Squidstat API User Manual

Generated by Admiral Instruments LLC
July 20, 2023

1 Squidstat	API User Manual	1
2 Identifying	USB Serial Ports	3
	2.0.0.1 Introduction	3
	2.0.0.2 Windows	3
	2.0.0.3 Mac	3
	2.0.0.4 Linux	4
3 Building A	PI using CMake	5
	3.0.0.1 Introduction	5
	3.0.0.2 Clone API from Git	5
	3.0.0.3 Cmake Installtion	6
	3.0.0.4 Build project	6
4 Running tl	he API with Qt	7
	4.0.0.1 Introduction	7
	4.0.0.2 Clone API from Git	7
	4.0.0.3 Qt Installation	8
	4.0.0.4 Open Project with Qt and Cmake	9
5 Updating I	Firmware	13
	5.0.0.1 Introduction	13
	5.0.0.2 Building a Custom Experiment with Python	13
6 The Basic	s of Running Experiments	15
6.0	.1 Creating A Custom Experiment	15
6.0	.2 Controlling The Experiment	16
	6.0.2.1 Creating Control Flow Logic Specific To A Handler	17
	6.0.2.2 Connecting Slots To Device-Tracker Signals	18
7 Manual Ex	periments	21
8 Automatic	ally Update Firmware	23
	8.0.0.1 Introduction	23
	8.0.0.2 Implementation	23
	8.0.0.3 Full Example	24
9 Advanced	Control Flow	27
10 Python E	xample	31
	10.0.0.1 Introduction	31
	10.0.0.2 How To Use Squidstatlibrary with Python	31
	10.0.0.3 Building a Custom Experiment with Python	32
11 Operating	g Squidstats Remotely	35
	11.0.0.1 Introduction	35

11.0.0.2 Server Implementation	 . 35
11.0.0.3 Client Implementation	
11.0.0.4 Full Example	 . 38
12 Hierarchical Index	41
12.1 Class Hierarchy	 . 41
13 Class Index	43
13.1 Class List	 . 43
14 File Index	45
14.1 File List	 . 45
15 Class Documentation	47
15.1 AisACData Struct Reference	 . 47
15.1.1 Detailed Description	 . 48
15.1.2 Member Data Documentation	 . 48
15.1.2.1 numberOfCycles	 . 48
15.2 AisCompRange Class Reference	 . 48
15.2.1 Detailed Description	 . 49
15.2.2 Constructor & Destructor Documentation	 . 49
15.2.2.1 AisCompRange()	 . 49
15.2.3 Member Function Documentation	 . 49
15.2.3.1 getBandwidthIndex()	 . 49
15.2.3.2 getCompRangeName()	 . 50
15.2.3.3 getStabilityFactor()	 . 50
15.2.3.4 setBandwidthIndex()	 . 50
15.2.3.5 setCompRangeName()	 . 50
15.2.3.6 setStabilityFactor()	 . 51
15.3 AisConstantCurrentElement Class Reference	 . 51
15.3.1 Detailed Description	 . 53
15.3.2 Constructor & Destructor Documentation	 . 53
15.3.2.1 AisConstantCurrentElement()	 . 53
15.3.3 Member Function Documentation	 . 53
15.3.3.1 getApproxMaxCurrent()	 . 54
15.3.3.2 getApproxMaxVoltage()	 . 54
15.3.3.3 getCategory()	 . 54
15.3.3.4 getCurrent()	 . 55
15.3.3.5 getMaxCapacity()	 . 55
15.3.3.6 getMaxDuration()	 . 55
15.3.3.7 getMaxVoltage()	 . 55
15.3.3.8 getMinSamplingVoltageDifference()	 . 56
15.3.3.9 getMinVoltage()	 . 56
15.3.3.10 getName()	 . 56

15.3.3.11 getSamplingInterval()	57
15.3.3.12 isAutoRange()	57
15.3.3.13 isAutoVoltageRange()	57
15.3.3.14 setApproxMaxCurrent()	57
15.3.3.15 setApproxMaxVoltage()	58
15.3.3.16 setAutoRange()	58
15.3.3.17 setAutoVoltageRange()	58
15.3.3.18 setCurrent()	58
15.3.3.19 setMaxCapacity()	59
15.3.3.20 setMaxDuration()	59
15.3.3.21 setMaxVoltage()	59
15.3.3.22 setMinSamplingVoltageDifference()	60
15.3.3.23 setMinVoltage()	60
15.3.3.24 setSamplingInterval()	60
15.4 AisConstantPotElement Class Reference	61
15.4.1 Detailed Description	62
15.4.2 Constructor & Destructor Documentation	62
15.4.2.1 AisConstantPotElement()	63
15.4.3 Member Function Documentation	63
15.4.3.1 getApproxMaxCurrent()	63
15.4.3.2 getCategory()	63
15.4.3.3 getMaxCapacity()	64
15.4.3.4 getMaxCurrent()	64
15.4.3.5 getMaxDuration()	64
15.4.3.6 getMinCurrent()	65
15.4.3.7 getMindldt()	65
15.4.3.8 getName()	65
15.4.3.9 getPotential()	66
15.4.3.10 getSamplingInterval()	66
15.4.3.11 isAutoRange()	66
15.4.3.12 isVoltageVsOCP()	66
15.4.3.13 setApproxMaxCurrent()	66
15.4.3.14 setAutoRange()	67
15.4.3.15 setMaxCapacity()	67
15.4.3.16 setMaxCurrent()	67
15.4.3.17 setMaxDuration()	68
15.4.3.18 setMinCurrent()	68
15.4.3.19 setMindldt()	68
15.4.3.20 setPotential()	69
15.4.3.21 setSamplingInterval()	69
15.4.3.22 setVoltageVsOCP()	69
15.5 AisConstantPowerFlement Class Reference	60

