiment simulates a constant resistance load. 61	
AisCyclicVoltammetryElement	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	66
AisDCCurrentSweepElement	This experiment performs a DC current sweep from the starting current to the ending current which progresses linearly according to the scan rate	76
AisDCData	Structure containing DC data information	83
AisDCPotentialSweepElement	This experiment performs a DC potential sweep from the starting current to the ending current which progresses linearly according to the scan rate	83
AisDeviceTracker	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	91

AisDiffPulseVoltammetryElement	
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	95
AisEISGalvanostaticElement	
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	105
AisEISPotentiostaticElement	
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	110
AisErrorCode	
This class contains the possible error codes returned to the user when working with the API . .	116
AisExperiment	
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	119
AisExperimentNode	
Structure containing some information regarding the running element	122
AisInstrumentHandler	
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	123
AisNormalPulseVoltammetryElement	
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	140
AisOpenCircuitElement	
This experiment observes the open circuit potential of the working electrode for a specific period of time.	149
AisSquareWaveVoltammetryElement	
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	155

Chapter 10

File Index

10.1 File List

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

AisCompRange.h	167
AisDataPoints.h	167
AisDeviceTracker.h	168
AisErrorCode.h	169
AisExperiment.h	169
AisInstrumentHandler.h	170
AisSquidstatGlobal.h	171
AisAbstractElement.h	171
AisConstantCurrentElement.h	172
AisConstantPotElement.h	173
AisConstantPowerElement.h	174
AisConstantResistanceElement.h	174
AisCyclicVoltammetryElement.h	175
AisDCCurrentSweepElement.h	176
AisDCPotentialSweepElement.h	177
AisDiffPulseVoltammetryElement.h	177
AisEISGalvanostaticElement.h	178
AisEISPotentiostaticElement.h	179
AisNormalPulseVoltammetryElement.h	179
AisOpenCircuitElement.h	180
AisSquareWaveVoltammetryElement.h	181

Chapter 11

Class Documentation

11.1 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.

11.1.1 Detailed Description

a structure containing AC data information.

11.1.2 Member Data Documentation

11.1.2.1 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

11.2 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.

11.2.1 Detailed Description

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

See also

[setBandwidthIndex](#)

[setStabilityFactor](#)

11.2.2 Constructor & Destructor Documentation

11.2.2.1 AisCompRange()

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

constructor for the compensation-range object.

Parameters

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

See also

[setBandwidthIndex](#)

[setStabilityFactor](#)

11.2.3 Member Function Documentation

11.2.3.1 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](#)

11.2.3.2 getCompRangeName()

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

get the name set for the compensation range.

Returns

the name set for the compensation range.

11.2.3.3 getStabilityFactor()

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

get the value set for the stability factor.

Returns

the value set for the stability factor.

11.2.3.4 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

<i>index</i>	the index value for the bandwidth (0-10).
--------------	-------------------------------------------

11.2.3.5 setCompRangeName()

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

set a name for the compensation range for reference purposes.

Parameters

<i>compRangeName</i>	the name to set for the compensation range.
----------------------	---------------------------------------------

11.2.3.6 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

<i>factor</i>	the stability-factor value (0-10)
---------------	-----------------------------------

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

- AisCompRange.h

11.3 AisConstantCurrentElement Class Reference

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

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

Inherits AisAbstractElement.

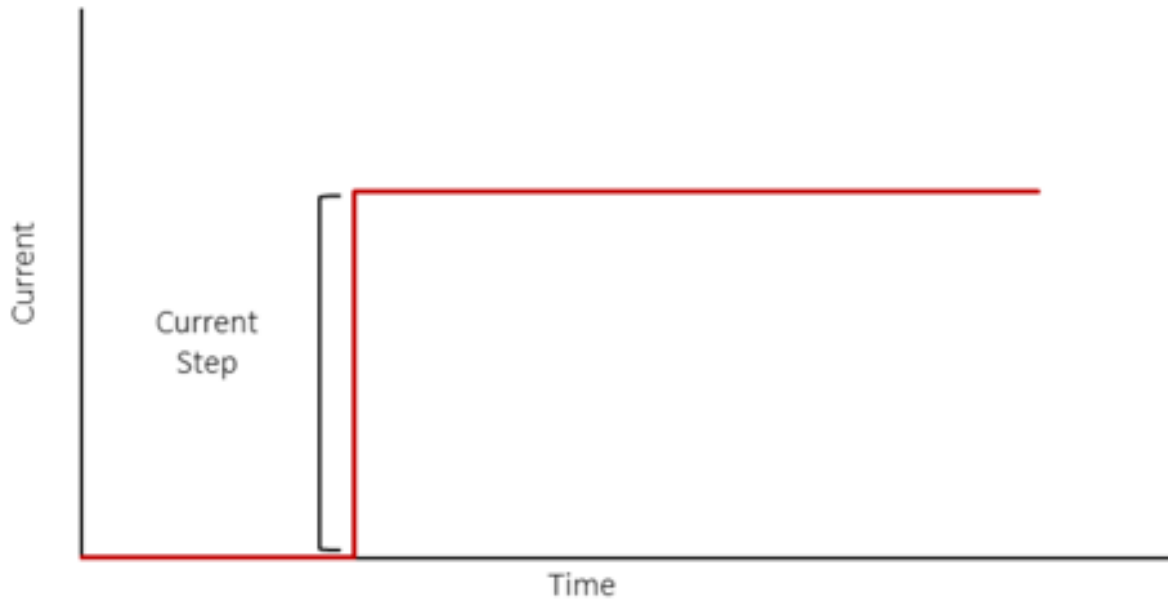
Public Member Functions

- [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.

11.3.1 Detailed Description

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



11.3.2 Constructor & Destructor Documentation

11.3.2.1 AisConstantCurrentElement()

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

the constant current element constructor.

Parameters

<i>current</i>	the value for the current in Amps.
<i>samplingInterval</i>	the data sampling interval value in seconds.
<i>duration</i>	the maximum duration for the experiment in seconds.

11.3.3 Member Function Documentation

11.3.3.1 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.

11.3.3.2 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.

11.3.3.3 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").

11.3.3.4 `getCurrent()`

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

get the value set for the current.

Returns

the value for the current in Amps.

11.3.3.5 `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.

11.3.3.6 `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.

11.3.3.7 `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

11.3.3.8 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.

11.3.3.9 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

11.3.3.10 getName()

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

get the name of the element.

Returns

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

11.3.3.11 getSamplingInterval()

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

get how frequently we are sampling the data.

Returns

the data sampling interval value in seconds.

11.3.3.12 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.

11.3.3.13 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.

11.3.3.14 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

<i>approxMaxCurrent</i>	the value for the maximum current expected in Amps.
-------------------------	-----------------------------------------------------

11.3.3.15 setApproxMaxVoltage()

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

set maximum voltage expected, for manual voltage range selection.

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

Parameters

<i>approxMaxVoltage</i>	the value for the maximum current expected in V.
-------------------------	--------------------------------------------------

11.3.3.16 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.

11.3.3.17 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.

11.3.3.18 setCurrent()

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

set the value for the current.

Parameters

<i>current</i>	the value for the current in Amps.
----------------	------------------------------------

11.3.3.19 setMaxCapacity()

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

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

This 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

<i>maxCapacity</i>	the value to set for the cell maximum capacity.
--------------------	-------------------------------------------------

11.3.3.20 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

<i>maxDuration</i>	the maximum duration for the experiment in seconds.
--------------------	-----------------------------------------------------

11.3.3.21 setMaxVoltage()

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

set a maximum voltage to stop the experiment.

This 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

<i>maxVoltage</i>	the maximum voltage value in volts at which the experiment will stop.
-------------------	-----------------------------------------------------------------------

11.3.3.22 setMinSamplingVoltageDifference()

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

set a minimum sampling voltage difference for reporting the voltage.

This 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

<i>minVoltageDifference</i>	the minimum sampling voltage difference value in volts.
-----------------------------	---------------------------------------------------------

11.3.3.23 setMinVoltage()

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

set a minimum voltage to stop the experiment.

This is an **optional** condition. If nothing is set, then the experiment will not stop based on a 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

<i>minVoltage</i>	the minimum voltage value in volts at which the experiment will stop.
-------------------	-----------------------------------------------------------------------

11.3.3.24 setSamplingInterval()

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

set how frequently we are sampling the data.

Parameters

<i>samplingInterval</i>	the data sampling interval value in seconds.
-------------------------	----------------------------------------------

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

- AisConstantCurrentElement.h

11.4 AisConstantPotElement Class Reference

an experiment that simulates a constant applied voltage.

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

Inherits AisAbstractElement.

Public Member Functions

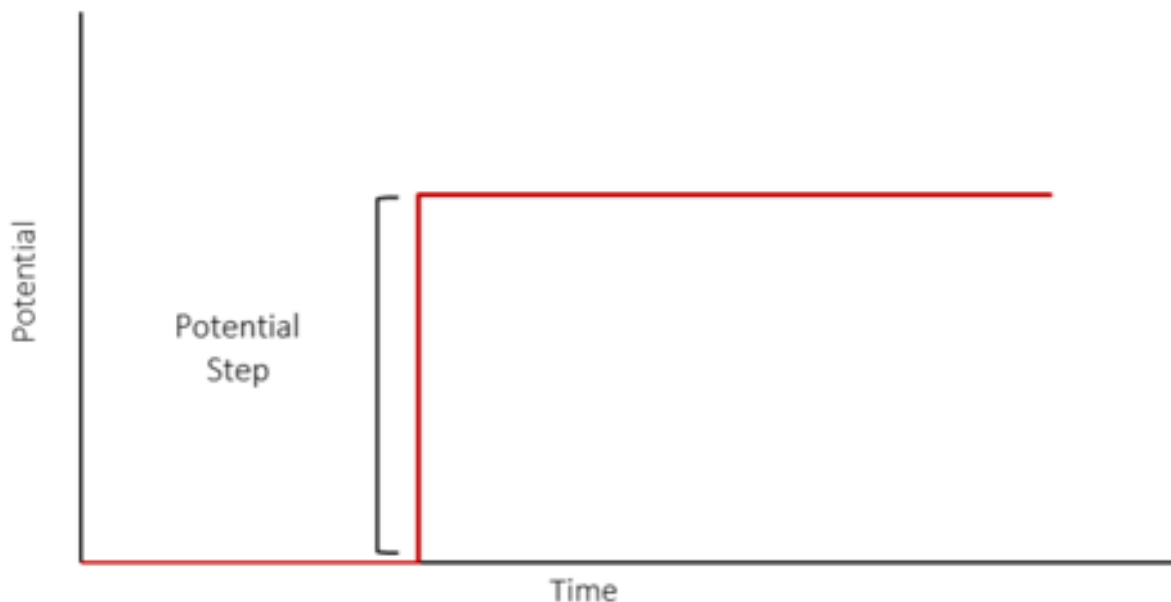
- [AisConstantPotElement](#) (double voltage, double samplingInterval, double duration)
the constant potential element constructor.
- **AisConstantPotElement** (const [AisConstantPotElement](#) &)
copy constructor for the [AisConstantPotElement](#) object.
- [AisConstantPotElement](#) & **operator=** (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.

11.4.1 Detailed Description

an experiment that simulates a constant applied voltage.



11.4.2 Constructor & Destructor Documentation

11.4.2.1 AisConstantPotElement()

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

the constant potential element constructor.

Parameters

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

11.4.3 Member Function Documentation

11.4.3.1 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.

11.4.3.2 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")

11.4.3.3 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.

11.4.3.4 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.

11.4.3.5 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.

11.4.3.6 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.

11.4.3.7 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.

11.4.3.8 getName()

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

get the name of the element.

Returns

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

11.4.3.9 getPotential()

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

get the value set for the potential in volts.

Returns

the value set for the potential in volts.

11.4.3.10 getSamplingInterval()

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

get how frequently we are sampling the data.

Returns

the data sampling interval value in seconds.

11.4.3.11 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 range has been selected.

11.4.3.12 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

11.4.3.13 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

<i>approxMaxCurrent</i>	the value for the maximum current expected in Amps.
-------------------------	-----------------------------------------------------

11.4.3.14 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.

11.4.3.15 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

<i>maxCapacity</i>	the value to set for the cell maximum capacity.
--------------------	-------------------------------------------------

11.4.3.16 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

<i>maxCurrent</i>	the maximum current value in Amps.
-------------------	------------------------------------

11.4.3.17 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

<i>maxDuration</i>	the maximum duration for the experiment in seconds.
--------------------	-----------------------------------------------------

11.4.3.18 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

<i>minCurrent</i>	the value to set for the absolute minimum current.
-------------------	----------------------------------------------------

11.4.3.19 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

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

11.4.3.20 setPotential()

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

set the value for the potential in volts.

Parameters

<i>potential</i>	the value to set for the potential in volts.
------------------	----------------------------------------------

11.4.3.21 setSamplingInterval()

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

set how frequently we are sampling the data.

Parameters

<i>samplingInterval</i>	the data sampling interval value in seconds.
-------------------------	----------------------------------------------

11.4.3.22 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

<i>vsOCP</i>	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

11.5 AisConstantPowerElement Class Reference

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

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

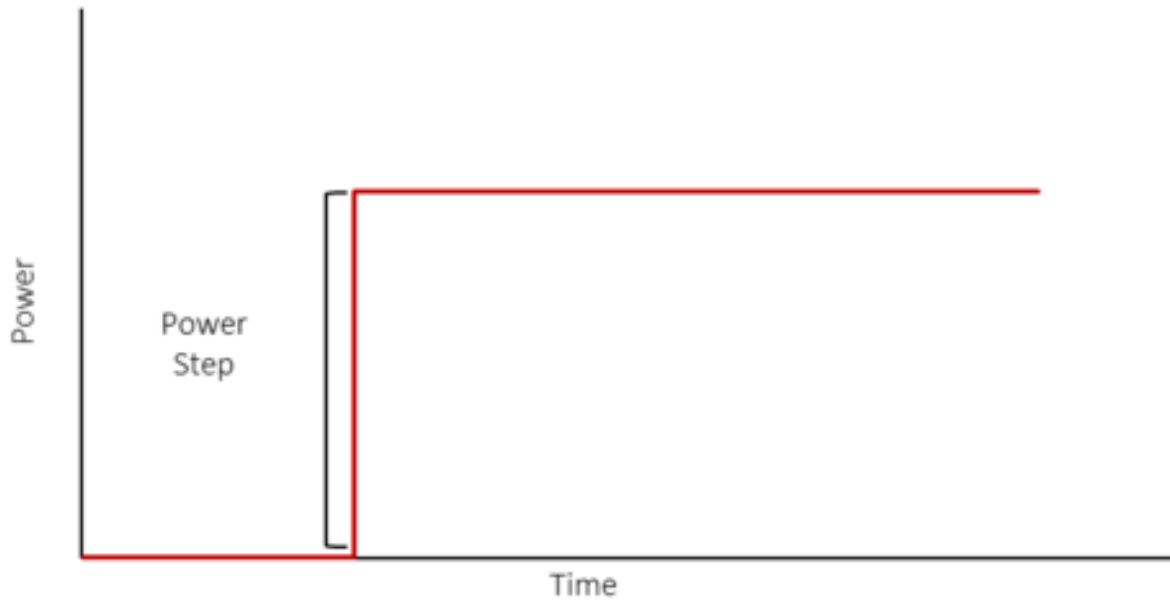
Inherits AisAbstractElement.

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.

11.5.1 Detailed Description

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



11.5.2 Constructor & Destructor Documentation

11.5.2.1 AisConstantPowerElement()

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

the constant power element constructor

Parameters

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

11.5.3 Member Function Documentation

11.5.3.1 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").

11.5.3.2 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.

11.5.3.3 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.

11.5.3.4 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

11.5.3.5 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

11.5.3.6 getName()

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

get the name of the element.

Returns

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

11.5.3.7 `getPower()`

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

get the value set for the power.

Returns

the value set for the power in watts.

11.5.3.8 `getSamplingInterval()`

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

get how frequently we are sampling the data.

Returns

the data sampling interval value in seconds.

11.5.3.9 `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.

11.5.3.10 `setCharge()`

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

set whether the experiment is to simulate charge or discharge.

Parameters

<i>isCharge</i>	if the given argument is true, the experiment will simulate charge and discharge if given false.
-----------------	--------------------------------------------------------------------------------------------------

11.5.3.11 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

<i>maxCapacity</i>	the value to set for the cell maximum capacity.
--------------------	-------------------------------------------------

11.5.3.12 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

<i>maxDuration</i>	the maximum duration for the experiment in seconds.
--------------------	-----------------------------------------------------

11.5.3.13 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

<i>maxVoltage</i>	the maximum voltage value in volts at which the experiment will stop.
-------------------	-----------------------------------------------------------------------

11.5.3.14 setMinVoltage()

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

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

Parameters

<i>minVoltage</i>	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.

11.5.3.15 setPower()

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

set the value for the power.

Parameters

<i>power</i>	the value set for the power in watts.
--------------	---------------------------------------

11.5.3.16 setSamplingInterval()

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

set how frequently we are sampling the data.

Parameters

<i>samplingInterval</i>	the data sampling interval value in seconds.
-------------------------	----------------------------------------------

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

- AisConstantPowerElement.h

11.6 AisConstantResistanceElement Class Reference

This element/experiment simulates a constant resistance load.