15.5.1 Detailed Description	71
15.5.2 Constructor & Destructor Documentation	71
15.5.2.1 AisConstantPowerElement()	71
15.5.3 Member Function Documentation	72
15.5.3.1 getCategory()	72
15.5.3.2 getMaxCapacity()	72
15.5.3.3 getMaxDuration()	72
15.5.3.4 getMaxVoltage()	73
15.5.3.5 getMinVoltage()	73
15.5.3.6 getName()	73
15.5.3.7 getPower()	74
15.5.3.8 getSamplingInterval()	74
15.5.3.9 isCharge()	74
15.5.3.10 setCharge()	74
15.5.3.11 setMaxCapacity()	75
15.5.3.12 setMaxDuration()	75
15.5.3.13 setMaxVoltage()	75
15.5.3.14 setMinVoltage()	76
15.5.3.15 setPower()	76
15.5.3.16 setSamplingInterval()	76
15.6 AisConstantResistanceElement Class Reference	77
15.6.1 Detailed Description	78
15.6.2 Constructor & Destructor Documentation	78
15.6.2.1 AisConstantResistanceElement()	78
15.6.3 Member Function Documentation	78
15.6.3.1 getCategory()	79
15.6.3.2 getMaxCapacity()	79
15.6.3.3 getMaxDuration()	79
15.6.3.4 getMinVoltage()	79
15.6.3.5 getName()	80
15.6.3.6 getResistance()	80
15.6.3.7 getSamplingInterval()	80
15.6.3.8 setMaxCapacity()	80
15.6.3.9 setMaxDuration()	81
15.6.3.10 setMinVoltage()	81
15.6.3.11 setResistance()	81
15.6.3.12 setSamplingInterval()	82
15.7 AisCyclicVoltammetryElement Class Reference	82
15.7.1 Detailed Description	83
15.7.2 Constructor & Destructor Documentation	84
15.7.2.1 AisCyclicVoltammetryElement()	84
15.7.3 Member Function Documentation	84

15.7.3.1 getApproxMaxCurrent()	. 84
15.7.3.2 getCategory()	. 85
15.7.3.3 getdEdt()	. 85
15.7.3.4 getEndVoltage()	. 85
15.7.3.5 getFirstVoltageLimit()	. 85
15.7.3.6 getName()	. 86
15.7.3.7 getNumberOfCycles()	. 86
15.7.3.8 getSamplingInterval()	. 86
15.7.3.9 getSecondVoltageLimit()	. 86
15.7.3.10 getStartVoltage()	. 87
15.7.3.11 isAutoRange()	. 87
15.7.3.12 isEndVoltageVsOCP()	. 87
15.7.3.13 isFirstVoltageLimitVsOCP()	. 87
15.7.3.14 isSecondVoltageLimitVsOCP()	. 88
15.7.3.15 isStartVoltageVsOCP()	. 88
15.7.3.16 setApproxMaxCurrent()	. 88
15.7.3.17 setAutoRange()	. 88
15.7.3.18 setdEdt()	. 89
15.7.3.19 setEndVoltage()	. 89
15.7.3.20 setEndVoltageVsOCP()	. 89
15.7.3.21 setFirstVoltageLimit()	. 89
15.7.3.22 setFirstVoltageLimitVsOCP()	. 90
15.7.3.23 setNumberOfCycles()	. 90
15.7.3.24 setSamplingInterval()	. 90
15.7.3.25 setSecondVoltageLimit()	. 91
15.7.3.26 setSecondVoltageLimitVsOCP()	. 91
15.7.3.27 setStartVoltage()	. 91
15.7.3.28 setStartVoltageVsOCP()	. 92
15.8 AisDCCurrentSweepElement Class Reference	. 92
15.8.1 Detailed Description	. 93
15.8.2 Constructor & Destructor Documentation	. 94
15.8.2.1 AisDCCurrentSweepElement()	. 94
15.8.3 Member Function Documentation	. 95
15.8.3.1 getCategory()	. 95
15.8.3.2 getEndingCurrent()	. 95
15.8.3.3 getMaxVoltage()	. 95
15.8.3.4 getMinVoltage()	. 96
15.8.3.5 getName()	. 96
15.8.3.6 getSamplingInterval()	. 96
15.8.3.7 getScanRate()	. 96
15.8.3.8 getStartingCurrent()	. 97
15.8.3.9 setEndingCurrent()	. 97

15.8.3.10 setMaxVoltage()	97
15.8.3.11 setMinVoltage()	97
15.8.3.12 setSamplingInterval()	98
15.8.3.13 setScanRate()	98
15.8.3.14 setStartingCurrent()	98
15.9 AisDCData Struct Reference	99
15.9.1 Detailed Description	99
15.10 AisDCPotentialSweepElement Class Reference	99
15.10.1 Detailed Description	101
15.10.2 Constructor & Destructor Documentation	101
15.10.2.1 AisDCPotentialSweepElement()	101
15.10.3 Member Function Documentation	102
15.10.3.1 getApproxMaxCurrent()	102
15.10.3.2 getCategory()	102
15.10.3.3 getEndingPot()	102
15.10.3.4 getName()	103
15.10.3.5 getSamplingInterval()	103
15.10.3.6 getScanRate()	103
15.10.3.7 getStartingPot()	103
15.10.3.8 isAutoRange()	104
15.10.3.9 isEndVoltageVsOCP()	104
15.10.3.10 isStartVoltageVsOCP()	104
15.10.3.11 setApproxMaxCurrent()	104
15.10.3.12 setAutoRange()	105
15.10.3.13 setEndingPot()	105
15.10.3.14 setEndVoltageVsOCP()	105
15.10.3.15 setSamplingInterval()	106
15.10.3.16 setScanRate()	106
15.10.3.17 setStartingPot()	106
15.10.3.18 setStartVoltageVsOCP()	106
15.11 AisDeviceTracker Class Reference	107
15.11.1 Detailed Description	108
15.11.2 Member Function Documentation	108
15.11.2.1 connectAllPluggedInDevices()	108
15.11.2.2 connectToDeviceOnComPort()	108
15.11.2.3 deviceDisconnected	109
15.11.2.4 getConnectedDevices()	109
15.11.2.5 getInstrumentHandler()	109
15.11.2.6 newDeviceConnected	110
15.11.2.7 updateFirmwareOnAllAvailableDevices()	110
15.11.2.8 updateFirmwareOnComPort()	111
15.12 AisDiffPulseVoltammetryElement Class Reference	111

15.12.1 Detailed Description	113
15.12.2 Constructor & Destructor Documentation	113
15.12.2.1 AisDiffPulseVoltammetryElement()	113
15.12.3 Member Function Documentation	114
15.12.3.1 getApproxMaxCurrent()	114
15.12.3.2 getCategory()	114
15.12.3.3 getEndVoltage()	114
15.12.3.4 getName()	115
15.12.3.5 getPulseHeight()	115
15.12.3.6 getPulsePeriod()	115
15.12.3.7 getPulseWidth()	116
15.12.3.8 getStartVoltage()	116
15.12.3.9 getVStep()	116
15.12.3.10 isAutoRange()	117
15.12.3.11 isEndVoltageVsOCP()	117
15.12.3.12 isStartVoltageVsOCP()	117
15.12.3.13 setApproxMaxCurrent()	117
15.12.3.14 setAutoRange()	118
15.12.3.15 setEndVoltage()	118
15.12.3.16 setEndVoltageVsOCP()	118
15.12.3.17 setPulseHeight()	119
15.12.3.18 setPulsePeriod()	119
15.12.3.19 setPulseWidth()	119
15.12.3.20 setStartVoltage()	120
15.12.3.21 setStartVoltageVsOCP()	120
15.12.3.22 setVStep()	120
15.13 AisEISGalvanostaticElement Class Reference	121
15.13.1 Detailed Description	122
15.13.2 Constructor & Destructor Documentation	122
15.13.2.1 AisEISGalvanostaticElement()	122
15.13.3 Member Function Documentation	123
15.13.3.1 getAmplitude()	123
15.13.3.2 getBiasCurrent()	123
15.13.3.3 getCategory()	123
15.13.3.4 getEndFreq()	124
15.13.3.5 getName()	124
15.13.3.6 getStartFreq()	124
15.13.3.7 getStepsPerDecade()	124
15.13.3.8 setAmplitude()	124
15.13.3.9 setBiasCurrent()	125
15.13.3.10 setEndFreq()	125
15.13.3.11 setStartFreq()	125

15.13.3.12 setStepsPerDecade()	126
15.14 AisEISPotentiostaticElement Class Reference	126
15.14.1 Detailed Description	127
15.14.2 Constructor & Destructor Documentation	127
15.14.2.1 AisEISPotentiostaticElement()	128
15.14.3 Member Function Documentation	128
15.14.3.1 getAmplitude()	128
15.14.3.2 getBiasVoltage()	128
15.14.3.3 getCategory()	129
15.14.3.4 getEndFreq()	129
15.14.3.5 getName()	129
15.14.3.6 getStartFreq()	129
15.14.3.7 getStepsPerDecade()	130
15.14.3.8 isBiasVoltageVsOCP()	130
15.14.3.9 setAmplitude()	130
15.14.3.10 setBiasVoltage()	130
15.14.3.11 setBiasVoltageVsOCP()	131
15.14.3.12 setEndFreq()	131
15.14.3.13 setStartFreq()	131
15.14.3.14 setStepsPerDecade()	132
15.15 AisErrorCode Class Reference	132
15.15.1 Detailed Description	133
15.15.2 Member Enumeration Documentation	133
15.15.2.1 ErrorCode	133
15.15.3 Member Function Documentation	134
15.15.3.1 message()	134
15.15.3.2 value()	134
15.16 AisExperiment Class Reference	135
15.16.1 Detailed Description	135
15.16.2 Constructor & Destructor Documentation	135
15.16.2.1 AisExperiment()	135
15.16.3 Member Function Documentation	136
15.16.3.1 appendElement()	136
15.16.3.2 appendSubExperiment()	136
15.16.3.3 getCategory()	137
15.16.3.4 getDescription()	137
15.16.3.5 getExperimentName()	137
15.16.3.6 operator=()	137
15.16.3.7 setDescription()	138
15.16.3.8 setExperimentName()	138
15.17 AisExperimentNode Struct Reference	138
15.17.1 Detailed Description	139