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

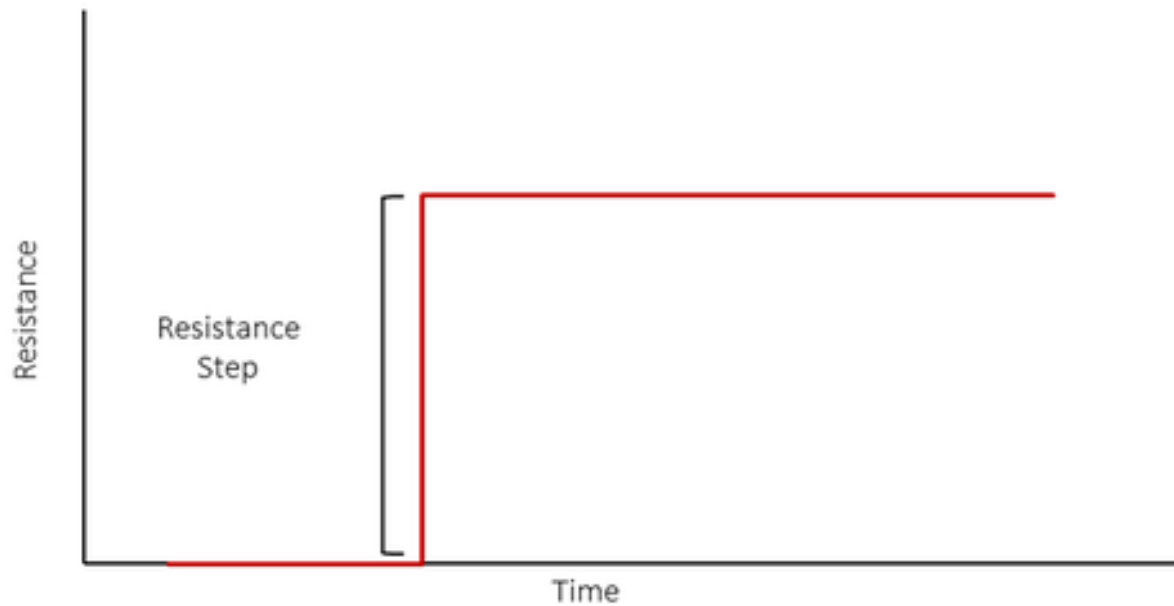
Inherits AisAbstractElement.

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.

11.6.1 Detailed Description

This element/experiment simulates a constant resistance load.



11.6.2 Constructor & Destructor Documentation

11.6.2.1 AisConstantResistanceElement()

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

the constant resistance element constructor.

Parameters

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

11.6.3 Member Function Documentation

11.6.3.1 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").

11.6.3.2 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.

11.6.3.3 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.

11.6.3.4 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

11.6.3.5 getName()

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

get the name of the element.

Returns

The name of the element: "Constant Resistance".

11.6.3.6 getResistance()

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

get the value set for the resistance as a load.

Returns

the value in ohm of the load resistance.

11.6.3.7 getSamplingInterval()

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

get how frequently we are sampling the data.

Returns

the data sampling interval value in seconds.

11.6.3.8 setMaxCapacity()

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

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

This 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

<i>maxCapacity</i>	the value to set for the cell maximum capacity.
--------------------	-------------------------------------------------

11.6.3.9 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

<i>maxDuration</i>	the maximum duration for the experiment in seconds.
--------------------	-----------------------------------------------------

11.6.3.10 setMinVoltage()

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

set a minimum voltage to stop the experiment.

There is an **optional** condition. If nothing is set, then the experiment will not stop based on a 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

<i>minVoltage</i>	the minimum voltage value in volts at which the experiment will stop.
-------------------	-----------------------------------------------------------------------

11.6.3.11 setResistance()

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

set the value for the resistance as a load

Parameters

<i>resistance</i>	the value in ohm of the load resistance.
-------------------	------------------------------------------

11.6.3.12 setSamplingInterval()

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

set how frequently we are sampling the data.

Parameters

<i>samplingInterval</i>	the data sampling interval value in seconds.
-------------------------	----------------------------------------------

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

- AisConstantResistanceElement.h

11.7 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.*

11.7.1 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.

11.7.2 Constructor & Destructor Documentation

11.7.2.1 AisCyclicVoltammetryElement()

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

constructor of the cyclic voltammetry element.

Parameters

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

11.7.3 Member Function Documentation

11.7.3.1 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.

11.7.3.2 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").

11.7.3.3 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.

11.7.3.4 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.

11.7.3.5 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.

11.7.3.6 getName()

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

get the name of the element.

Returns

The name of the element: "Cyclic Voltammetry".

11.7.3.7 getNumberOfCycles()

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

get the value set for the number of cycles

Returns

the number of cycles set.

11.7.3.8 getSamplingInterval()

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

get how frequently we are sampling the data.

Returns

the data sampling interval value in seconds.

11.7.3.9 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.

11.7.3.10 getStartVoltage()

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

get the value set for the start voltage

Returns

the value of the start voltage in volts

11.7.3.11 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 range has been selected.

11.7.3.12 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

11.7.3.13 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.

11.7.3.14 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.

11.7.3.15 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.

11.7.3.16 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

<i>approxMaxCurrent</i>	the value for the maximum current expected in Amps.
-------------------------	-----------------------------------------------------

11.7.3.17 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.

11.7.3.18 setdEdt()

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

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

Parameters

<i>dEdt</i>	the value set for the constant scan rate dE/dt in V/s.
-------------	--------------------------------------------------------

11.7.3.19 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

<i>endVoltage</i>	the value to set for the ending potential in volts.
-------------------	-----------------------------------------------------

11.7.3.20 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

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

11.7.3.21 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

<i>v1</i>	first voltage-limit value in volts
-----------	------------------------------------

11.7.3.22 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

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

11.7.3.23 setNumberOfCycles()

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

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

Parameters

<i>cycles</i>	the number of cycles to set
---------------	-----------------------------

11.7.3.24 setSamplingInterval()

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

set how frequently we are sampling the data.

Parameters

<i>sampInterval</i>	the data sampling interval value in seconds.
---------------------	----------------------------------------------

11.7.3.25 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

<i>v2</i>	the second voltage-limit value in volts
-----------	-----------------------------------------

11.7.3.26 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

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

11.7.3.27 setStartVoltage()

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

set the value for the start voltage.

Parameters

<code>startVoltage</code>	the value of the start voltage in volts
---------------------------	-----------------------------------------

11.7.3.28 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

<code>startVoltageVsOCP</code>	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`

11.8 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

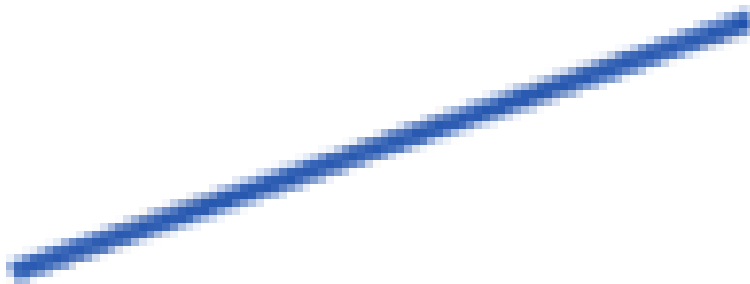
- [AisDCCurrentSweepElement](#) (double startCurrent, double endCurrent, double scanRate, double samplingInterval)
the DC current sweep element.
- **AisDCCurrentSweepElement** (const [AisDCCurrentSweepElement](#) &)
copy constructor for the [AisDCCurrentSweepElement](#) object.
- [AisDCCurrentSweepElement](#) & **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.

11.8.1 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**.

DC Current Linear Sweep



11.8.2 Constructor & Destructor Documentation

11.8.2.1 AisDCCurrentSweepElement()

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

the DC current sweep element.

Parameters

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

11.8.3 Member Function Documentation

11.8.3.1 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").

11.8.3.2 getEndingCurrent()

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

get the value set for the ending current.

Returns

the value for the ending current in Amps.

11.8.3.3 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

11.8.3.4 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

11.8.3.5 getName()

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

get the name of the element.

Returns

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

11.8.3.6 getSamplingInterval()

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

get how frequently we are sampling the data.

Returns

the data sampling interval value in seconds.

11.8.3.7 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](#)

11.8.3.8 getStartingCurrent()

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

get the value set for the starting current.

Returns

the value set for the constant current in Amps.

11.8.3.9 setEndingCurrent()

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

set the value for the ending current.

Parameters

<i>endingCurrent</i>	the value for the ending current in Amps
----------------------	------------------------------------------

11.8.3.10 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

<i>maxVoltage</i>	the maximum voltage value in volts at which the experiment will stop.
-------------------	-----------------------------------------------------------------------

11.8.3.11 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

<i>minVoltage</i>	the minimum voltage value in volts at which the experiment will stop.
-------------------	-----------------------------------------------------------------------

11.8.3.12 setSamplingInterval()

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

set how frequently we are sampling the data.

Parameters

<i>samplingInterval</i>	the data sampling interval value in seconds.
-------------------------	----------------------------------------------

11.8.3.13 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

<i>scanRate</i>	the value to set for the scan rate.
-----------------	-------------------------------------

11.8.3.14 setStartingCurrent()

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

set the value for the starting current.

Parameters

<i>startingCurrent</i>	the value to set for the starting current in Amps
------------------------	---------------------------------------------------

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

- AisDCCurrentSweepElement.h

11.9 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.

11.9.1 Detailed Description

a structure containing DC data information.

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

- AisDataPoints.h

11.10 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>
```

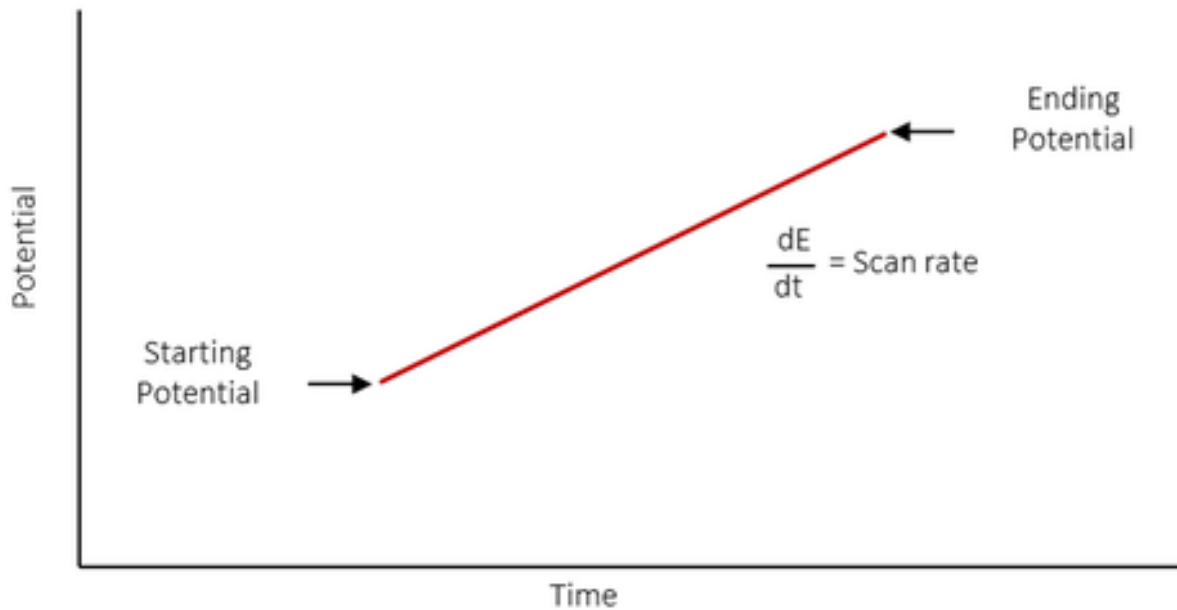
Inherits AisAbstractElement.

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.

11.10.1 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**.



11.10.2 Constructor & Destructor Documentation

11.10.2.1 AisDCPotentialSweepElement()

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

the potential sweep element constructor.

Parameters

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

11.10.3 Member Function Documentation

11.10.3.1 `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.

11.10.3.2 `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").

11.10.3.3 `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.

11.10.3.4 getName()

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

get the name of the element.

Returns

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

11.10.3.5 getSamplingInterval()

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

get how frequently we are sampling the data.

Returns

the data sampling interval value in seconds.

11.10.3.6 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](#)

11.10.3.7 getStartingPot()

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

get the value set for the starting potential.

Returns

the value of the starting potential in volts.

11.10.3.8 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 range has been selected.

11.10.3.9 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](#)

11.10.3.10 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](#)

11.10.3.11 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

<i>approxMaxCurrent</i>	the value for the maximum current expected in Amps.
-------------------------	-----------------------------------------------------

11.10.3.12 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.

11.10.3.13 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

<i>endingPotential</i>	the value to set for the ending potential in volts.
------------------------	-----------------------------------------------------

11.10.3.14 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

<i>endVoltageVsOCP</i>	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.

11.10.3.15 setSamplingInterval()

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

set how frequently we are sampling the data.

Parameters

<i>samplingInterval</i>	the data sampling interval value in seconds.
-------------------------	----------------------------------------------

11.10.3.16 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

<i>scanRate</i>	the value to set for the scan rate.
-----------------	-------------------------------------

11.10.3.17 setStartingPot()

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

set the value for the starting potential.

Parameters

<i>startingPotential</i>	the value of the starting potential in volts
--------------------------	----------------------------------------------

11.10.3.18 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

<code>startVoltageVsOCP</code>	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:

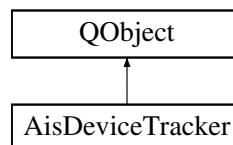
- AisDCPotentialSweepElement.h

11.11 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 ()

11.11.1 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.

11.11.2 Member Function Documentation

11.11.2.1 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.

11.11.2.2 connectToDeviceOnComPort()

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

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

Parameters

<i>comPort</i>	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".

11.11.2.3 deviceDisconnected

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

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

Parameters

<i>deviceName</i>	the name of the newly disconnected device.
-------------------	--------------------------------------------

11.11.2.4 getConnectedDevices()

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

get a list of all the connected devices.

Returns

a list of all the connected devices.

11.11.2.5 getInstrumentHandler()

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

get an instrument handler to control a specific device.

Parameters

<i>deviceName</i>	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\(\)](#)

11.11.2.6 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

<i>deviceName</i>	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.

11.11.2.7 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](#)

11.11.2.8 updateFirmwareOnComPort()

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

update firmware on connected device at USB port.

Parameters

<i>comPort</i>	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

11.12 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.

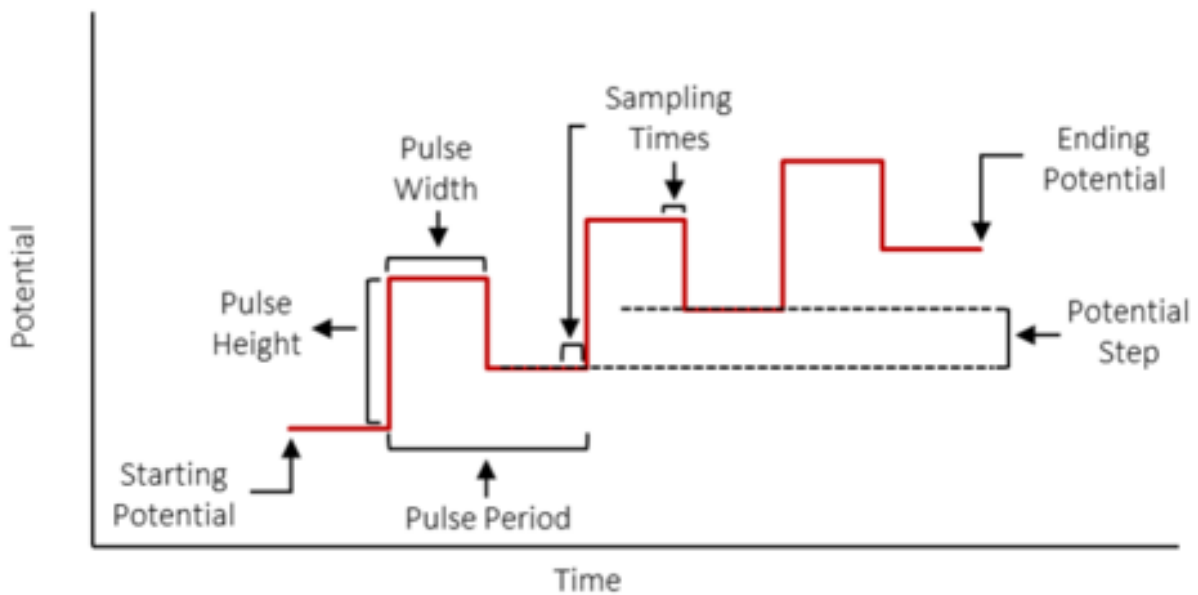
Public Member Functions

- [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.

11.12.1 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.



11.12.2 Constructor & Destructor Documentation

11.12.2.1 AisDiffPulseVoltammetryElement()

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

the differential pulse element constructor.

Parameters

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

11.12.3 Member Function Documentation

11.12.3.1 `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.

11.12.3.2 `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").

11.12.3.3 `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.

11.12.3.4 getName()

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

get the name of the element.

Returns

The name of the element: "Differential Pulse Potential Voltammetry".

11.12.3.5 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](#)

11.12.3.6 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](#)

11.12.3.7 `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](#)

11.12.3.8 `getStartVoltage()`

```
double AisDiffPulseVoltammetryElement::getStartVoltage ( ) const
```

get the value set for the start voltage.

Returns

the value of the start voltage in volts.

11.12.3.9 `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](#)

11.12.3.10 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 range has been selected.

11.12.3.11 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](#)

11.12.3.12 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](#)

11.12.3.13 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

<i>approxMaxCurrent</i>	the value for the maximum current expected in Amps.
-------------------------	-----------------------------------------------------

11.12.3.14 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.

11.12.3.15 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

<i>endVoltage</i>	the value to set for the ending voltage in volts.
-------------------	---------------------------------------------------

11.12.3.16 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

<i>endVoltageVsOCP</i>	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.

11.12.3.17 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

<i>pulseHeight</i>	the value to set for the pulse height in volts.
--------------------	-------------------------------------------------

11.12.3.18 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

<i>pulsePeriod</i>	the value to set for the pulse period in seconds.
--------------------	---------------------------------------------------

11.12.3.19 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

<i>pulseWidth</i>	the value to set for the pulse width in seconds.
-------------------	--------------------------------------------------

See also

[setPulseHeight](#)

11.12.3.20 setStartVoltage()

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

set the value for the start voltage.

Parameters

<i>startVoltage</i>	the value of the start voltage in volts
---------------------	-----------------------------------------

11.12.3.21 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

<i>startVoltageVsOCP</i>	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.

11.12.3.22 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

<i>vStep</i>	the value to set for the potential step in volts.
--------------	---------------------------------------------------

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

- AisDiffPulseVoltammetryElement.h

11.13 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.

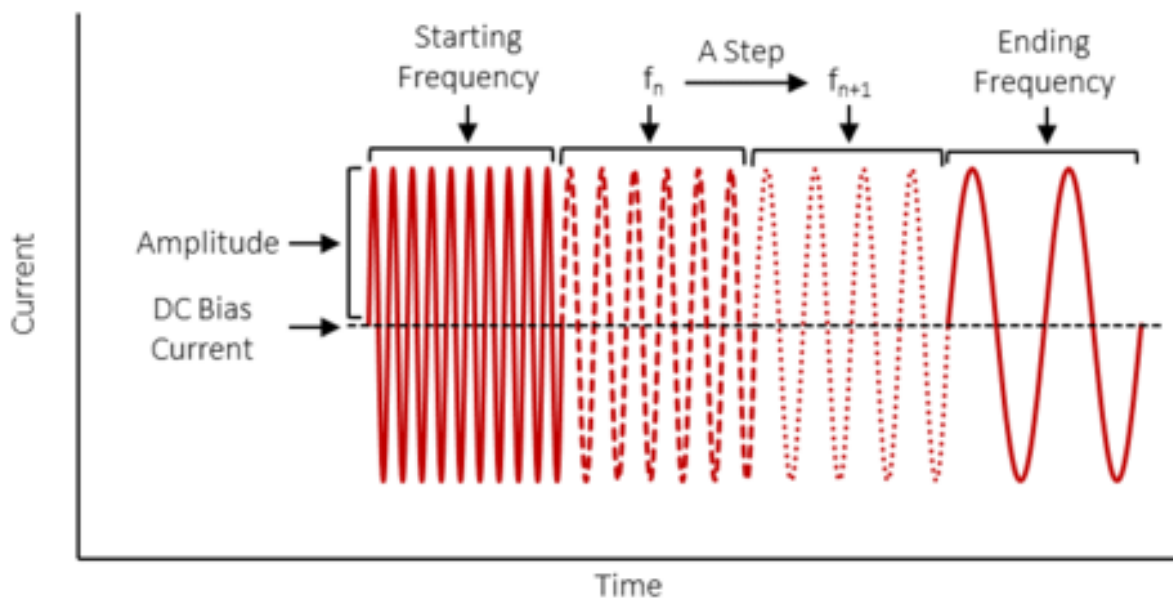
Public Member Functions

- [AisEISGalvanostaticElement](#) (double startFrequency, double endFrequency, double stepsPerDecade, double currentBias, double currentAmplitude)
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.

11.13.1 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**.



11.13.2 Constructor & Destructor Documentation

11.13.2.1 AisEISGalvanostaticElement()

```
AisEISGalvanostaticElement::AisEISGalvanostaticElement (
    double startFrequency,
    double endFrequency,
    double stepsPerDecade,
    double currentBias,
    double currentAmplitude ) [explicit]
```

the EIS galvanostatic element constructor.

Parameters

<i>startFrequency</i>	the value for the current starting frequency
<i>endFrequency</i>	the value for the current ending frequency
<i>stepsPerDecade</i>	the value for the current frequency steps per decade.
<i>currentBias</i>	the value for the DC bias (DC offset).
<i>currentAmplitude</i>	the AC current amplitude.

11.13.3 Member Function Documentation

11.13.3.1 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.

11.13.3.2 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.

11.13.3.3 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").

11.13.3.4 getEndFreq()

```
double AisEISGalvanostaticElement::getEndFreq ( ) const
```

the value set for the current ending frequency.

Returns

the value set for the current end frequency in Hz

11.13.3.5 getName()

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

get the name of the element.

Returns

The name of the element: "Galvanostatic EIS".

11.13.3.6 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?

11.13.3.7 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.

11.13.3.8 setAmplitude()

```
void AisEISGalvanostaticElement::setAmplitude (
    double amplitude )
```

set the value for the AC current amplitude.

Parameters

<i>amplitude</i>	the value to set for the AC current amplitude in Amps.
------------------	--------------------------------------------------------

11.13.3.9 setBiasCurrent()

```
void AisEISGalvanostaticElement::setBiasCurrent (
    double biasCurrent )
```

set the value for the DC bias (DC offset).

Parameters

<i>biasCurrent</i>	the value to set for the DC bias in Amps.
--------------------	-------------------------------------------

11.13.3.10 setEndFreq()

```
void AisEISGalvanostaticElement::setEndFreq (
    double endFreq )
```

set the value for the current end frequency.

Parameters

<i>endFreq</i>	the value to set for the current end frequency in Hz
----------------	------------------------------------------------------

11.13.3.11 setStartFreq()

```
void AisEISGalvanostaticElement::setStartFreq (
    double startFreq )
```

set the value for the current starting frequency.

Parameters

<i>startFreq</i>	the value to set the current starting frequency in Hz
------------------	-------------------------------------------------------

11.13.3.12 setStepsPerDecade()

```
void AisEISGalvanostaticElement::setStepsPerDecade (
    double stepsPerDecade )
```

set the number of the current frequency steps per decade.

Parameters

<i>stepsPerDecade</i>	the value to set for the number of steps per decade.
-----------------------	------------------------------------------------------

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

- AisEISGalvanostaticElement.h

11.14 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

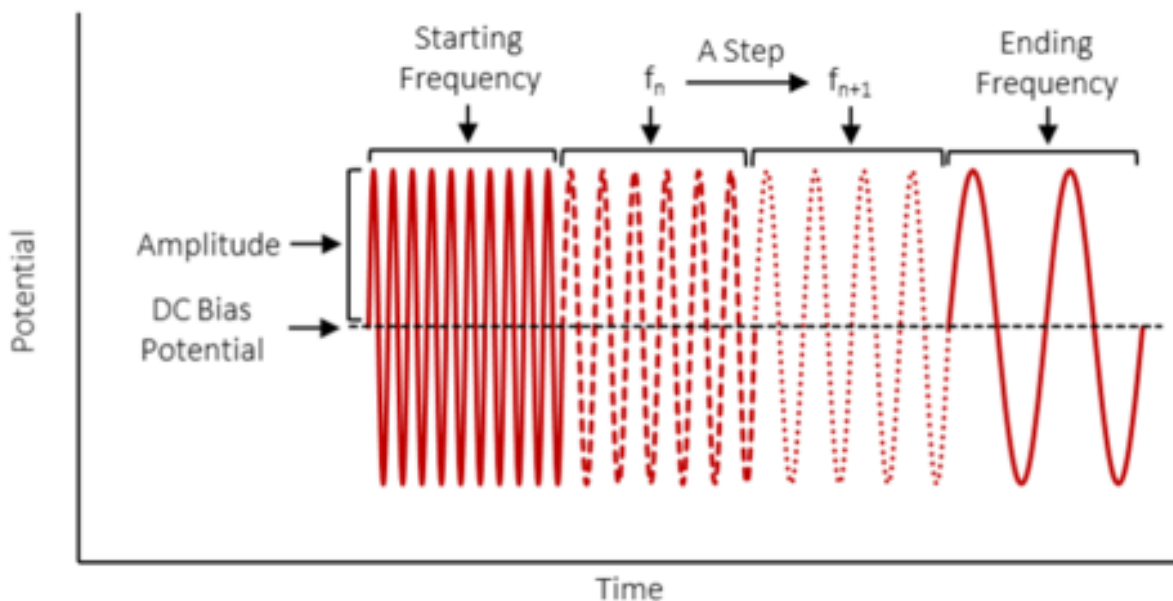
- [AisEISPotentiostaticElement](#) (double startFrequency, double endFrequency, double stepsPerDecade, double voltageBias, double voltageAmplitude)
the EIS potentiostatic element
- [AisEISPotentiostaticElement](#) (const [AisEISPotentiostaticElement](#) &)
copy constructor for the [AisEISPotentiostaticElement](#) 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.

11.14.1 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**.



11.14.2 Constructor & Destructor Documentation

11.14.2.1 AisEISPotentiostaticElement()

```
AisEISPotentiostaticElement::AisEISPotentiostaticElement (
    double startFrequency,
    double endFrequency,
    double stepsPerDecade,
    double voltageBias,
    double voltageAmplitude ) [explicit]
```

the EIS potentiostatic element

Parameters

<i>startFrequency</i>	the value for the voltage starting frequency
<i>endFrequency</i>	the value for the voltage ending frequency
<i>stepsPerDecade</i>	the value for the voltage frequency steps per decade.
<i>voltageBias</i>	the value for the DC bias (DC offset).
<i>voltageAmplitude</i>	the AC voltage amplitude.

11.14.3 Member Function Documentation

11.14.3.1 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.

11.14.3.2 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.

11.14.3.3 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").

11.14.3.4 getEndFreq()

```
double AisEISPotentiostaticElement::getEndFreq ( ) const
```

the value set for the voltage ending frequency.

Returns

the value set for the voltage end frequency in Hz

11.14.3.5 getName()

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

get the name of the element.

Returns

The name of the element: "Potentiostatic EIS".

11.14.3.6 getStartFreq()

```
double AisEISPotentiostaticElement::getStartFreq ( ) const
```

get the value set for the voltage starting frequency

Returns

the value set for the start frequency in Hz

11.14.3.7 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.

11.14.3.8 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.

11.14.3.9 setAmplitude()

```
void AisEISPotentiostaticElement::setAmplitude (
    double amplitude )
```

set the value for the AC voltage amplitude.

Parameters

<i>amplitude</i>	the value to set for the AC voltage amplitude in volts.
------------------	---------------------------------------------------------

11.14.3.10 setBiasVoltage()

```
void AisEISPotentiostaticElement::setBiasVoltage (
    double biasVoltage )
```

set the value for the DC bias (DC offset).

Parameters

<i>biasVoltage</i>	the value to set for the DC bias in volts.
--------------------	--------------------------------------------

11.14.3.11 setBiasVoltageVsOCP()

```
void AisEISPotentiostaticElement::setBiasVoltageVsOCP (
    bool biasVsOCP )
```

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

Parameters

<i>biasVsOCP</i>	true to if the DC-bias voltage is set to reference the open-circuit voltage and false if set against the reference terminal.
------------------	------------------------------------------------------------------------------------------------------------------------------

11.14.3.12 setEndFreq()

```
void AisEISPotentiostaticElement::setEndFreq (
    double endFreq )
```

set the value for the voltage end frequency.

Parameters

<i>endFreq</i>	the value to set for the voltage end frequency in Hz
----------------	------------------------------------------------------

11.14.3.13 setStartFreq()

```
void AisEISPotentiostaticElement::setStartFreq (
    double startFreq )
```

set the value for the voltage starting frequency.

Parameters

<i>startFreq</i>	the value to set the starting frequency Hz
------------------	--------------------------------------------

11.14.3.14 setStepsPerDecade()

```
void AisEISPotentiostaticElement::setStepsPerDecade (
    double stepsPerDecade )
```

set the number of the voltage frequency steps per decade.

Parameters

<i>stepsPerDecade</i>	the value to set for the number of steps per decade.
-----------------------	------------------------------------------------------

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

- AisEISPotentiostaticElement.h

11.15 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

11.15.1 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](#).

11.15.2 Member Enumeration Documentation

11.15.2.1 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.

11.15.3 Member Function Documentation**11.15.3.1 message()**

```
QString AISErrorCode::message ( ) const
```

a function to get a message explaining the error.

Returns

a message that explains the error.

11.15.3.2 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

11.16 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**

11.16.1 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.

11.16.2 Constructor & Destructor Documentation

11.16.2.1 AisExperiment()

```
AisExperiment::AisExperiment (
    const AisExperiment & exp ) [explicit]
```

this is the copy constructor for the custom experiment.

Parameters

<i>exp</i>	the custom experiment to copy from.
------------	-------------------------------------

11.16.3 Member Function Documentation**11.16.3.1 appendElement()**

```
bool AisExperiment::appendElement (
    AisAbstractElement & element,
    uint repeat = 1 )
```

append an element to the custom experiment.

Parameters

<i>element</i>	an elemental experiment to be appended to this/(the calling) custom experiment.
<i>repeat</i>	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.

11.16.3.2 appendSubExperiment()

```
bool AisExperiment::appendSubExperiment (
    const AisExperiment & subExp,
    uint repeat = 1 )
```

append a sub experiment to this/(the calling) custom experiment.

Parameters

<i>subExp</i>	a sub experiment to be appended to this/(the calling) custom experiment.
<i>repeat</i>	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.

11.16.3.3 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").

11.16.3.4 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".

11.16.3.5 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"

11.16.3.6 operator=()

```
void AisExperiment::operator= (
    const AisExperiment & exp )
```

the assignment operator for the custom experiment.

Parameters

<i>exp</i>	the custom experiment to copy from.
------------	-------------------------------------

11.16.3.7 setDescription()

```
void AisExperiment::setDescription (
    QString description )
```

set a description for the experiment.

Parameters

<i>description</i>	the description to be set for the custom experiment.
--------------------	------------------------------------------------------

11.16.3.8 setExperimentName()

```
void AisExperiment::setExperimentName (
    QString name )
```

set a name for the custom experiment.

Parameters

<i>name</i>	the name to be set for the custom experiment.
-------------	-----------------------------------------------

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

- AisExperiment.h

11.17 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.

11.17.1 Detailed Description

a structure containing some information regarding the running element.

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

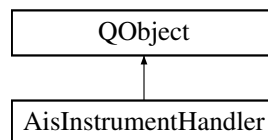
- AisDataPoints.h

11.18 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.
- 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.

11.18.1 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.

11.18.2 Member Function Documentation

11.18.2.1 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

<i>channel</i>	the channel number from which the AC data arrived.
<i>ACData</i>	the AC data that just arrived.

11.18.2.2 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

<i>channel</i>	the channel number from which the DC data arrived.
<i>DCData</i>	the DC data that just arrived.

11.18.2.3 deviceDisconnected

```
void AisInstrumentHandler::deviceDisconnected ( ) [signal]
```

a signal that is emitted if the device associated with this handler has been disconnected.

11.18.2.4 eraseRecoverData()

```
AisErrorCode AisInstrumentHandler::eraseRecoverData ( ) const
```

delete the recover data from device.

Returns

[AisErrorCode::Success](#) if request is successfully send for delete the data. If not successful, possible returned errors are:

- [AisErrorCode::DeviceNotFound](#)
- [AisErrorCode::DeviceCommunicationFailed](#)

11.18.2.5 experimentNewElementStarting

```
void AisInstrumentHandler::experimentNewElementStarting (
    uint8_t channel,
    const AisExperimentNode & stepInfo ) [signal]
```

a signal that is emitted whenever a new elemental experiment has started.

Parameters

<i>channel</i>	the channel number on which the experiment was started.
<i>stepInfo</i>	information regarding the current step.

See also

[AisExperimentNode](#)

11.18.2.6 experimentPaused

```
void AisInstrumentHandler::experimentPaused (
    uint8_t channel ) [signal]
```

a signal that is emitted whenever an experiment was paused.

Parameters

<i>channel</i>	the channel on which the experiment was paused.
----------------	-------------------------------------------------

11.18.2.7 experimentResumed

```
void AisInstrumentHandler::experimentResumed (
    uint8_t channel ) [signal]
```

a signal that is emitted whenever an experiment was resumed.

Parameters

<i>channel</i>	the channel on which the experiment was resumed.
----------------	--------------------------------------------------

11.18.2.8 experimentStopped

```
void AisInstrumentHandler::experimentStopped (
    uint8_t channel ) [signal]
```

a signal that is emitted whenever an experiment was stopped manually or has completed.

Parameters

<i>channel</i>	the channel on which the experiment has stopped.
----------------	--------------------------------------------------

11.18.2.9 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

<i>channel</i>	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.

11.18.2.10 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.

11.18.2.11 getLinkedChannels()

```
std::vector< uint8_t > AisInstrumentHandler::getLinkedChannels (
    uint8_t channel ) const
```

get a list of channels linked to the given channel.

Parameters

<i>channel</i>	a valid channel number to find which other channels are linked to it.
----------------	-----------------------------------------------------------------------

Returns

a list of channels linked to the channel parameter.

11.18.2.12 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

<i>channel</i>	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.

11.18.2.13 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.

11.18.2.14 groundFloatStateChanged