15.18 AisInstrumentHandler Class Reference	139
15.18.1 Detailed Description	141
15.18.2 Member Function Documentation	141
15.18.2.1 activeACDataReady	141
15.18.2.2 activeDCDataReady	141
15.18.2.3 deviceDisconnected	142
15.18.2.4 eraseRecoverData()	142
15.18.2.5 experimentNewElementStarting	142
15.18.2.6 experimentPaused	142
15.18.2.7 experimentResumed	144
15.18.2.8 experimentStopped	144
15.18.2.9 getExperimentUTCStartTime()	144
15.18.2.10 getFreeChannels()	145
15.18.2.11 getLinkedChannels()	145
15.18.2.12 getManualModeCurrentRangeList()	145
15.18.2.13 getNumberOfChannels()	146
15.18.2.14 groundFloatStateChanged	146
15.18.2.15 hasBipolarMode()	146
15.18.2.16 idleDCDataReady	147
15.18.2.17 isChannelBusy()	147
15.18.2.18 isChannelPaused()	147
15.18.2.19 pauseExperiment()	148
15.18.2.20 recoverDataErased	148
15.18.2.21 recoveryACDataReady	148
15.18.2.22 recoveryDCDataReady	149
15.18.2.23 resumeExperiment()	149
15.18.2.24 setBipolarLinkedChannels()	150
15.18.2.25 setCompRange()	150
15.18.2.26 setIRComp()	151
15.18.2.27 setLinkedChannels()	151
15.18.2.28 setManualModeConstantCurrent()	152
15.18.2.29 setManualModeConstantVoltage() [1/2]	152
15.18.2.30 setManualModeConstantVoltage() [2/2]	153
15.18.2.31 setManualModeOCP()	153
15.18.2.32 setManualModeSamplingInterval()	154
15.18.2.33 skipExperimentStep()	154
15.18.2.34 startManualExperiment()	155
15.18.2.35 startUploadedExperiment()	155
15.18.2.36 stopExperiment()	156
15.18.2.37 uploadExperimentToChannel() [1/2]	156
15.18.2.38 uploadExperimentToChannel() [2/2]	157
15.19 AisNormalPulseVoltammetryElement Class Reference	158

15.19.1 Detailed Description	159
15.19.2 Constructor & Destructor Documentation	160
15.19.2.1 AisNormalPulseVoltammetryElement()	160
15.19.3 Member Function Documentation	160
15.19.3.1 getApproxMaxCurrent()	161
15.19.3.2 getCategory()	161
15.19.3.3 getEndVoltage()	161
15.19.3.4 getName()	161
15.19.3.5 getPulsePeriod()	162
15.19.3.6 getPulseWidth()	162
15.19.3.7 getStartVoltage()	162
15.19.3.8 getVStep()	163
15.19.3.9 isAutoRange()	163
15.19.3.10 isEndVoltageVsOCP()	163
15.19.3.11 isStartVoltageVsOCP()	164
15.19.3.12 setApproxMaxCurrent()	164
15.19.3.13 setAutoRange()	164
15.19.3.14 setEndVoltage()	164
15.19.3.15 setEndVoltageVsOCP()	165
15.19.3.16 setPulsePeriod()	165
15.19.3.17 setPulseWidth()	165
15.19.3.18 setStartVoltage()	166
15.19.3.19 setStartVoltageVsOCP()	166
15.19.3.20 setVStep()	166
15.20 AisOpenCircuitElement Class Reference	167
15.20.1 Detailed Description	168
15.20.2 Constructor & Destructor Documentation	168
15.20.2.1 AisOpenCircuitElement()	168
15.20.3 Member Function Documentation	169
15.20.3.1 getApproxMaxVoltage()	169
15.20.3.2 getCategory()	169
15.20.3.3 getMaxDuration()	169
15.20.3.4 getMaxVoltage()	170
15.20.3.5 getMindVdt()	170
15.20.3.6 getMinVoltage()	170
15.20.3.7 getName()	171
15.20.3.8 getSamplingInterval()	171
15.20.3.9 isAutoVoltageRange()	171
15.20.3.10 setApproxMaxVoltage()	171
15.20.3.11 setAutoVoltageRange()	172
15.20.3.12 setMaxDuration()	172
15.20.3.13 setMaxVoltage()	172