```
void AisInstrumentHandler::groundFloatStateChanged (
    bool grounded ) [signal]
```

a signal that is emitted when the floating ground connection state has changed.

Parameters

<i>grounded</i>	true if there is a connection to ground and false if the ground has disconnected.
-----------------	-----------------------------------------------------------------------------------

11.18.2.15 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

<i>channel</i>	the channel number from which the DC data arrived.
<i>DCData</i>	the DC data that just arrived.

11.18.2.16 isChannelBusy()

```
bool AisInstrumentHandler::isChannelBusy (
    uint8_t channel ) const
```

tells whether the given channel is busy or not.

Parameters

<i>channel</i>	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.

11.18.2.17 isChannelPaused()

```
bool AisInstrumentHandler::isChannelPaused (
    uint8_t channel ) const
```

tells whether the given channel has a paused experiment or not.

Parameters

<i>channel</i>	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.

11.18.2.18 pauseExperiment()

```
AisErrorCode AisInstrumentHandler::pauseExperiment (
    uint8_t channel ) const
```

pause a running experiment on the channel.

Parameters

<i>channel</i>	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.

11.18.2.19 recoverDataErased

```
void AisInstrumentHandler::recoverDataErased (
    bool successful ) [signal]
```

a signal that is emitted whenever data erase process is completed.

Parameters

<i>successful</i>	is true on erased correctly, and false on data is not erased.
-------------------	---------------------------------------------------------------

11.18.2.20 recoveryACDataReady

```
void AisInstrumentHandler::recoveryACDataReady (
    uint8_t channel,
    const AisACData & ACData ) [signal]
```

a signal that is emitted whenever new AC recovery data are ready.

Parameters

<i>channel</i>	the channel number from which the AC data are recovered from.
<i>ACData</i>	the AC data that just arrived.

11.18.2.21 recoveryDCDataReady

```
void AisInstrumentHandler::recoveryDCDataReady (
    uint8_t channel,
    const AisDCData & DCData ) [signal]
```

a signal that is emitted whenever new DC recovery data are ready.

Parameters

<i>channel</i>	the channel number from which the DC data are recovered from.
<i>DCData</i>	the DC data that just arrived.

11.18.2.22 resumeExperiment()

```
AisErrorCode AisInstrumentHandler::resumeExperiment (
    uint8_t channel ) const
```

resume a paused experiment on the channel.

Parameters

<i>channel</i>	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.

11.18.2.23 setCompRange()

```
AisErrorCode AisInstrumentHandler::setCompRange (
    uint8_t channel,
    const AisCompRange & compRange ) const
```

set a compensation range with stability factor and bandwidth index.

Parameters

<i>channel</i>	the channel for which to set the compensation range.
<i>compRange</i>	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](#)

11.18.2.24 setIRComp()

```
AisErrorCode AisInstrumentHandler::setIRComp (
    uint8_t channel,
    double uncompensatedResistance,
    double compensationLevel ) const
```

set IR compensation.

Parameters

<i>channel</i>	the channel for which to set the IR compensation.
<i>uncompensatedResistance</i>	the value of the uncompensated resistance in Ohms.
<i>compensationLevel</i>	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](#)

11.18.2.25 setLinkedChannels()

```
int8_t AisInstrumentHandler::setLinkedChannels (
    std::vector< uint8_t > channels ) const
```

connect several channels together.

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 cyclor model.

Parameters

<i>channels</i>	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 cyclor model.

11.18.2.26 setManualModeConstantCurrent()

```
AisErrorCode AisInstrumentHandler::setManualModeConstantCurrent (
    uint8_t channel,
    double value ) const
```

set constant current for the manual experiment.

Parameters

<i>channel</i>	a valid channel number to set a constant voltage for.
<i>value</i>	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](#)

11.18.2.27 setManualModeConstantVoltage() [1/2]

```
AisErrorCode AisInstrumentHandler::setManualModeConstantVoltage (
    uint8_t channel,
    double value ) const
```

set constant voltage for the manual experiment.

Parameters

<i>channel</i>	a valid channel number to set a constant voltage for.
<i>value</i>	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](#)

11.18.2.28 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

<i>channel</i>	a valid channel number to set a constant voltage for.
<i>value</i>	the value to set the constant voltage in volts.
<i>currentRangeIndex</i>	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](#)

11.18.2.29 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

<i>channel</i>	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](#)

11.18.2.30 setManualModeSamplingInterval()

```
AisErrorCode AisInstrumentHandler::setManualModeSamplingInterval (
    uint8_t channel,
    double value ) const
```

set an interval for sampling the data.

Parameters

<i>channel</i>	the channel to set the sampling interval for.
<i>value</i>	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](#)

11.18.2.31 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

<i>channel</i>	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](#)

11.18.2.32 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

<i>channel</i>	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](#)

11.18.2.33 startUploadedExperiment()

```
AisErrorCode AisInstrumentHandler::startUploadedExperiment (
    uint8_t channel ) const
```

start the previously uploaded experiment on the specific channel.

Parameters

<i>channel</i>	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](#)

11.18.2.34 stopExperiment()

```
AisErrorCode AisInstrumentHandler::stopExperiment (
    uint8_t channel ) const
```

stop a running or a paused experiment on the channel.

Parameters

<i>channel</i>	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](#)
- [AisErrorCode::ChannelNotBusy](#)

This will only return [AisErrorCode::Success](#) if there is currently a running or a paused experiment on a valid channel on a connected device.

11.18.2.35 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

<i>channel</i>	the channel number to upload the experiment to.
<i>experiment</i>	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](#)

11.18.2.36 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

<i>channel</i>	the channel number to upload the experiment to.
<i>experiment</i>	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

11.19 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](#).

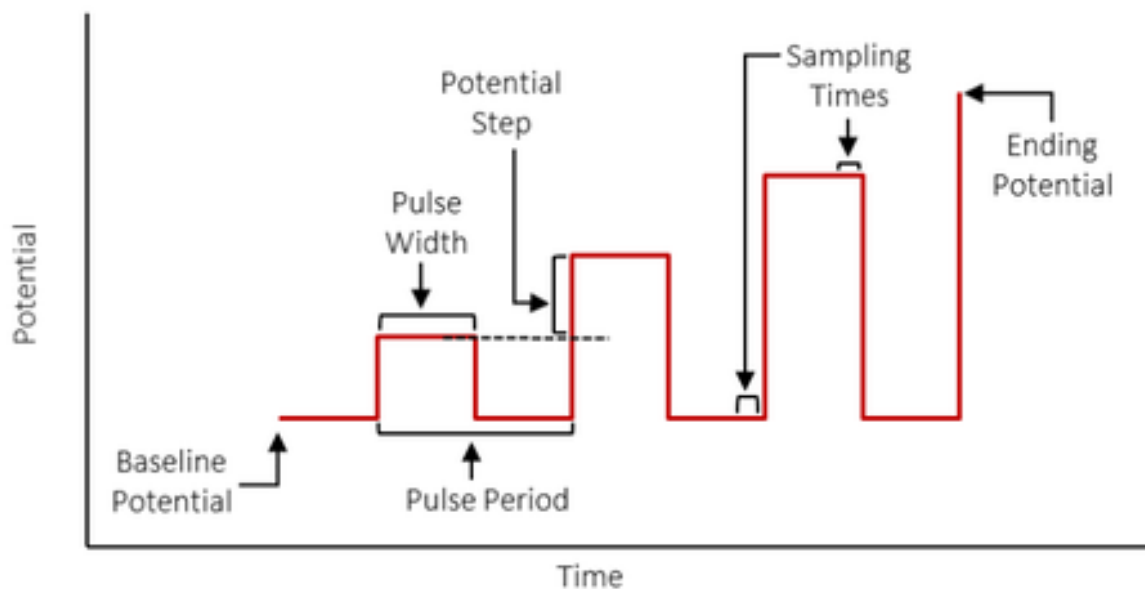
Public Member Functions

- [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.

11.19.1 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.



11.19.2 Constructor & Destructor Documentation

11.19.2.1 AisNormalPulseVoltammetryElement()

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

the normal-pulse-voltammetry element constructor

Parameters

<i>startVoltage</i>	the value of the starting potential in volts
<i>endVoltage</i>	the value of the ending potential in volts
<i>voltageStep</i>	the value set for the voltage step in volts.
<i>pulseWidth</i>	the value for the pulse width in volts.
<i>pulsePeriod</i>	the value for the pulse period in volts.

11.19.3 Member Function Documentation

11.19.3.1 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.

11.19.3.2 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").

11.19.3.3 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.

11.19.3.4 getName()

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

get the name of the element.

Returns

The name of the element: "Normal Pulse Potential Voltammetry".

11.19.3.5 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](#)

11.19.3.6 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](#)

11.19.3.7 getStartVoltage()

```
double AisNormalPulseVoltammetryElement::getStartVoltage ( ) const
```

get the value set for the start voltage.

Returns

the value of the start voltage in volts.

11.19.3.8 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](#)

11.19.3.9 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 range has been selected.

11.19.3.10 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.

11.19.3.11 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](#)

11.19.3.12 setApproxMaxCurrent()

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

set maximum current expected, for manual current range selection.

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

Parameters

<i>approxMaxCurrent</i>	the value for the maximum current expected in Amps.
-------------------------	-----------------------------------------------------

11.19.3.13 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.

11.19.3.14 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

<i>endVoltage</i>	the value to set for the ending potential in volts.
-------------------	-----------------------------------------------------

11.19.3.15 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

<i>endVoltageVsOcp</i>	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.

11.19.3.16 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

<i>pulsePeriod</i>	the value to set for the pulse period in seconds.
--------------------	---------------------------------------------------

11.19.3.17 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

<i>pulseWidth</i>	the value to set for the pulse width in seconds.
-------------------	--------------------------------------------------

11.19.3.18 setStartVoltage()

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

set the value for the start voltage.

Parameters

<i>startVoltage</i>	the value of the start voltage in volts
---------------------	-----------------------------------------

11.19.3.19 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

<i>startVoltageVsOCP</i>	true 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.

11.19.3.20 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

<code>vStep</code>	the value for the voltage step in volts.
--------------------	------------------------------------------

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

- `AisNormalPulseVoltammetryElement.h`

11.20 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`.

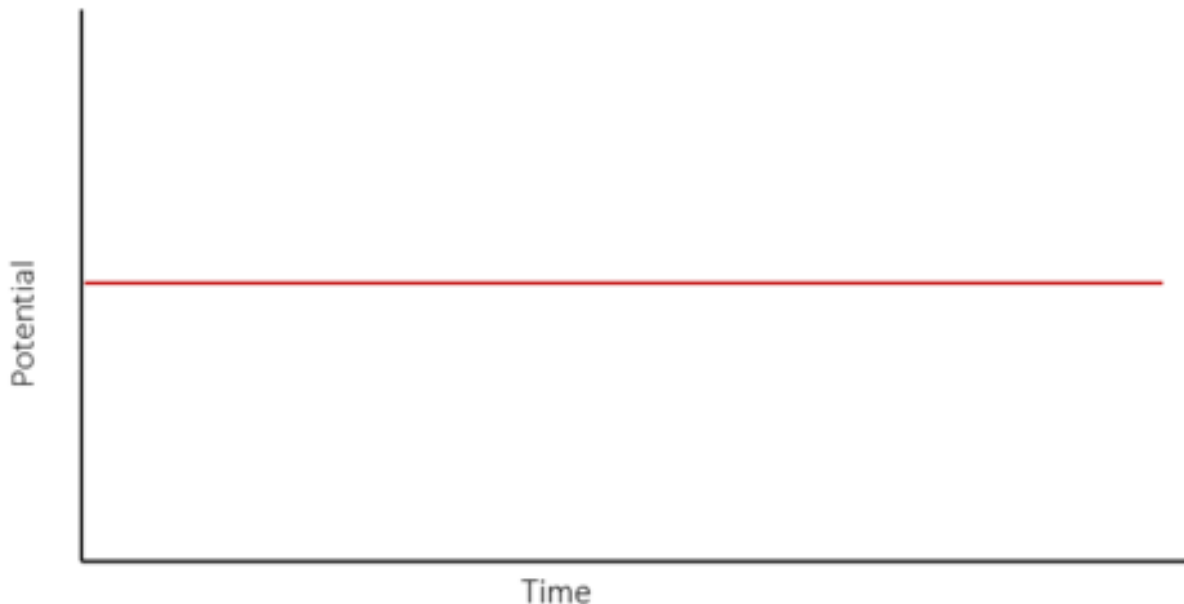
Public Member Functions

- `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.

11.20.1 Detailed Description

This experiment observes the **open circuit potential** of the working electrode for a specific period of time.



11.20.2 Constructor & Destructor Documentation

11.20.2.1 AisOpenCircuitElement()

```
AisOpenCircuitElement::AisOpenCircuitElement (
    double duration,
    double samplingInterval ) [explicit]
```

the open-circuit element constructor.

Parameters

<i>duration</i>	the maximum duration for the experiment in seconds.
<i>samplingInterval</i>	the data sampling interval value in seconds.

11.20.3 Member Function Documentation**11.20.3.1 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.

11.20.3.2 getCategory()

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

get a list of applicable categories of the element.

Returns

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

11.20.3.3 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.

11.20.3.4 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.

11.20.3.5 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

11.20.3.6 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.

11.20.3.7 getName()

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

get the name of the element.

Returns

The name of the element: "Open Circuit Potential".

11.20.3.8 getSamplingInterval()

```
double AisOpenCircuitElement::getSamplingInterval ( ) const
```

get how frequently we are sampling the data.

Returns

the data sampling interval value in seconds.

11.20.3.9 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.

11.20.3.10 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

<i>approxMaxVoltage</i>	the value for the maximum current expected in V.
-------------------------	--------------------------------------------------

11.20.3.11 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.

11.20.3.12 setMaxDuration()

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

set the value set for the duration of the experiment.

Parameters

<i>maxDuration</i>	the value to set for the duration of the experiment in seconds.
--------------------	-----------------------------------------------------------------

11.20.3.13 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

<i>maxVoltage</i>	the maximum voltage value in volts at which the experiment will stop.
-------------------	-----------------------------------------------------------------------

11.20.3.14 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

<i>mindVdt</i>	the minimum value for the voltage rate of change with respect to time (minimum dV/dt).
----------------	----------------------------------------------------------------------------------------

11.20.3.15 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

<i>minVoltage</i>	the minimum voltage value in volts at which the experiment will stop.
-------------------	-----------------------------------------------------------------------

11.20.3.16 setSamplingInterval()

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

set how frequently we are sampling the data.

Parameters

<i>samplingInterval</i>	the data sampling interval value in seconds.
-------------------------	----------------------------------------------

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

- AisOpenCircuitElement.h

11.21 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.

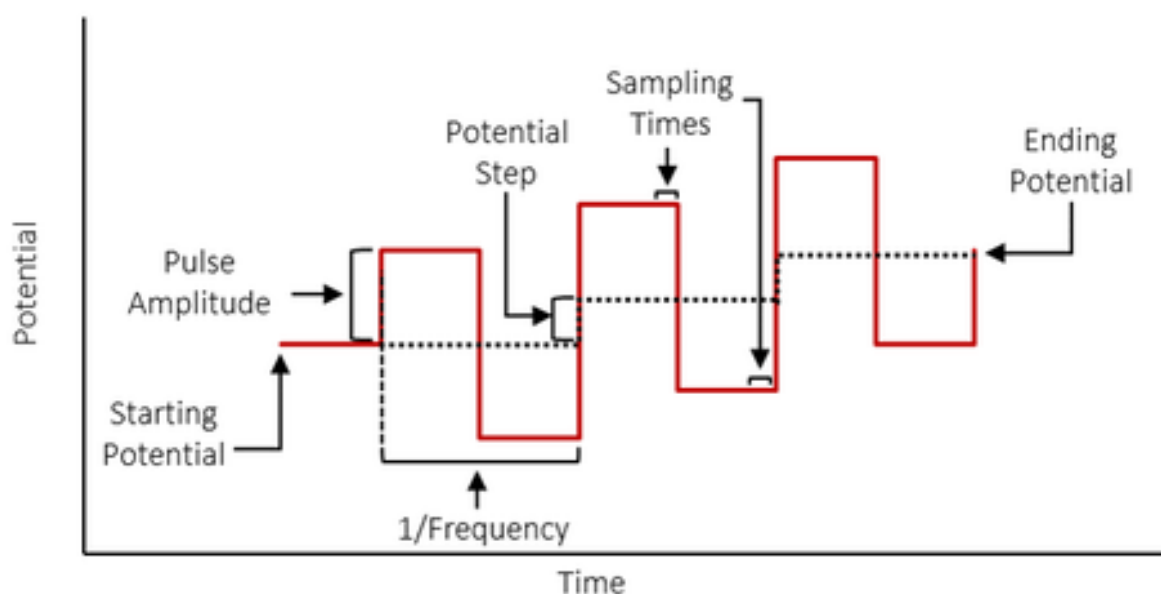
Public Member Functions

- [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.

11.21.1 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.



11.21.2 Constructor & Destructor Documentation

11.21.2.1 AisSquareWaveVoltammetryElement()

```
AisSquareWaveVoltammetryElement::AisSquareWaveVoltammetryElement (
    double startVoltage,
    double endVoltage,
    double voltageStep,
    double pulseAmp,
    double pulseFrequency ) [explicit]
```

the square wave element constructor

Parameters

<i>startVoltage</i>	the value of the starting potential in volts
<i>endVoltage</i>	the value of the ending potential in volts
<i>voltageStep</i>	the value set for the voltage step in volts.
<i>pulseAmp</i>	the value for the pulse amplitude in volts.
<i>pulseFrequency</i>	the value for the pulse frequency in Hz.

11.21.3 Member Function Documentation**11.21.3.1 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.

11.21.3.2 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").

11.21.3.3 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.

11.21.3.4 getName()

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

get the name of the element.

Returns

The name of the element: "Square Wave Potential Voltammetry".

11.21.3.5 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](#)

11.21.3.6 getPulseFreq()

```
double AisSquareWaveVoltammetryElement::getPulseFreq ( ) const
```

get the value set for the pulse frequency.

Returns

the value set for the frequency in Hz.

11.21.3.7 getStartVoltage()

```
double AisSquareWaveVoltammetryElement::getStartVoltage ( ) const
```

get the value set for the start voltage.

Returns

the value of the start voltage in volts.

11.21.3.8 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](#)

11.21.3.9 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 range has been selected.

11.21.3.10 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.

11.21.3.11 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](#)

11.21.3.12 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

<i>approxMaxCurrent</i>	the value for the maximum current expected in Amps.
-------------------------	-----------------------------------------------------

11.21.3.13 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.

11.21.3.14 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

<i>endVoltage</i>	the value to set for the ending potential in volts.
-------------------	-----------------------------------------------------

11.21.3.15 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

<i>endVoltageVsOcp</i>	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.

11.21.3.16 setPulseAmp()

```
void AisSquareWaveVoltammetryElement::setPulseAmp (
    double pulseAmp )
```

set the value for the pulse amplitude.

The voltage pulse goes up in height 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

<i>pulseAmp</i>	the value to set for the pulse amplitude in volts.
-----------------	----------------------------------------------------

11.21.3.17 setPulseFreq()

```
void AisSquareWaveVoltammetryElement::setPulseFreq (
    double pulseFreq )
```


set the value for the pulse frequency.

Parameters

<i>pulseFreq</i>	the value to set for the pulse frequency in Hz.
------------------	-------------------------------------------------

11.21.3.18 setStartVoltage()

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

set the value for the start voltage.

Parameters

<i>startVoltage</i>	the value of the start voltage in volts
---------------------	-----------------------------------------

11.21.3.19 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

<i>startVoltageVsOcp</i>	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.

11.21.3.20 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

<code>vStep</code>	the value for the voltage step in volts.
--------------------	------------------------------------------

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

- AisSquareWaveVoltammetryElement.h

Chapter 12

File Documentation

12.1 AisCompRange.h

```
1 #ifndef SQUIDSTATLIBRARY_AISCOMPRANGE_H
2 #define SQUIDSTATLIBRARY_AISCOMPRANGE_H
3
4 #include "AisSquidstatGlobal.h"
5 #include <QString>
6 #include <memory>
7
8 class AisCompRangePrivate;
9
10 class SQUIDSTATLIBRARY_EXPORT AisCompRange final {
11 public:
12     explicit AisCompRange(const QString& compRangeName, uint8_t bandwidthIndex, uint8_t stabilityFactor);
13
14     AisCompRange(const AisCompRange&);
15
16     uint8_t getBandwidthIndex() const;
17
18     void setBandwidthIndex(uint8_t index);
19
20     uint8_t getStabilityFactor() const;
21
22     void setStabilityFactor(uint8_t factor);
23
24     void setCompRangeName(const QString& compRangeName);
25
26     const QString& getCompRangeName() const;
27 private:
28     std::shared_ptr<AisCompRangePrivate> m_data;
29 };
30 #endif
```

12.2 AisDataPoints.h

```
1 #ifndef SQUIDSTATLIBRARY_AISDATAPOINTS_H
2 #define SQUIDSTATLIBRARY_AISDATAPOINTS_H
3
4 struct AisDCData {
5
6     double timestamp;
7
8     double workingElectrodeVoltage;
9
10    double counterElectrodeVoltage;
11
12    double current;
13
14    double temperature;
15 };
16
17 struct AisACData {
```

```

43     double timestamp;
44
48     double frequency;
49
53     double absoluteImpedance;
54
58     double realImpedance;
59
63     double imagImpedance;
64
68     double phaseAngle;
69
73     double totalHarmonicDistortion;
74
81     double numberOfCycles;
82
86     double workingElectrodeDCVoltage;
87
91     double DCCurrent;
92
96     double currentAmplitude;
97
101    double voltageAmplitude;
102 };
103
107 struct AisExperimentNode {
108
112     QString stepName;
113
117     int stepNumber;
118
122     int substepNumber;
123 };
124
125 #endif //SQUIDSTATLIBRARY_AISDATAPOINTS_H

```

12.3 AisDeviceTracker.h

```

1 #ifndef SQUIDSTATLIBRARY_AISDEVICETRACKER_H
2 #define SQUIDSTATLIBRARY_AISDEVICETRACKER_H
3
4 #include "AisErrorCode.h"
5 #include "AisSquidstatGlobal.h"
6 #include <QObject>
7 #include <memory>
8
9
10
11 class AisDeviceTrackerPrivate;
12 class AisInstrumentHandler;
13
18 class SQUIDSTATLIBRARY_EXPORT AisDeviceTracker final : public QObject
19 {
20     Q_OBJECT
21 public:
22     ~AisDeviceTracker() override;
23     static AisDeviceTracker *Instance();
24
37     AisErrorCode connectToDeviceOnComPort(const QString &comPort);
38
50     const AisInstrumentHandler &getInstrumentHandler(const QString &deviceName) const;
51
56     const std::list<QString> getConnectedDevices() const;
57
66     int connectAllPluggedInDevices();
67
80     AisErrorCode updateFirmwareOnComPort(QString comPort) const;
81
93     int updateFirmwareOnAllAvailableDevices();
94
95 signals:
101     void newDeviceConnected(const QString &deviceName);
102
107     void deviceDisconnected(const QString &deviceName);
108
109     void firmwareUpdateNotification(const QString& message);
110
111 private:
112     AisDeviceTracker();
113     AisDeviceTracker(const AisDeviceTracker &);
114     void operator=(const AisDeviceTracker &);
115
116     std::unique_ptr<AisDeviceTrackerPrivate> m_data;

```

```

117
118
119 };
120
121 #endif

```

12.4 AisErrorCode.h

```

1
2 #ifndef AIS_ERROR_CODE_H
3 #define AIS_ERROR_CODE_H
4
5 #include "AisSquidstatGlobal.h"
6 #include <QString.h>
7
15 class SQUIDSTATLIBRARY_EXPORT AisErrorCode {
16
17 public:
18     enum ErrorCode : uint8_t {
19         Unknown = 255,
20         Success = 0,
21         ConnectionFailed = 1,
22         FirmwareNotSupported = 2,
23         FirmwareFileNotFound = 3,
24         FirmwareUptodate = 4,
25
26         InvalidChannel = 10,
27         BusyChannel = 11,
28         DeviceNotFound = 13,
29         ManualExperimentNotRunning = 51,
30         ExperimentNotUploaded = 52,
31         ExperimentIsEmpty = 53,
32         InvalidParameters = 54,
33         ChannelNotBusy = 55,
34         DeviceCommunicationFailed = 100,
35         FailedToSetManualModeCurrentRange = 101,
36         FailedToSetManualModeConstantVoltage = 102,
37         FailedToPauseExperiment = 103,
38         FailedToResumeExperiment = 104,
39         FailedToStopExperiment = 105,
40         FailedToUploadExperiment = 106,
41         ExperimentAlreadyPaused = 107,
42         ExperimentAlreadyRun = 108,
43     };
44
45     AisErrorCode();
46     AisErrorCode(ErrorCode error);
47     AisErrorCode(ErrorCode error, QString message);
48
49     QString message() const;
50
51     int value() const;
52
53     operator ErrorCode() const;
54
55 private:
56     ErrorCode code;
57     QString errorMessage;
58 };
59
60 #endif // ! AIS_ERROR_CODE_H

```

12.5 AisExperiment.h

```

1 #ifndef SQUIDSTATLIBRARY_AISCUSTOMEXPERIMENTRUNNER_H
2 #define SQUIDSTATLIBRARY_AISCUSTOMEXPERIMENTRUNNER_H
3
4 #include "AisSquidstatGlobal.h"
5 #include "experiments/builder_elements/AisAbstractElement.h"
6 #include <QString>
7
8 class CustomExperimentRunner;
9 class AisExperimentPrivate;
10
11 class SQUIDSTATLIBRARY_EXPORT AisExperiment final {
12 public:
13     explicit AisExperiment();
14
15     explicit AisExperiment(const AisExperiment& exp);

```

```

31
32     void operator=(const AisExperiment& exp);
33
34     ~AisExperiment();
35
36     QString getExperimentName() const;
37
38     QString getDescription() const;
39
40     QStringList getCategory() const;
41
42     void setExperimentName(QString name);
43
44     void setDescription(QString description);
45
46     bool appendElement(AisAbstractElement& element, uint repeat = 1);
47
48     bool appendSubExperiment(const AisExperiment& subExp, uint repeat = 1);
49
50 private:
51     friend class AisInstrumentHandler;
52     std::shared_ptr<AisExperimentPrivate> m_data;
53 };
54
55 #endif //SQUIDSTATLIBRARY_AISCUSTOMEXPERIMENTRUNNER_H

```

12.6 AisInstrumentHandler.h

```

1 #ifndef SQUIDSTATLIBRARY_AISINSTRUMENTHANDLER_H
2 #define SQUIDSTATLIBRARY_AISINSTRUMENTHANDLER_H
3
4 #include <ctime>
5
6 #include <QObject>
7
8 #include "AisCompRange.h"
9 #include "AisDataPoints.h"
10 #include "AisErrorCode.h"
11 #include "AisSquidstatGlobal.h"
12
13 class AisInstrumentHandlerPrivate;
14 class AisExperiment;
15
16 class SQUIDSTATLIBRARY_EXPORT AisInstrumentHandler final : public QObject {
17     Q_OBJECT
18     AisInstrumentHandlerPrivate* m_data;
19 public:
20     explicit AisInstrumentHandler(AisInstrumentHandlerPrivate* privateData);
21     ~AisInstrumentHandler();
22
23     AisErrorCode uploadExperimentToChannel(uint8_t channel, std::shared_ptr<AisExperiment> experiment)
24     const;
25
26     AisErrorCode uploadExperimentToChannel(uint8_t channel, const AisExperiment& experiment) const;
27
28     AisErrorCode startUploadedExperiment(uint8_t channel) const;
29
30     AisErrorCode skipExperimentStep(uint8_t channel) const;
31
32     AisErrorCode pauseExperiment(uint8_t channel) const;
33
34     AisErrorCode resumeExperiment(uint8_t channel) const;
35
36     AisErrorCode stopExperiment(uint8_t channel) const;
37
38     double getExperimentUTCStartTime(uint8_t channel) const;
39
40     AisErrorCode setIRComp(uint8_t channel, double uncompensatedResistance, double compensationLevel)
41     const;
42
43     AisErrorCode setCompRange(uint8_t channel, const AisCompRange& compRange) const;
44
45     int8_t setLinkedChannels(std::vector<uint8_t> channels) const;
46
47     std::vector<uint8_t> getLinkedChannels(uint8_t channel) const;
48
49     bool isChannelBusy(uint8_t channel) const;
50
51     bool isChannelPaused(uint8_t channel) const;
52
53     std::vector<uint8_t> getFreeChannels() const;
54
55     int getNumberOfChannels() const;

```



```

236
244     AisErrorCode eraseRecoverData() const;
245
259     AisErrorCode startManualExperiment(uint8_t channel) const;
260
271     AisErrorCode setManualModeSamplingInterval(uint8_t channel, double value) const;
272
287     AisErrorCode setManualModeOCP(uint8_t channel) const;
288
300     AisErrorCode setManualModeConstantVoltage(uint8_t channel, double value) const;
301
316     AisErrorCode setManualModeConstantVoltage(uint8_t channel, double value, int currentRangeIndex)
const;
317
330     AisErrorCode setManualModeConstantCurrent(uint8_t channel, double value) const;
331
340     std::vector<std::pair<double, double> > getManualModeCurrentRangeList(uint8_t channel) const;
341
342 signals:
343
348     void deviceDisconnected();
349
354     void groundFloatStateChanged(bool grounded);
355
362     void experimentNewElementStarting(uint8_t channel, const AisExperimentNode& stepInfo);
363
369     void activeDCDataReady(uint8_t channel, const AisDCData& DCData);
370
378     void idleDCDataReady(uint8_t channel, const AisDCData& DCData);
379
385     void recoveryDCDataReady(uint8_t channel, const AisDCData& DCData);
386
392     void activeACDataReady(uint8_t channel, const AisACData& ACData);
393
399     void recoveryACDataReady(uint8_t channel, const AisACData& ACData);
400
405     void experimentStopped(uint8_t channel);
406
411     void experimentPaused(uint8_t channel);
412
417     void experimentResumed(uint8_t channel);
418
423     void recoverDataErased(bool successful);
424
425 private slots:
426     void onActiveExperimentNodeBeginning(uint8_t channel, const AisExperimentNode&);
427     void onRecoveryExperimentNodeBeginning(uint8_t channel, const AisExperimentNode&);
428     void onDeviceDisconnected();
429
430 private:
431     void connectWithOperatorSignals();
432 };
433
434 #endif //SQUIDSTATLIBRARY_AISINSTRUMENTHANDLER_H

```

12.7 AisSquidstatGlobal.h

```

1 #pragma once
2
3 #include <QtGlobal>
4
5
6 #ifndef BUILD_STATIC
7 #if defined(SQUIDSTATLIBRARY_LIB)
8 #define SQUIDSTATLIBRARY_EXPORT Q_DECL_EXPORT
9 #else
10 #define SQUIDSTATLIBRARY_EXPORT Q_DECL_IMPORT
11 #endif
12 #else
13 #define SQUIDSTATLIBRARY_EXPORT
14 #endif

```

12.8 AisAbstractElement.h

```

1 #ifndef SQUIDSTATLIBRARY_AISABSTRACTELEMENT_H
2 #define SQUIDSTATLIBRARY_AISABSTRACTELEMENT_H
3
4 #include "AisSquidstatGlobal.h"
5 #include <QString>

```

```

6 #include <memory>
7
8 class AbstractBuilderElement;
9
15 class SQUIDSTATLIBRARY_EXPORT AisAbstractElement {
16 public:
17     virtual ~AisAbstractElement();
18
24     virtual QString getName() const = 0;
25
31     virtual QStringList getCategory() const = 0;
32
33 protected:
35     std::shared_ptr<AbstractBuilderElement> m_data;
37     friend class AisExperiment;
38 };
39
40 #endif //SQUIDSTATLIBRARY_AISABSTRACTELEMENT_H

```

12.9 AisConstantCurrentElement.h

```

1 #pragma once
2
3 #include "AisAbstractElement.h"
4 #include "AisSquidstatGlobal.h"
5 #include <QString>
6
7 class ConstantCurrentAdvElement;
8
15 class SQUIDSTATLIBRARY_EXPORT AisConstantCurrentElement final : public AisAbstractElement {
16 public:
23     explicit AisConstantCurrentElement (
24         double current,
25         double samplingInterval,
26         double duration);
30     explicit AisConstantCurrentElement (const AisConstantCurrentElement&);
34     AisConstantCurrentElement& operator=(const AisConstantCurrentElement&);
35
36     ~AisConstantCurrentElement() override;
37
42     QString getName() const override;
43
48     QStringList getCategory() const override;
49
54     double getCurrent() const;
55
60     void setCurrent(double current);
61
66     double getSamplingInterval() const;
67
72     void setSamplingInterval(double samplingInterval);
73
82     double getMinSamplingVoltageDifference() const;
83
96     void setMinSamplingVoltageDifference(double minVoltageDifference);
97
104     double getMaxVoltage() const;
105
114     void setMaxVoltage(double maxVoltage);
115
121     double getMinVoltage() const;
122
131     void setMinVoltage(double minVoltage);
132
138     double getMaxDuration() const;
139
146     void setMaxDuration(double maxDuration);
147
153     double getMaxCapacity() const;
154
163     void setMaxCapacity(double maxCapacity);
164
169     bool isAutoRange() const;
170
176     void setAutoRange();
177
183     double getApproxMaxCurrent() const;
184
192     void setApproxMaxCurrent(double approxMaxCurrent);
193
198     bool isAutoVoltageRange() const;
199
205     void setAutoVoltageRange();

```

```

206
207
213     double getApproxMaxVoltage() const;
214
222     void setApproxMaxVoltage(double approxMaxVoltage);
223
224 private:
225     std::shared_ptr<ConstantCurrentAdvElement> m_dataDerived;
226 };

```

12.10 AisConstantPotElement.h

```

1  #pragma once
2
3  #include "AisAbstractElement.h"
4  #include "AisSquidstatGlobal.h"
5  #include <QString>
6
7  class ConstantPotAdvElement;
8
15 class SQUIDSTATLIBRARY_EXPORT AisConstantPotElement final : public AisAbstractElement {
16 public:
23     explicit AisConstantPotElement(
24         double voltage,
25         double samplingInterval,
26         double duration);
30     explicit AisConstantPotElement(const AisConstantPotElement&);
31
35     AisConstantPotElement& operator=(const AisConstantPotElement&);
36
37     ~AisConstantPotElement() override;
38
43     QString getName() const override;
44
49     QStringList getCategory() const override;
50
55     double getPotential() const;
56
61     void setPotential(double potential);
62
68     bool isVoltageVsOCP() const;
69
76     void setVoltageVsOCP(bool vsOCP);
77
82     double getSamplingInterval() const;
83
88     void setSamplingInterval(double samplingInterval);
89
95     double getMaxDuration() const;
96
105    void setMaxDuration(double maxDuration);
106
113    double getMaxCurrent() const;
114
123    void setMaxCurrent(double maxCurrent);
124
131    double getMinCurrent() const;
132
141    void setMinCurrent(double minCurrent);
142
148    double getMaxCapacity() const;
149
158    void setMaxCapacity(double maxCapacity);
159
165    double getMindIdt() const;
166
175    void setMindIdt(double mindIdt);
176
181    bool isAutoRange() const;
182
188    void setAutoRange();
189
195    double getApproxMaxCurrent() const;
196
204    void setApproxMaxCurrent(double approxMaxCurrent);
205
206 private:
207     std::shared_ptr<ConstantPotAdvElement> m_dataDerived;
208 };

```

12.11 AisConstantPowerElement.h