15.20.3.14 setMindVdt()	172
15.20.3.15 setMinVoltage()	173
15.20.3.16 setSamplingInterval()	173
15.21 AisSquareWaveVoltammetryElement Class Reference	173
15.21.1 Detailed Description	175
15.21.2 Constructor & Destructor Documentation	175
15.21.2.1 AisSquareWaveVoltammetryElement()	175
15.21.3 Member Function Documentation	176
15.21.3.1 getApproxMaxCurrent()	176
15.21.3.2 getCategory()	176
15.21.3.3 getEndVoltage()	177
15.21.3.4 getName()	177
15.21.3.5 getPulseAmp()	177
15.21.3.6 getPulseFreq()	177
15.21.3.7 getStartVoltage()	178
15.21.3.8 getVStep()	178
15.21.3.9 isAutoRange()	178
15.21.3.10 isEndVoltageVsOCP()	178
15.21.3.11 isStartVoltageVsOCP()	179
15.21.3.12 setApproxMaxCurrent()	179
15.21.3.13 setAutoRange()	179
15.21.3.14 setEndVoltage()	179
15.21.3.15 setEndVoltageVsOCP()	180
15.21.3.16 setPulseAmp()	180
15.21.3.17 setPulseFreq()	180
15.21.3.18 setStartVoltage()	182
15.21.3.19 setStartVoltageVsOCP()	182
15.21.3.20 setVStep()	182
16 File Documentation	185
	185
16.2 AisDataPoints.h	185
16.3 AisDeviceTracker.h	186
16.4 AisErrorCode.h	187
16.5 AisExperiment.h	187
16.6 AisInstrumentHandler.h	188
16.7 AisSquidstatGlobal.h	189
16.8 AisAbstractElement.h	189
16.9 AisConstantCurrentElement.h	190
16.10 AisConstantPotElement.h	191
16.11 AisConstantPowerElement.h	192
16.12 AisConstantResistanceElement.h	100

In	Index							
	16.21 AisSquareWaveVoltammetryElement.h	. 199						
	16.20 AisOpenCircuitElement.h	. 198						
	16.19 AisNormalPulseVoltammetryElement.h	. 197						
	16.18 AisEISPotentiostaticElement.h	. 197						
	16.17 AisEISGalvanostaticElement.h	. 196						
	16.16 AisDiffPulseVoltammetryElement.h	. 195						
	16.15 AisDCPotentialSweepElement.h	. 195						
	16.14 AisDCCurrentSweepElement.h	. 194						
	16.13 AisCyclicVoltammetryElement.h	. 193						

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.

Identifying USB Serial Ports

2.0.0.1 Introduction

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

2.0.0.2 Windows

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

2.0.0.3 Mac

- 1. Connect the device to the computer via USB and power on the device.
- 2. Open Terminal. You can open it by going to 'Applications' -> 'Utilities' -> 'Terminal.' Alternatively, you can use Spotlight Search (press Cmd + Space) and type "Terminal" to launch.
- 3. List the serial devices. In the Terminal window, enter the following command and press Enter:
 - ls /dev/cu *
- 4. Identify the USB serial port. Look for the entry in the list that corresponds to your USB device. Example output:

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

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

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

2.0.0.4 Linux

- 1. Connect the device to the computer via USB and power on the device.
- 2. Open a Terminal window by pressing Ctrl + Alt + T.
- 3. Execute the ls /dev command. In the Terminal, enter the following command and press Enter:

```
ls /dev | grep tty
```

- 4. Identify the USB serial port. Look for the lines that refer to USB devices (e.g., "ttyACM0" or "ttyUSB1"). If there are multiple entries, power cycle or disconnect the device and enter the same command to determine which is correct.
- 5. When referring to the serial port from the example above in the API, the format for the entry would be tty \leftarrow ACM0.

Building API using CMake

3.0.0.1 Introduction

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

3.0.0.2 Clone API from Git

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

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

The result in the command prompt will look like this:

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

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

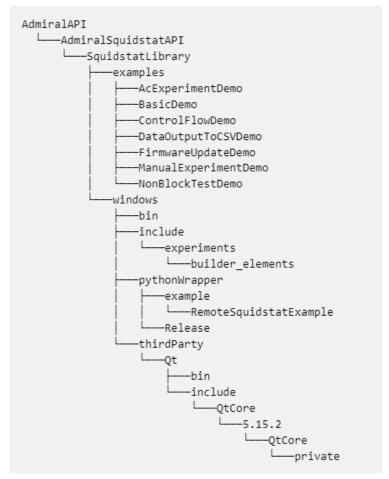


Figure 3.1 API Bundler Directory Structure

3.0.0.3 Cmake Installtion

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

3.0.0.4 Build project

- 1. Go to the directory using cd AdmiralAPI.
- 2. Open command prompt. Type the command below. This command will generate the build. It will take compiler which is available on your computer. Make sure on Windows you have the MSVC 64 compiler, and on Mac You have the Clang compiler.
 cmake -B build -S "AdmiralSquidstatAPI/SquidstatLibrary/"

Note: If you want use a different build generator, type the name of that generator followed by -G. You can check the build generator option with the command cmake -G.