```

1  #pragma once
2
3  #include "AisAbstractElement.h"
4  #include "AisSquidstatGlobal.h"
5  #include <QString>
6
7  class ConstantPowerElement;
8
15 class SQUIDSTATLIBRARY_EXPORT AisConstantPowerElement final : public AisAbstractElement {
16 public:
24     explicit AisConstantPowerElement (
25         bool isCharge,
26         double power,
27         double duration,
28         double smaplingInterval);
29
33     explicit AisConstantPowerElement (const AisConstantPowerElement&);
34
38     AisConstantPowerElement& operator=(const AisConstantPowerElement&);
39
40     ~AisConstantPowerElement() override;
41
46     QString getName() const override;
47
52     QStringList getCategory() const override;
53
58     bool isCharge() const;
59
64     void setCharge(bool isCharge);
65
70     double getPower() const;
71
76     void setPower(double power);
77
82     double getSamplingInterval() const;
83
88     void setSamplingInterval(double samplingInterval);
89
96     double getMaxVoltage() const;
97
106    void setMaxVoltage(double maxVoltage);
107
114    double getMinVoltage() const;
115
122    void setMinVoltage(double minVoltage);
123
129    double getMaxDuration() const;
130
137    void setMaxDuration(double maxDuration);
138
144    double getMaxCapacity() const;
145
154    void setMaxCapacity(double maxCapacity);
155
156 private:
157     std::shared_ptr<ConstantPowerElement> m_dataDerived;
158 };

```

12.12 AisConstantResistanceElement.h

```

1  #pragma once
2
3  #include "AisAbstractElement.h"
4  #include "AisSquidstatGlobal.h"
5  #include <QString>
6
7  class ConstantResistanceElement;
8
15 class SQUIDSTATLIBRARY_EXPORT AisConstantResistanceElement final : public AisAbstractElement {
16 public:
23     explicit AisConstantResistanceElement (
24         double resistance,
25         double duration,
26         double samplingInterval);
30     explicit AisConstantResistanceElement (const AisConstantResistanceElement&);
34     AisConstantResistanceElement& operator=(const AisConstantResistanceElement&);
35
36     ~AisConstantResistanceElement() override;
37
42     QString getName() const override;
43

```

```

48     QStringList getCategory() const override;
49
54     double getResistance() const;
55
60     void setResistance(double resistance);
61
66     double getSamplingInterval() const;
67
72     void setSamplingInterval(double samplingInterval);
73
79     double getMinVoltage() const;
80
89     void setMinVoltage(double minVoltage);
90
96     double getMaxDuration() const;
97
104    void setMaxDuration(double maxDuration);
105
111    double getMaxCapacity() const;
112
121    void setMaxCapacity(double maxCapacity);
122
123 private:
124     std::shared_ptr<ConstantResistanceElement> m_dataDerived;
125 };

```

12.13 AisCyclicVoltammetryElement.h

```

1  #pragma once
2
3  #include "AisAbstractElement.h"
4  #include "AisSquidstatGlobal.h"
5  #include <QString>
6
7  class CyclicVoltammetryElement;
8
22 class SQUIDSTATLIBRARY_EXPORT AisCyclicVoltammetryElement final : public AisAbstractElement {
23 public:
33     explicit AisCyclicVoltammetryElement(
34         double startVoltage,
35         double firstVoltageLimit,
36         double secondVoltageLimit,
37         double endVoltage,
38         double dEdt,
39         double samplingInterval);
40
44     explicit AisCyclicVoltammetryElement(const AisCyclicVoltammetryElement&);
45
49     AisCyclicVoltammetryElement& operator=(const AisCyclicVoltammetryElement&);
50
51     ~AisCyclicVoltammetryElement() override;
52
57     QString getName() const override;
58
63     QStringList getCategory() const override;
64
69     double getStartVoltage() const;
70
75     void setStartVoltage(double startVoltage);
76
81     bool isStartVoltageVsOCP() const;
82
89     void setStartVoltageVsOCP(bool startVoltageVsOCP);
90
98     double getFirstVoltageLimit() const;
99
107    void setFirstVoltageLimit(double v1);
108
114    bool isFirstVoltageLimitVsOCP() const;
115
122    void setFirstVoltageLimitVsOCP(bool firstVoltageLimitVsOCP);
123
131    double getSecondVoltageLimit() const;
132
140    void setSecondVoltageLimit(double v2);
141
147    bool isSecondVoltageLimitVsOCP() const;
148
155    void setSecondVoltageLimitVsOCP(bool secondVoltageLimitVsOCP);
156
161    double getNumberOfCycles();
162
167    void setNumberOfCycles(int cycles);

```

```

168
176     double getEndVoltage() const;
177
185     void setEndVoltage(double endVoltage);
186
192     bool isEndVoltageVsOCP() const;
193
200     void setEndVoltageVsOCP(bool endVoltageVsOCP);
201
206     double getdEdt() const;
207
212     void setdEdt(double dEdt);
213
218     double getSamplingInterval() const;
219
224     void setSamplingInterval(double sampInterval);
225
230     bool isAutoRange() const;
231
237     void setAutoRange();
238
244     double getApproxMaxCurrent() const;
245
253     void setApproxMaxCurrent(double approxMaxCurrent);
254
255 private:
256     std::shared_ptr<CyclicVoltammetryElement> m_dataDerived;
257 };

```

12.14 AisDCCurrentSweepElement.h

```

1  #pragma once
2
3  #include "AisAbstractElement.h"
4  #include "AisSquidstatGlobal.h"
5  #include <QString>
6
7  class DCCurrentSweepElement;
8
17 class SQUIDSTATLIBRARY_EXPORT AisDCCurrentSweepElement : public AisAbstractElement {
18 public:
26     explicit AisDCCurrentSweepElement(
27         double startCurrent,
28         double endCurrent,
29         double scanRate,
30         double samplingInterval
31     );
32
36     explicit AisDCCurrentSweepElement(const AisDCCurrentSweepElement&);
40     AisDCCurrentSweepElement& operator=(const AisDCCurrentSweepElement&);
41
42     ~AisDCCurrentSweepElement() override;
43
48     QString getName() const override;
49
59     QStringList getCategory() const override;
60
65     double getStartingCurrent() const;
66
71     void setStartingCurrent(double startingCurrent);
72
77     double getEndingCurrent() const;
78
83     void setEndingCurrent(double endingCurrent);
84
90     double getScanRate() const;
91
98     void setScanRate(double scanRate);
99
104     double getSamplingInterval() const;
105
110     void setSamplingInterval(double samplingInterval);
111
118     double getMaxVoltage() const;
119
128     void setMaxVoltage(double maxVoltage);
129
135     double getMinVoltage() const;
136
145     void setMinVoltage(double minVoltage);
146
147 private:
148     std::shared_ptr<DCCurrentSweepElement> m_dataDerived;
149 };

```

12.15 AisDCPotentialSweepElement.h

```

1 #pragma once
2
3 #include "AisAbstractElement.h"
4 #include "AisSquidstatGlobal.h"
5 #include <QString>
6
7 class DCPotentialSweepElement;
8
17 class SQUIDSTATLIBRARY_EXPORT AisDCPotentialSweepElement final : public AisAbstractElement {
18 public:
26     explicit AisDCPotentialSweepElement(
27         double startPotential,
28         double endPotential,
29         double scanRate,
30         double samplingInterval);
34     explicit AisDCPotentialSweepElement(const AisDCPotentialSweepElement&);
38     AisDCPotentialSweepElement& operator=(const AisDCPotentialSweepElement&);
39
40     ~AisDCPotentialSweepElement() override;
41
46     QString getName() const override;
56     QStringList getCategory() const override;
57
62     double getStartingPot() const;
63
68     void setStartingPot(double startingPotential);
69
75     bool isStartVoltageVsOCP() const;
76
84     void setStartVoltageVsOCP(bool startVoltageVsOCP);
85
92     double getEndingPot() const;
93
100    void setEndingPot(double endingPotential);
101
107    bool isEndVoltageVsOCP() const;
108
116    void setEndVoltageVsOCP(bool endVoltageVsOCP);
117
123    double getScanRate() const;
124
131    void setScanRate(double scanRate);
132
137    double getSamplingInterval() const;
138
143    void setSamplingInterval(double samplingInterval);
144
149    bool isAutoRange() const;
150
156    void setAutoRange();
157
163    double getApproxMaxCurrent() const;
164
172    void setApproxMaxCurrent(double approxMaxCurrent);
173
174 private:
175     std::shared_ptr<DCPotentialSweepElement> m_dataDerived;
176 };

```

12.16 AisDiffPulseVoltammetryElement.h

```

1 #pragma once
2
3 #include "AisAbstractElement.h"
4 #include "AisSquidstatGlobal.h"
5 #include <QString>
6
7 class DiffPulseVoltammetryElement;
8
21 class SQUIDSTATLIBRARY_EXPORT AisDiffPulseVoltammetryElement final : public AisAbstractElement {
22 public:
32     explicit AisDiffPulseVoltammetryElement(
33         double startVoltage,
34         double endVoltage,
35         double voltageStep,
36         double pulseHeight,
37         double pulseWidth,
38         double pulsePeriod);
42     explicit AisDiffPulseVoltammetryElement(const AisDiffPulseVoltammetryElement&);
46     AisDiffPulseVoltammetryElement& operator=(const AisDiffPulseVoltammetryElement&);
47

```

```

48     ~AisDiffPulseVoltammetryElement() override;
49
50     QString getName() const override;
51
52     QStringList getCategory() const override;
53
54     double getStartVoltage() const;
55
56     void setStartVoltage(double startVoltage);
57
58     bool isStartVoltageVsOCP() const;
59
60     void setStartVoltageVsOCP(bool startVoltageVsOCP);
61
62     double getEndVoltage() const;
63
64     void setEndVoltage(double endVoltage);
65
66     bool isEndVoltageVsOCP() const;
67
68     void setEndVoltageVsOCP(bool endVoltageVsOCP);
69
70     double getVStep() const;
71
72     void setVStep(double vStep);
73
74     double getPulseHeight() const;
75
76     void setPulseHeight(double pulseHeight);
77
78     double getPulseWidth() const;
79
80     void setPulseWidth(double pulseWidth);
81
82     double getPulsePeriod() const;
83
84     void setPulsePeriod(double pulsePeriod);
85
86     bool isAutoRange() const;
87
88     void setAutoRange();
89
90     double getApproxMaxCurrent() const;
91
92     void setApproxMaxCurrent(double approxMaxCurrent);
93
94 private:
95     std::shared_ptr<DiffPulseVoltammetryElement> m_dataDerived;
96 };

```

12.17 AisEISGalvanostaticElement.h

```

1  #pragma once
2
3  #include "AisAbstractElement.h"
4  #include "AisSquidstatGlobal.h"
5  #include <QString>
6
7  class EISGalvanostaticElement;
8
9  class SQUIDSTATLIBRARY_EXPORT AisEISGalvanostaticElement final : public AisAbstractElement {
10 public:
11     explicit AisEISGalvanostaticElement(
12         double startFrequency,
13         double endFrequency,
14         double stepsPerDecade,
15         double currentBias,
16         double currentAmplitude);
17     explicit AisEISGalvanostaticElement(const AisEISGalvanostaticElement&);
18     AisEISGalvanostaticElement& operator=(const AisEISGalvanostaticElement&);
19
20     ~AisEISGalvanostaticElement() override;
21
22     QString getName() const override;
23
24     QStringList getCategory() const override;
25
26     double getStartFreq() const;
27
28     void setStartFreq(double startFreq);
29
30     double getEndFreq() const;
31 };

```



```

86     void setEndFreq(double endFreq);
87
92     double getStepsPerDecade() const;
93
98     void setStepsPerDecade(double stepsPerDecade);
99
104    double getBiasCurrent() const;
105
110    void setBiasCurrent(double biasCurrent);
111
116    double getAmplitude() const;
117
122    void setAmplitude(double amplitude);
123
124 private:
125     std::shared_ptr<EISGalvanostaticElement> m_dataDerived;
126 };

```

12.18 AisEISPotentiostaticElement.h

```

1 #pragma once
2
3 #include "AisAbstractElement.h"
4 #include "AisSquidstatGlobal.h"
5 #include <QString>
6
7 class EISPotentiostaticElement;
8
20 class SQUIDSTATLIBRARY_EXPORT AisEISPotentiostaticElement final : public AisAbstractElement {
21 public:
30     explicit AisEISPotentiostaticElement(
31         double startFrequency,
32         double endFrequency,
33         double stepsPerDecade,
34         double voltageBias,
35         double voltageAmplitude);
39     explicit AisEISPotentiostaticElement(const AisEISPotentiostaticElement&);
43     AisEISPotentiostaticElement& operator=(const AisEISPotentiostaticElement&);
44
45     ~AisEISPotentiostaticElement() override;
46
51     QString getName() const override;
52
57     QStringList getCategory() const override;
58
63     double getStartFreq() const;
64
69     void setStartFreq(double startFreq);
70
75     double getEndFreq() const;
76
81     void setEndFreq(double endFreq);
82
87     double getStepsPerDecade() const;
88
93     void setStepsPerDecade(double stepsPerDecade);
94
99     double getBiasVoltage() const;
100
105     void setBiasVoltage(double biasVoltage);
106
113     bool isBiasVoltageVsOCP() const;
114
119     void setBiasVoltageVsOCP(bool biasVsOCP);
120
125     double getAmplitude() const;
126
131     void setAmplitude(double amplitude);
132
133 private:
134     std::shared_ptr<EISPotentiostaticElement> m_dataDerived;
135 };

```

12.19 AisNormalPulseVoltammetryElement.h

```

1 #pragma once
2
3 #include "AisAbstractElement.h"
4 #include "AisSquidstatGlobal.h"

```

```

5 #include <QString>
6
7 class NormalPulseVoltammetryElement;
8
21 class SQUIDSTATLIBRARY_EXPORT AisNormalPulseVoltammetryElement final : public AisAbstractElement {
22 public:
31     explicit AisNormalPulseVoltammetryElement (
32         double startVoltage,
33         double endVoltage,
34         double voltageStep,
35         double pulseWidth,
36         double pulsePeriod);
40     explicit AisNormalPulseVoltammetryElement(const AisNormalPulseVoltammetryElement&);
44     AisNormalPulseVoltammetryElement& operator=(const AisNormalPulseVoltammetryElement&);
45
46     ~AisNormalPulseVoltammetryElement() override;
47
52     QString getName() const override;
53
58     QStringList getCategory() const override;
59
64     double getStartVoltage() const;
65
70     void setStartVoltage(double startVoltage);
71
78     bool isStartVoltageVsOCP() const;
79
87     void setStartVoltageVsOCP(bool startVoltageVsOCP);
88
95     double getEndVoltage() const;
96
103     void setEndVoltage(double endVoltage);
104
110     bool isEndVoltageVsOCP() const;
111
119     void setEndVoltageVsOCP(bool endVoltageVsOcp);
120
126     double getVStep() const;
127
134     void setVStep(double vStep);
135
142     double getPulseWidth() const;
143
150     void setPulseWidth(double pulseWidth);
151
157     double getPulsePeriod() const;
158
165     void setPulsePeriod(double pulsePeriod);
166
171     bool isAutoRange() const;
172
178     void setAutoRange();
179
185     double getApproxMaxCurrent() const;
186
194     void setApproxMaxCurrent(double approxMaxCurrent);
195
196 private:
197     std::shared_ptr<NormalPulseVoltammetryElement> m_dataDerived;
198 };

```

12.20 AisOpenCircuitElement.h

```

1 #pragma once
2
3 #include "AisAbstractElement.h"
4 #include "AisSquidstatGlobal.h"
5 #include <QString>
6
7 class OpenCircuitElement;
8
15 class SQUIDSTATLIBRARY_EXPORT AisOpenCircuitElement final : public AisAbstractElement {
16 public:
22     explicit AisOpenCircuitElement (
23         double duration,
24         double samplingInterval);
28     explicit AisOpenCircuitElement(const AisOpenCircuitElement&);
32     AisOpenCircuitElement& operator=(const AisOpenCircuitElement&);
33
34     ~AisOpenCircuitElement() override;
35
40     QString getName() const override;
41

```

```

46     QStringList getCategory() const override;
47
52     double getSamplingInterval() const;
53
58     void setSamplingInterval(double samplingInterval);
59
64     double getMaxDuration() const;
65
70     void setMaxDuration(double maxDuration);
71
78     double getMaxVoltage() const;
79
88     void setMaxVoltage(double maxVoltage);
89
95     double getMinVoltage() const;
96
105    void setMinVoltage(double minVoltage);
106
113    double getMindVdt() const;
114
123    void setMindVdt(double mindVdt);
124
129    bool isAutoVoltageRange() const;
130
136    void setAutoVoltageRange();
137
143    double getApproxMaxVoltage() const;
144
152    void setApproxMaxVoltage(double approxMaxVoltage);
153
154 private:
155     std::shared_ptr<OpenCircuitElement> m_dataDerived;
156 };

```

12.21 AisSquareWaveVoltammetryElement.h

```

1 #pragma once
2
3 #include "AisAbstractElement.h"
4 #include "AisSquidstatGlobal.h"
5 #include <QString>
6
7 class SquareWaveVoltammetryElement;
8
23 class SQUIDSTATLIBRARY_EXPORT AisSquareWaveVoltammetryElement final : public AisAbstractElement {
24 public:
33     explicit AisSquareWaveVoltammetryElement(
34         double startVoltage,
35         double endVoltage,
36         double voltageStep,
37         double pulseAmp,
38         double pulseFrequency);
39
43     explicit AisSquareWaveVoltammetryElement(const AisSquareWaveVoltammetryElement&);
47     AisSquareWaveVoltammetryElement& operator=(const AisSquareWaveVoltammetryElement&);
48
49     ~AisSquareWaveVoltammetryElement() override;
50
55     QString getName() const override;
56
61     QStringList getCategory() const override;
62
67     double getStartVoltage() const;
68
73     void setStartVoltage(double startVoltage);
74
81     bool isStartVoltageVsOCP() const;
82
90     void setStartVoltageVsOCP(bool startVoltageVsOcp);
91
98     double getEndVoltage() const;
99
106    void setEndVoltage(double endVoltage);
107
113    bool isEndVoltageVsOCP() const;
114
122    void setEndVoltageVsOCP(bool endVoltageVsOcp);
123
129    double getVStep() const;
130
137    void setVStep(double vStep);
138
144    double getPulseAmp() const;

```