3. Build the project using the command below, which will compile all examples present the in Squidstat← Library directory.

```
cmake --build ./build
```

Running the API with Qt

4.0.0.1 Introduction

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

4.0.0.2 Clone API from Git

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

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

The result in the command prompt will look like this:

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

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

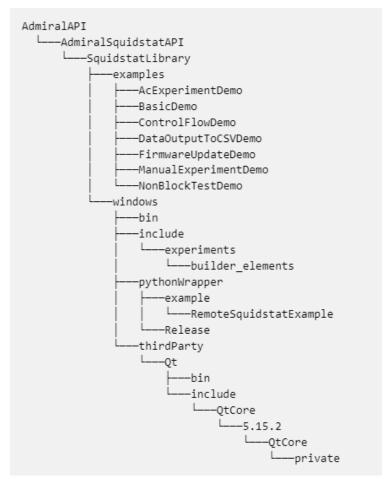


Figure 4.1 API Bundler Directory Structure

4.0.0.3 Qt Installation.

- 1. Download Qt by clicking here. To compile the API with Qt, it is required to also install the MSVC 64-bit compiler kit on Windows, Dekstop GCC 64bit on Linux, and Clang 64 on Mac during the Qt installation process.
 - Enter your Qt login account information. If you don't have an account, you can sign up and create a new one to proceed with the installation.
 - During the installation process, please ensure that you add at least one of the following components: MSVC2019 64-bit or any MSVC**** 64-bit kit on Windows, Dekstop GCC 64bit on Linux, and Clang 64 on Mac.

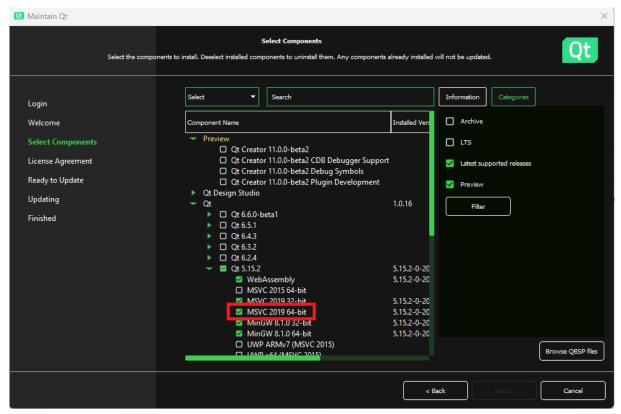


Figure 4.2 MSVC kit selection

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

4.0.0.4 Open Project with Qt and Cmake

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

```
AdmiralAPI

L---AdmiralSquidstatAPI

README.md

L---SquidstatLibrary

CMakeLists.txt

refman.pdf
```

Figure 4.3 Top Level Cmakelist file

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

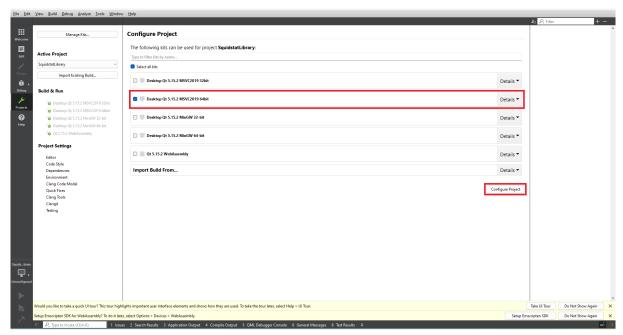


Figure 4.4 MSVC kit check mark

- 4. You can check the General Message section located in the footer. CMake will automatically configure the project, including the required libraries and header files.
- 5. The project solution will look like the image below.

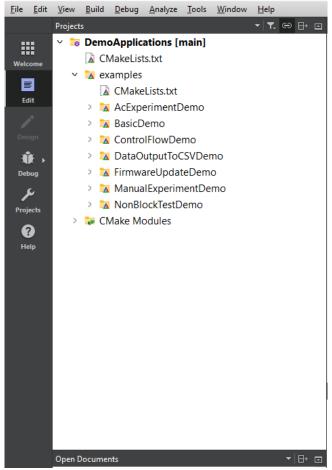


Figure 4.5 Qt project solution image

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

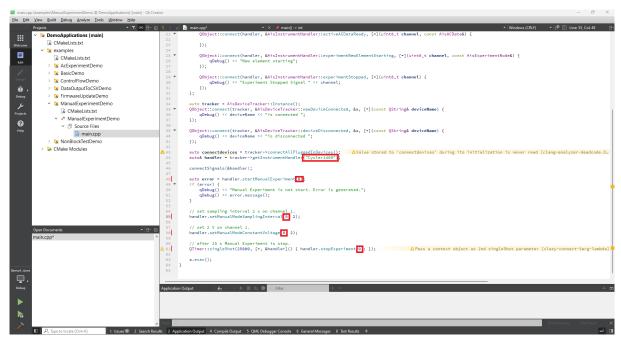


Figure 4.6 Qt manual experiment code

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

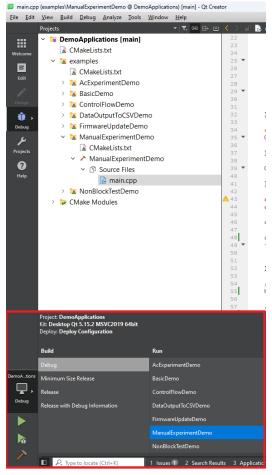


Figure 4.7 Qt selection of Debug and Release

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

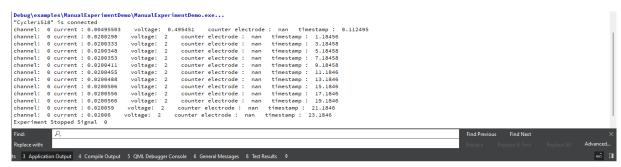


Figure 4.8 Qt output window

Updating Firmware

5.0.0.1 Introduction

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

We will go through an example.