```
145
153     void setPulseAmp(double pulseAmp);
154
159     double getPulseFreq() const;
160
165     void setPulseFreq(double pulseFreq);
166
171     bool isAutoRange() const;
172
178     void setAutoRange();
179
185     double getApproxMaxCurrent() const;
194     void setApproxMaxCurrent(double approxMaxCurrent);
195
196 private:
197     std::shared_ptr<SquareWaveVoltammetryElement> m_dataDerived;
198 };
```

Index

- activeACDataReady
 - AisInstrumentHandler, [125](#)
- activeDCDataReady
 - AisInstrumentHandler, [125](#)
- AisAbstractElement.h, [171](#)
- AisACData, [31](#)
 - numberOfCycles, [32](#)
- AisCompRange, [32](#)
 - AisCompRange, [33](#)
 - getBandwidthIndex, [33](#)
 - getCompRangeName, [33](#)
 - getStabilityFactor, [34](#)
 - setBandwidthIndex, [34](#)
 - setCompRangeName, [34](#)
 - setStabilityFactor, [35](#)
- AisCompRange.h, [167](#)
- AisConstantCurrentElement, [35](#)
 - AisConstantCurrentElement, [37](#)
 - getApproxMaxCurrent, [37](#)
 - getApproxMaxVoltage, [38](#)
 - getCategory, [38](#)
 - getCurrent, [38](#)
 - getMaxCapacity, [39](#)
 - getMaxDuration, [39](#)
 - getMaxVoltage, [39](#)
 - getMinSamplingVoltageDifference, [39](#)
 - getMinVoltage, [40](#)
 - getName, [40](#)
 - getSamplingInterval, [40](#)
 - isAutoRange, [41](#)
 - isAutoVoltageRange, [41](#)
 - setApproxMaxCurrent, [41](#)
 - setApproxMaxVoltage, [42](#)
 - setAutoRange, [42](#)
 - setAutoVoltageRange, [42](#)
 - setCurrent, [42](#)
 - setMaxCapacity, [42](#)
 - setMaxDuration, [43](#)
 - setMaxVoltage, [43](#)
 - setMinSamplingVoltageDifference, [43](#)
 - setMinVoltage, [44](#)
 - setSamplingInterval, [44](#)
- AisConstantCurrentElement.h, [172](#)
- AisConstantPotElement, [45](#)
 - AisConstantPotElement, [46](#)
 - getApproxMaxCurrent, [47](#)
 - getCategory, [47](#)
 - getMaxCapacity, [47](#)
 - getMaxCurrent, [48](#)
 - getMaxDuration, [48](#)
 - getMinCurrent, [48](#)
 - getMindIdt, [49](#)
 - getName, [49](#)
 - getPotential, [49](#)
 - getSamplingInterval, [50](#)
 - isAutoRange, [50](#)
 - isVoltageVsOCP, [50](#)
 - setApproxMaxCurrent, [50](#)
 - setAutoRange, [51](#)
 - setMaxCapacity, [51](#)
 - setMaxCurrent, [51](#)
 - setMaxDuration, [51](#)
 - setMinCurrent, [52](#)
 - setMindIdt, [52](#)
 - setPotential, [52](#)
 - setSamplingInterval, [53](#)
 - setVoltageVsOCP, [53](#)
- AisConstantPotElement.h, [173](#)
- AisConstantPowerElement, [53](#)
 - AisConstantPowerElement, [55](#)
 - getCategory, [56](#)
 - getMaxCapacity, [56](#)
 - getMaxDuration, [56](#)
 - getMaxVoltage, [56](#)
 - getMinVoltage, [57](#)
 - getName, [57](#)
 - getPower, [57](#)
 - getSamplingInterval, [58](#)
 - isCharge, [58](#)
 - setCharge, [58](#)
 - setMaxCapacity, [59](#)
 - setMaxDuration, [59](#)
 - setMaxVoltage, [59](#)
 - setMinVoltage, [60](#)
 - setPower, [60](#)
 - setSamplingInterval, [60](#)
- AisConstantPowerElement.h, [174](#)
- AisConstantResistanceElement, [61](#)
 - AisConstantResistanceElement, [62](#)
 - getCategory, [62](#)
 - getMaxCapacity, [63](#)
 - getMaxDuration, [63](#)
 - getMinVoltage, [63](#)
 - getName, [63](#)
 - getResistance, [64](#)
 - getSamplingInterval, [64](#)
 - setMaxCapacity, [64](#)
 - setMaxDuration, [65](#)

- setMinVoltage, 65
 - setResistance, 65
 - setSamplingInterval, 66
- AisConstantResistanceElement.h, 174
- AisCyclicVoltammetryElement, 66
 - AisCyclicVoltammetryElement, 68
 - getApproxMaxCurrent, 68
 - getCategory, 68
 - getdEdt, 69
 - getEndVoltage, 69
 - getFirstVoltageLimit, 69
 - getName, 69
 - getNumberOfCycles, 70
 - getSamplingInterval, 70
 - getSecondVoltageLimit, 70
 - getStartVoltage, 70
 - isAutoRange, 71
 - isEndVoltageVsOCP, 71
 - isFirstVoltageLimitVsOCP, 71
 - isSecondVoltageLimitVsOCP, 71
 - isStartVoltageVsOCP, 72
 - setApproxMaxCurrent, 72
 - setAutoRange, 72
 - setdEdt, 72
 - setEndVoltage, 73
 - setEndVoltageVsOCP, 73
 - setFirstVoltageLimit, 73
 - setFirstVoltageLimitVsOCP, 74
 - setNumberOfCycles, 74
 - setSamplingInterval, 74
 - setSecondVoltageLimit, 75
 - setSecondVoltageLimitVsOCP, 75
 - setStartVoltage, 75
 - setStartVoltageVsOCP, 76
- AisCyclicVoltammetryElement.h, 175
- AisDataPoints.h, 167
- AisDCCurrentSweepElement, 76
 - AisDCCurrentSweepElement, 78
 - getCategory, 79
 - getEndingCurrent, 79
 - getMaxVoltage, 79
 - getMinVoltage, 79
 - getName, 80
 - getSamplingInterval, 80
 - getScanRate, 80
 - getStartingCurrent, 80
 - setEndingCurrent, 81
 - setMaxVoltage, 81
 - setMinVoltage, 81
 - setSamplingInterval, 82
 - setScanRate, 82
 - setStartingCurrent, 82
- AisDCCurrentSweepElement.h, 176
- AisDCData, 83
- AisDCPotentialSweepElement, 83
 - AisDCPotentialSweepElement, 85
 - getApproxMaxCurrent, 86
 - getCategory, 86
 - getEndingPot, 86
 - getName, 86
 - getSamplingInterval, 87
 - getScanRate, 87
 - getStartingPot, 87
 - isAutoRange, 87
 - isEndVoltageVsOCP, 88
 - isStartVoltageVsOCP, 88
 - setApproxMaxCurrent, 88
 - setAutoRange, 89
 - setEndingPot, 89
 - setEndVoltageVsOCP, 89
 - setSamplingInterval, 89
 - setScanRate, 90
 - setStartingPot, 90
 - setStartVoltageVsOCP, 90
- AisDCPotentialSweepElement.h, 177
- AisDeviceTracker, 91
 - connectAllPluggedInDevices, 92
 - connectToDeviceOnComPort, 92
 - deviceDisconnected, 93
 - getConnectedDevices, 93
 - getInstrumentHandler, 93
 - newDeviceConnected, 94
 - updateFirmwareOnAllAvailableDevices, 94
 - updateFirmwareOnComPort, 95
- AisDeviceTracker.h, 168
- AisDiffPulseVoltammetryElement, 95
 - AisDiffPulseVoltammetryElement, 97
 - getApproxMaxCurrent, 98
 - getCategory, 98
 - getEndVoltage, 98
 - getName, 98
 - getPulseHeight, 99
 - getPulsePeriod, 99
 - getPulseWidth, 99
 - getStartVoltage, 100
 - getVStep, 100
 - isAutoRange, 100
 - isEndVoltageVsOCP, 101
 - isStartVoltageVsOCP, 101
 - setApproxMaxCurrent, 101
 - setAutoRange, 102
 - setEndVoltage, 102
 - setEndVoltageVsOCP, 102
 - setPulseHeight, 102
 - setPulsePeriod, 103
 - setPulseWidth, 103
 - setStartVoltage, 103
 - setStartVoltageVsOCP, 104
 - setVStep, 104
- AisDiffPulseVoltammetryElement.h, 177
- AisEISGalvanostaticElement, 105
 - AisEISGalvanostaticElement, 106
 - getAmplitude, 107
 - getBiasCurrent, 107
 - getCategory, 107
 - getEndFreq, 107

- getName, 108
- getStartFreq, 108
- getStepsPerDecade, 108
- setAmplitude, 108
- setBiasCurrent, 109
- setEndFreq, 109
- setStartFreq, 109
- setStepsPerDecade, 109
- AisEISGalvanostaticElement.h, 178
- AisEISPotentiostaticElement, 110
 - AisEISPotentiostaticElement, 111
 - getAmplitude, 112
 - getBiasVoltage, 112
 - getCategory, 112
 - getEndFreq, 113
 - getName, 113
 - getStartFreq, 113
 - getStepsPerDecade, 113
 - isBiasVoltageVsOCP, 114
 - setAmplitude, 114
 - setBiasVoltage, 114
 - setBiasVoltageVsOCP, 115
 - setEndFreq, 115
 - setStartFreq, 115
 - setStepsPerDecade, 115
- AisEISPotentiostaticElement.h, 179
- AisErrorCode, 116
 - BusyChannel, 117
 - ChannelNotBusy, 117
 - ConnectionFailed, 117
 - DeviceCommunicationFailed, 117
 - DeviceNotFound, 117
 - ErrorCode, 117
 - ExperimentAlreadyPaused, 118
 - ExperimentAlreadyRun, 118
 - ExperimentIsEmpty, 117
 - ExperimentNotUploaded, 117
 - FailedToPauseExperiment, 117
 - FailedToResumeExperiment, 118
 - FailedToSetManualModeConstantVoltage, 117
 - FailedToSetManualModeCurrentRange, 117
 - FailedToStopExperiment, 118
 - FailedToUploadExperiment, 118
 - FirmwareNotSupported, 117
 - InvalidChannel, 117
 - InvalidParameters, 117
 - ManualExperimentNotRunning, 117
 - message, 118
 - Success, 117
 - Unknown, 117
 - value, 118
- AisErrorCode.h, 169
- AisExperiment, 119
 - AisExperiment, 119
 - appendElement, 120
 - appendSubExperiment, 120
 - getCategory, 121
 - getDescription, 121
 - getExperimentName, 121
 - operator=, 121
 - setDescription, 122
 - setExperimentName, 122
- AisExperiment.h, 169
- AisExperimentNode, 122
- AisInstrumentHandler, 123
 - activeACDataReady, 125
 - activeDCDataReady, 125
 - deviceDisconnected, 125
 - eraseRecoverData, 125
 - experimentNewElementStarting, 126
 - experimentPaused, 126
 - experimentResumed, 126
 - experimentStopped, 127
 - getExperimentUTCStartTime, 127
 - getFreeChannels, 127
 - getLinkedChannels, 128
 - getManualModeCurrentRangeList, 128
 - getNumberOfChannels, 128
 - groundFloatStateChanged, 129
 - idleDCDataReady, 129
 - isChannelBusy, 129
 - isChannelPaused, 130
 - pauseExperiment, 130
 - recoverDataErased, 130
 - recoveryACDataReady, 131
 - recoveryDCDataReady, 131
 - resumeExperiment, 131
 - setCompRange, 132
 - setIRComp, 132
 - setLinkedChannels, 134
 - setManualModeConstantCurrent, 134
 - setManualModeConstantVoltage, 135
 - setManualModeOCP, 136
 - setManualModeSamplingInterval, 136
 - skipExperimentStep, 137
 - startManualExperiment, 137
 - startUploadedExperiment, 138
 - stopExperiment, 138
 - uploadExperimentToChannel, 139
- AisInstrumentHandler.h, 170
- AisNormalPulseVoltammetryElement, 140
 - AisNormalPulseVoltammetryElement, 142
 - getApproxMaxCurrent, 143
 - getCategory, 143
 - getEndVoltage, 143
 - getName, 143
 - getPulsePeriod, 144
 - getPulseWidth, 144
 - getStartVoltage, 144
 - getVStep, 145
 - isAutoRange, 145
 - isEndVoltageVsOCP, 145
 - isStartVoltageVsOCP, 145
 - setApproxMaxCurrent, 146
 - setAutoRange, 146
 - setEndVoltage, 146

- setEndVoltageVsOCP, 147
 - setPulsePeriod, 147
 - setPulseWidth, 147
 - setStartVoltage, 148
 - setStartVoltageVsOCP, 148
 - setVStep, 148
- AisNormalPulseVoltammetryElement.h, 179
- AisOpenCircuitElement, 149
 - AisOpenCircuitElement, 150
 - getApproxMaxVoltage, 151
 - getCategory, 151
 - getMaxDuration, 151
 - getMaxVoltage, 151
 - getMindVdt, 152
 - getMinVoltage, 152
 - getName, 152
 - getSamplingInterval, 153
 - isAutoVoltageRange, 153
 - setApproxMaxVoltage, 153
 - setAutoVoltageRange, 154
 - setMaxDuration, 154
 - setMaxVoltage, 154
 - setMindVdt, 154
 - setMinVoltage, 155
 - setSamplingInterval, 155
- AisOpenCircuitElement.h, 180
- AisSquareWaveVoltammetryElement, 155
 - AisSquareWaveVoltammetryElement, 157
 - getApproxMaxCurrent, 158
 - getCategory, 158
 - getEndVoltage, 158
 - getName, 159
 - getPulseAmp, 159
 - getPulseFreq, 159
 - getStartVoltage, 159
 - getVStep, 160
 - isAutoRange, 160
 - isEndVoltageVsOCP, 160
 - isStartVoltageVsOCP, 160
 - setApproxMaxCurrent, 161
 - setAutoRange, 161
 - setEndVoltage, 161
 - setEndVoltageVsOCP, 162
 - setPulseAmp, 162
 - setPulseFreq, 162
 - setStartVoltage, 164
 - setStartVoltageVsOCP, 164
 - setVStep, 164
- AisSquareWaveVoltammetryElement.h, 181
- AisSquidstatGlobal.h, 171
- appendElement
 - AisExperiment, 120
- appendSubExperiment
 - AisExperiment, 120
- BusyChannel
 - AisErrorCode, 117
- ChannelNotBusy
 - AisErrorCode, 117
- connectAllPluggedInDevices
 - AisDeviceTracker, 92
- ConnectionFailed
 - AisErrorCode, 117
- connectToDeviceOnComPort
 - AisDeviceTracker, 92
- DeviceCommunicationFailed
 - AisErrorCode, 117
- deviceDisconnected
 - AisDeviceTracker, 93
 - AisInstrumentHandler, 125
- DeviceNotFound
 - AisErrorCode, 117
- eraseRecoverData
 - AisInstrumentHandler, 125
- ErrorCode
 - AisErrorCode, 117
- ExperimentAlreadyPaused
 - AisErrorCode, 118
- ExperimentAlreadyRun
 - AisErrorCode, 118
- ExperimentIsEmpty
 - AisErrorCode, 117
- experimentNewElementStarting
 - AisInstrumentHandler, 126
- ExperimentNotUploaded
 - AisErrorCode, 117
- experimentPaused
 - AisInstrumentHandler, 126
- experimentResumed
 - AisInstrumentHandler, 126
- experimentStopped
 - AisInstrumentHandler, 127
- FailedToPauseExperiment
 - AisErrorCode, 117
- FailedToResumeExperiment
 - AisErrorCode, 118
- FailedToSetManualModeConstantVoltage
 - AisErrorCode, 117
- FailedToSetManualModeCurrentRange
 - AisErrorCode, 117
- FailedToStopExperiment
 - AisErrorCode, 118
- FailedToUploadExperiment
 - AisErrorCode, 118
- FirmwareNotSupported
 - AisErrorCode, 117
- getAmplitude
 - AisEISGalvanostaticElement, 107
 - AisEISPotentiostaticElement, 112
- getApproxMaxCurrent
 - AisConstantCurrentElement, 37
 - AisConstantPotElement, 47
 - AisCyclicVoltammetryElement, 68

- AisDCPotentialSweepElement, 86
- AisDiffPulseVoltammetryElement, 98
- AisNormalPulseVoltammetryElement, 143
- AisSquareWaveVoltammetryElement, 158
- getApproxMaxVoltage
 - AisConstantCurrentElement, 38
 - AisOpenCircuitElement, 151
- getBandwidthIndex
 - AisCompRange, 33
- getBiasCurrent
 - AisEISGalvanostaticElement, 107
- getBiasVoltage
 - AisEISPotentiostaticElement, 112
- getCategory
 - AisConstantCurrentElement, 38
 - AisConstantPotElement, 47
 - AisConstantPowerElement, 56
 - AisConstantResistanceElement, 62
 - AisCyclicVoltammetryElement, 68
 - AisDCCurrentSweepElement, 79
 - AisDCPotentialSweepElement, 86
 - AisDiffPulseVoltammetryElement, 98
 - AisEISGalvanostaticElement, 107
 - AisEISPotentiostaticElement, 112
 - AisExperiment, 121
 - AisNormalPulseVoltammetryElement, 143
 - AisOpenCircuitElement, 151
 - AisSquareWaveVoltammetryElement, 158
- getCompRangeName
 - AisCompRange, 33
- getConnectedDevices
 - AisDeviceTracker, 93
- getCurrent
 - AisConstantCurrentElement, 38
- getdEdt
 - AisCyclicVoltammetryElement, 69
- getDescription
 - AisExperiment, 121
- getEndFreq
 - AisEISGalvanostaticElement, 107
 - AisEISPotentiostaticElement, 113
- getEndingCurrent
 - AisDCCurrentSweepElement, 79
- getEndingPot
 - AisDCPotentialSweepElement, 86
- getEndVoltage
 - AisCyclicVoltammetryElement, 69
 - AisDiffPulseVoltammetryElement, 98
 - AisNormalPulseVoltammetryElement, 143
 - AisSquareWaveVoltammetryElement, 158
- getExperimentName
 - AisExperiment, 121
- getExperimentUTCStartTime
 - AisInstrumentHandler, 127
- getFirstVoltageLimit
 - AisCyclicVoltammetryElement, 69
- getFreeChannels
 - AisInstrumentHandler, 127
- getInstrumentHandler
 - AisDeviceTracker, 93
- getLinkedChannels
 - AisInstrumentHandler, 128
- getManualModeCurrentRangeList
 - AisInstrumentHandler, 128
- getMaxCapacity
 - AisConstantCurrentElement, 39
 - AisConstantPotElement, 47
 - AisConstantPowerElement, 56
 - AisConstantResistanceElement, 63
- getMaxCurrent
 - AisConstantPotElement, 48
- getMaxDuration
 - AisConstantCurrentElement, 39
 - AisConstantPotElement, 48
 - AisConstantPowerElement, 56
 - AisConstantResistanceElement, 63
 - AisOpenCircuitElement, 151
- getMaxVoltage
 - AisConstantCurrentElement, 39
 - AisConstantPowerElement, 56
 - AisDCCurrentSweepElement, 79
 - AisOpenCircuitElement, 151
- getMinCurrent
 - AisConstantPotElement, 48
- getMindIdt
 - AisConstantPotElement, 49
- getMindVdt
 - AisOpenCircuitElement, 152
- getMinSamplingVoltageDifference
 - AisConstantCurrentElement, 39
- getMinVoltage
 - AisConstantCurrentElement, 40
 - AisConstantPowerElement, 57
 - AisConstantResistanceElement, 63
 - AisDCCurrentSweepElement, 79
 - AisOpenCircuitElement, 152
- getName
 - AisConstantCurrentElement, 40
 - AisConstantPotElement, 49
 - AisConstantPowerElement, 57
 - AisConstantResistanceElement, 63
 - AisCyclicVoltammetryElement, 69
 - AisDCCurrentSweepElement, 80
 - AisDCPotentialSweepElement, 86
 - AisDiffPulseVoltammetryElement, 98
 - AisEISGalvanostaticElement, 108
 - AisEISPotentiostaticElement, 113
 - AisNormalPulseVoltammetryElement, 143
 - AisOpenCircuitElement, 152
 - AisSquareWaveVoltammetryElement, 159
- getNumberOfChannels
 - AisInstrumentHandler, 128
- getNumberOfCycles
 - AisCyclicVoltammetryElement, 70
- getPotential
 - AisConstantPotElement, 49

- getPower
 - AisConstantPowerElement, 57
- getPulseAmp
 - AisSquareWaveVoltammetryElement, 159
- getPulseFreq
 - AisSquareWaveVoltammetryElement, 159
- getPulseHeight
 - AisDiffPulseVoltammetryElement, 99
- getPulsePeriod
 - AisDiffPulseVoltammetryElement, 99
 - AisNormalPulseVoltammetryElement, 144
- getPulseWidth
 - AisDiffPulseVoltammetryElement, 99
 - AisNormalPulseVoltammetryElement, 144
- getResistance
 - AisConstantResistanceElement, 64
- getSamplingInterval
 - AisConstantCurrentElement, 40
 - AisConstantPotElement, 50
 - AisConstantPowerElement, 58
 - AisConstantResistanceElement, 64
 - AisCyclicVoltammetryElement, 70
 - AisDCCurrentSweepElement, 80
 - AisDCPotentialSweepElement, 87
 - AisOpenCircuitElement, 153
- getScanRate
 - AisDCCurrentSweepElement, 80
 - AisDCPotentialSweepElement, 87
- getSecondVoltageLimit
 - AisCyclicVoltammetryElement, 70
- getStabilityFactor
 - AisCompRange, 34
- getStartFreq
 - AisEISGalvanostaticElement, 108
 - AisEISPotentiostaticElement, 113
- getStartingCurrent
 - AisDCCurrentSweepElement, 80
- getStartingPot
 - AisDCPotentialSweepElement, 87
- getStartVoltage
 - AisCyclicVoltammetryElement, 70
 - AisDiffPulseVoltammetryElement, 100
 - AisNormalPulseVoltammetryElement, 144
 - AisSquareWaveVoltammetryElement, 159
- getStepsPerDecade
 - AisEISGalvanostaticElement, 108
 - AisEISPotentiostaticElement, 113
- getVStep
 - AisDiffPulseVoltammetryElement, 100
 - AisNormalPulseVoltammetryElement, 145
 - AisSquareWaveVoltammetryElement, 160
- groundFloatStateChanged
 - AisInstrumentHandler, 129
- idleDCDataReady
 - AisInstrumentHandler, 129
- InvalidChannel
 - AisErrorCode, 117
- InvalidParameters
 - AisErrorCode, 117
- isAutoRange
 - AisConstantCurrentElement, 41
 - AisConstantPotElement, 50
 - AisCyclicVoltammetryElement, 71
 - AisDCPotentialSweepElement, 87
 - AisDiffPulseVoltammetryElement, 100
 - AisNormalPulseVoltammetryElement, 145
 - AisSquareWaveVoltammetryElement, 160
- isAutoVoltageRange
 - AisConstantCurrentElement, 41
 - AisOpenCircuitElement, 153
- isBiasVoltageVsOCP
 - AisEISPotentiostaticElement, 114
- isChannelBusy
 - AisInstrumentHandler, 129
- isChannelPaused
 - AisInstrumentHandler, 130
- isCharge
 - AisConstantPowerElement, 58
- isEndVoltageVsOCP
 - AisCyclicVoltammetryElement, 71
 - AisDCPotentialSweepElement, 88
 - AisDiffPulseVoltammetryElement, 101
 - AisNormalPulseVoltammetryElement, 145
 - AisSquareWaveVoltammetryElement, 160
- isFirstVoltageLimitVsOCP
 - AisCyclicVoltammetryElement, 71
- isSecondVoltageLimitVsOCP
 - AisCyclicVoltammetryElement, 71
- isStartVoltageVsOCP
 - AisCyclicVoltammetryElement, 72
 - AisDCPotentialSweepElement, 88
 - AisDiffPulseVoltammetryElement, 101
 - AisNormalPulseVoltammetryElement, 145
 - AisSquareWaveVoltammetryElement, 160
- isVoltageVsOCP
 - AisConstantPotElement, 50
- ManualExperimentNotRunning
 - AisErrorCode, 117
- message
 - AisErrorCode, 118
- newDeviceConnected
 - AisDeviceTracker, 94
- numberOfCycles
 - AisACData, 32
- operator=
 - AisExperiment, 121
- pauseExperiment
 - AisInstrumentHandler, 130
- recoverDataErased
 - AisInstrumentHandler, 130
- recoveryACDataReady
 - AisInstrumentHandler, 131

- recoveryDCDataReady
 - AisInstrumentHandler, [131](#)
- resumeExperiment
 - AisInstrumentHandler, [131](#)
- setAmplitude
 - AisEISGalvanostaticElement, [108](#)
 - AisEISPotentiostaticElement, [114](#)
- setApproxMaxCurrent
 - AisConstantCurrentElement, [41](#)
 - AisConstantPotElement, [50](#)
 - AisCyclicVoltammetryElement, [72](#)
 - AisDCPotentialSweepElement, [88](#)
 - AisDiffPulseVoltammetryElement, [101](#)
 - AisNormalPulseVoltammetryElement, [146](#)
 - AisSquareWaveVoltammetryElement, [161](#)
- setApproxMaxVoltage
 - AisConstantCurrentElement, [42](#)
 - AisOpenCircuitElement, [153](#)
- setAutoRange
 - AisConstantCurrentElement, [42](#)
 - AisConstantPotElement, [51](#)
 - AisCyclicVoltammetryElement, [72](#)
 - AisDCPotentialSweepElement, [89](#)
 - AisDiffPulseVoltammetryElement, [102](#)
 - AisNormalPulseVoltammetryElement, [146](#)
 - AisSquareWaveVoltammetryElement, [161](#)
- setAutoVoltageRange
 - AisConstantCurrentElement, [42](#)
 - AisOpenCircuitElement, [154](#)
- setBandwidthIndex
 - AisCompRange, [34](#)
- setBiasCurrent
 - AisEISGalvanostaticElement, [109](#)
- setBiasVoltage
 - AisEISPotentiostaticElement, [114](#)
- setBiasVoltageVsOCP
 - AisEISPotentiostaticElement, [115](#)
- setCharge
 - AisConstantPowerElement, [58](#)
- setCompRange
 - AisInstrumentHandler, [132](#)
- setCompRangeName
 - AisCompRange, [34](#)
- setCurrent
 - AisConstantCurrentElement, [42](#)
- setdEdt
 - AisCyclicVoltammetryElement, [72](#)
- setDescription
 - AisExperiment, [122](#)
- setEndFreq
 - AisEISGalvanostaticElement, [109](#)
 - AisEISPotentiostaticElement, [115](#)
- setEndingCurrent
 - AisDCCurrentSweepElement, [81](#)
- setEndingPot
 - AisDCPotentialSweepElement, [89](#)
- setEndVoltage
 - AisCyclicVoltammetryElement, [73](#)
 - AisDiffPulseVoltammetryElement, [102](#)
 - AisNormalPulseVoltammetryElement, [146](#)
 - AisSquareWaveVoltammetryElement, [161](#)
- setEndVoltageVsOCP
 - AisCyclicVoltammetryElement, [73](#)
 - AisDCPotentialSweepElement, [89](#)
 - AisDiffPulseVoltammetryElement, [102](#)
 - AisNormalPulseVoltammetryElement, [147](#)
 - AisSquareWaveVoltammetryElement, [162](#)
- setExperimentName
 - AisExperiment, [122](#)
- setFirstVoltageLimit
 - AisCyclicVoltammetryElement, [73](#)
- setFirstVoltageLimitVsOCP
 - AisCyclicVoltammetryElement, [74](#)
- setIRComp
 - AisInstrumentHandler, [132](#)
- setLinkedChannels
 - AisInstrumentHandler, [134](#)
- setManualModeConstantCurrent
 - AisInstrumentHandler, [134](#)
- setManualModeConstantVoltage
 - AisInstrumentHandler, [135](#)
- setManualModeOCP
 - AisInstrumentHandler, [136](#)
- setManualModeSamplingInterval
 - AisInstrumentHandler, [136](#)
- setMaxCapacity
 - AisConstantCurrentElement, [42](#)
 - AisConstantPotElement, [51](#)
 - AisConstantPowerElement, [59](#)
 - AisConstantResistanceElement, [64](#)
- setMaxCurrent
 - AisConstantPotElement, [51](#)
- setMaxDuration
 - AisConstantCurrentElement, [43](#)
 - AisConstantPotElement, [51](#)
 - AisConstantPowerElement, [59](#)
 - AisConstantResistanceElement, [65](#)
 - AisOpenCircuitElement, [154](#)
- setMaxVoltage
 - AisConstantCurrentElement, [43](#)
 - AisConstantPowerElement, [59](#)
 - AisDCCurrentSweepElement, [81](#)
 - AisOpenCircuitElement, [154](#)
- setMinCurrent
 - AisConstantPotElement, [52](#)
- setMindIdt
 - AisConstantPotElement, [52](#)
- setMindVdt
 - AisOpenCircuitElement, [154](#)
- setMinSamplingVoltageDifference
 - AisConstantCurrentElement, [43](#)
- setMinVoltage
 - AisConstantCurrentElement, [44](#)
 - AisConstantPowerElement, [60](#)
 - AisConstantResistanceElement, [65](#)
 - AisDCCurrentSweepElement, [81](#)

- AisOpenCircuitElement, [155](#)
- setNumberOfCycles
 - AisCyclicVoltammetryElement, [74](#)
- setPotential
 - AisConstantPotElement, [52](#)
- setPower
 - AisConstantPowerElement, [60](#)
- setPulseAmp
 - AisSquareWaveVoltammetryElement, [162](#)
- setPulseFreq
 - AisSquareWaveVoltammetryElement, [162](#)
- setPulseHeight
 - AisDiffPulseVoltammetryElement, [102](#)
- setPulsePeriod
 - AisDiffPulseVoltammetryElement, [103](#)
 - AisNormalPulseVoltammetryElement, [147](#)
- setPulseWidth
 - AisDiffPulseVoltammetryElement, [103](#)
 - AisNormalPulseVoltammetryElement, [147](#)
- setResistance
 - AisConstantResistanceElement, [65](#)
- setSamplingInterval
 - AisConstantCurrentElement, [44](#)
 - AisConstantPotElement, [53](#)
 - AisConstantPowerElement, [60](#)
 - AisConstantResistanceElement, [66](#)
 - AisCyclicVoltammetryElement, [74](#)
 - AisDCCurrentSweepElement, [82](#)
 - AisDCPotentialSweepElement, [89](#)
 - AisOpenCircuitElement, [155](#)
- setScanRate
 - AisDCCurrentSweepElement, [82](#)
 - AisDCPotentialSweepElement, [90](#)
- setSecondVoltageLimit
 - AisCyclicVoltammetryElement, [75](#)
- setSecondVoltageLimitVsOCP
 - AisCyclicVoltammetryElement, [75](#)
- setStabilityFactor
 - AisCompRange, [35](#)
- setStartFreq
 - AisEISGalvanostaticElement, [109](#)
 - AisEISPotentiostaticElement, [115](#)
- setStartingCurrent
 - AisDCCurrentSweepElement, [82](#)
- setStartingPot
 - AisDCPotentialSweepElement, [90](#)
- setStartVoltage
 - AisCyclicVoltammetryElement, [75](#)
 - AisDiffPulseVoltammetryElement, [103](#)
 - AisNormalPulseVoltammetryElement, [148](#)
 - AisSquareWaveVoltammetryElement, [164](#)
- setStartVoltageVsOCP
 - AisCyclicVoltammetryElement, [76](#)
 - AisDCPotentialSweepElement, [90](#)
 - AisDiffPulseVoltammetryElement, [104](#)
 - AisNormalPulseVoltammetryElement, [148](#)
 - AisSquareWaveVoltammetryElement, [164](#)
- setStepsPerDecade
 - AisEISGalvanostaticElement, [109](#)
 - AisEISPotentiostaticElement, [115](#)
- setVoltageVsOCP
 - AisConstantPotElement, [53](#)
- setVStep
 - AisDiffPulseVoltammetryElement, [104](#)
 - AisNormalPulseVoltammetryElement, [148](#)
 - AisSquareWaveVoltammetryElement, [164](#)
- skipExperimentStep
 - AisInstrumentHandler, [137](#)
- startManualExperiment
 - AisInstrumentHandler, [137](#)
- startUploadedExperiment
 - AisInstrumentHandler, [138](#)
- stopExperiment
 - AisInstrumentHandler, [138](#)
- Success
 - AisErrorCode, [117](#)
- Unknown
 - AisErrorCode, [117](#)
- updateFirmwareOnAllAvailableDevices
 - AisDeviceTracker, [94](#)
- updateFirmwareOnComPort
 - AisDeviceTracker, [95](#)
- uploadExperimentToChannel
 - AisInstrumentHandler, [139](#)
- value
 - AisErrorCode, [118](#)