5.0.0.2 Building a Custom Experiment with Python

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

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

```
\textbf{5.0.0.2.2} \quad \textbf{Connect the Notification signal.} \quad \texttt{QObject::connect(tracker,}
```

```
&AisDeviceTracker::firmwareUpdateNotification, &a, [=](const QString& message) {
   gInfo() « message;
```

5.0.0.2.3 request to update the firmware to all connected device. auto nmberOfDevice =

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

```
5.0.0.2.4 request to update the firmware to specific device using comport. QRegExp rx("^[Cc][Oo][Mm][0-9]+$"); if (rx.exactMatch(comPort) == false) {
   qInfo() « " Arguments is not valid. Example: " «
       QFileInfo(QCoreApplication::applicationFilePath()).fileName() « " COM3";
   auto error = tracker->updateFirmwareOnComPort(comPort);
   if (error) {
      qInfo() « error.message();
```

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

```
#include "AisDeviceTracker.h"
#include <QCoreApplication>
#include <qdebug.h>
#include <qfileinfo.h>
int main(int argc, char* argv[])
    QCoreApplication a(argc, argv);
    auto tracker = AisDeviceTracker::Instance();
    QObject::connect(tracker, &AisDeviceTracker::firmwareUpdateNotification, &a, [=](const QString& message)
```

14 Updating Firmware

```
qInfo() « message;
});
if (argc == 1) {
    auto nmberOfDevice = tracker->updateFirmwareOnAllAvailableDevices();
    if (nmberOfDevice == 0) {
        qInfo() « "Firmware update is not start in any of device";
    } else {
        qInfo() « "Firmware update start in " « nmberOfDevice « "device.";
}

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

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

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

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

The Basics of Running Experiments

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

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

6.0.1 Creating A Custom Experiment

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

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

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

Let us go through it step by step:

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

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

Note

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

We create another element of a different type.

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

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

Note

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

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

auto subExperiment = std::make_shared<AisExperiment>(); // this line creates a custom experiment, intended to be used as a sub-subExperiment

subExperiment.appendElement(ccElement, 2); // append the CC element to the sub-experiment and set it to run 2 times

subExperiment.appendElement(cvElement, 3); // append the CV element to the sub-experiment and set it to run 3 times

customExperiment->appendSubExperiment(&subExperiment, 2); // append the sub-experiment to the main experiment and set the sub-experiment to run 2 times.

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

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

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

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

6.0.2 Controlling The Experiment

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

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

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

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

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

6.0.2.1 Creating Control Flow Logic Specific To A Handler

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

```
auto connectHandlerSignals = [=](AisInstrumentHandler* handler) {
    QObject::connect(handler, &AisInstrumentHandler::activeDCDataReady, [=] (uint8_t channel, const
      AisDCData& data) {
        // do something when DC data are received, such as writing to a CSV file output
        // THIS IS WHERE YOU RECEIVE DC DATA FROM THE DEVICE
        //example: print the data to the standard output as follows:
qDebug() « "channel: " « (int)channel « "current :" « data.current « "
                                                                                      voltage:
      data.workingElectrodeVoltage « "
                                           counter electrode : " « data.counterElectrodeVoltage « "
                   " « data.timestamp;
      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&) {
        \ensuremath{//}\xspace 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:

```
OString 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 = (OString(OStandardPaths::writableLocation(OStandardPaths::DesktopLocation)) + name);
        OFile file(filePath);
          f (!file.open(QIODevice::WriteOnly | QIODevice::Text)) // overwrite existing files with the same
             return;
        OTextStream 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;
         if (!file.open(OIODevice::Append | OIODevice::WriteOnly | OIODevice::Text))
             return;
         QTextStream out(&file);
         out « data.timestamp « ","
             « data.counterElectrodeVoltage « ",'
             « data.workingElectrodeVoltage « ","
             « data.current
             « "\n";
         file.close();
    QObject::connect(handler, &AisInstrumentHandler::activeACDataReady, [=] (uint8_t channel, const
      AisACData& data) {
        qDebug() « data.frequency « " \,
                                                    " « data.absoluteImpedance « "
                                                                                                " « data.phaseAngle;
    QObject::connect(handler, &AisInstrumentHandler::experimentStopped, [=](uint8_t channel) {
```

```
qDebug() « "Experiment Completed Signal " « channel; \});
```

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

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

6.0.2.2 Connecting Slots To Device-Tracker Signals

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

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

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

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

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

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

Note

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

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

tracker->connectAllPluggedInDevices();

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

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

Finally, we can start the application by calling: app.exec();

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

The Basics of Running Experiment	The	Basics	of Rur	nning E	Experime	nts
----------------------------------	-----	---------------	--------	---------	----------	-----

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: " «
     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
});
```

22 Manual Experiments

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();

Automatically Update Firmware

8.0.0.1 Introduction

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

8.0.0.2 Implementation

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

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

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

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

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

8.0.0.3 Full Example

```
#include "AisInstrumentHandler.h"
#include "AisDeviceTracker.h"
#include "AisExperiment.h'
#include "experiments/builder_elements/AisConstantCurrentElement.h"
#include <QCoreApplication>
#include <QThread>
#include <QDebug>
#define COMPORT "COM5"
#define CHANNEL 0
int main()
   int args;
   QCoreApplication a(args, nullptr);
   auto tracker = AisDeviceTracker::Instance();
   // Custom Experiment with one constant current element
   std::shared_ptr<AisExperiment> experiment = std::make_shared<AisExperiment>();
   AisConstantCurrentElement ccElement(1, 1, 10);
   experiment->appendElement(ccElement, 1);
      This set up the signals and slots for each device that gets connected
   auto createLogic = [=](const AisInstrumentHandler* handler) {
       Object::connect(handler, &AisInstrumentHandler::activeDCDataReady, [=](uint8 t channel, const
     AisDCData& data) {
           auto utcTime = handler->getExperimentUTCStartTime(0);
                                                                 " « data.workingElectrodeVoltage « "
           qDebug() « "current :" « data.current « "
                                                       voltage:
     counter electrode :
               « data.counterElectrodeVoltage « " timestamp : " « data.timestamp
               « " start UTC time:
                                    " « qSetRealNumberPrecision(20) « utcTime;
       QObject::connect(handler, &AisInstrumentHandler::experimentNewElementStarting, [=] (uint8_t channel,
     const AisExperimentNode& data) {
           " « data.substepNumber;
       Object::connect(handler, &AisInstrumentHandler::experimentStopped, [=] (uint8_t channel) {
           qDebug() « "Experiment Completed on channel" « channel;
   \dot{//} When a device is connected, create the signals and slots to print status messages, and then start the
     experiment.
   QObject::connect(tracker, &AisDeviceTracker::newDeviceConnected, &a, [=](const QString& deviceName) {
       auto& handler = tracker->getInstrumentHandler(deviceName);
       createLogic(&handler);
       handler.uploadExperimentToChannel(CHANNEL, experiment);
       qDebug() « "Starting experiment on" « deviceName « "channel" « CHANNEL+1;
       handler.startUploadedExperiment(CHANNEL);
   /// While a device is updating firmware, print out the messages.
// When the update is complete, connect to the device, which will start the experiment
   QObject::connect(tracker, &AisDeviceTracker::firmwareUpdateNotification, [=](const QString& message) {
       qInfo() « message;
       if (message.contains("firmware is updated."))
           const int retryCount = 3;
            // Give the Squidstat some time to reconnect
           AisErrorCode error(AisErrorCode::ConnectionFailed);
           for (int i = 0; i < retryCount && error == error.ConnectionFailed; i++)</pre>
               QThread::sleep(1); //Give the last Squidstat a moment to re-establish the comport
               error = tracker->connectToDeviceOnComPort(COMPORT);
           if (error != error.Success) {
               qDebug() « "Error: " « error.message();
```

Automaticall	v U	pdate	Firmware
--------------	-----	-------	----------

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
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
```

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

28 Advanced Control Flow

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

```
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
          ^{\prime}/ 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);
                  });
                     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
                  OObject::connect(timer, &OTimer::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.

```
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. \leftarrow html#static-public-members

30 Advanced Control Flow

Python Example

10.0.0.1 Introduction

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

10.0.0.2 How To Use Squidstatlibrary with Python.

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

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

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

32 Python Example

10.0.0.3 Building a Custom Experiment with Python

10.0.0.3.1 Import all the required basic modules from SquidstatPyLibrary. import sys

```
Import struct
from PySide2.QtWidgets import QApplication
from SquidstatPyLibrary import AisDeviceTracker
from SquidstatPyLibrary import AisCompRange
from SquidstatPyLibrary import AisDCData
from SquidstatPyLibrary import AisExperimentNode
from SquidstatPyLibrary import AisExperimentNode
from SquidstatPyLibrary import AisExperiment
from SquidstatPyLibrary import AisExperiment
from SquidstatPyLibrary import AisExperiment
from SquidstatPyLibrary import AisInstrumentHandler
```

10.0.0.3.2 Import experiment modules depending on the requirement. from SquidstatPyLibrary import

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

10.0.0.3.3 Create the custom experiment. experiment = AisExperiment();

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

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

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

```
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
{\tt from \ SquidstatPyLibrary \ import \ AisErrorCode}
from SquidstatPyLibrary import AisExperiment
from SquidstatPyLibrary import AisInstrumentHandler
from SquidstatPyLibrary import AisConstantPotElement
from SquidstatPyLibrary import AisEISPotentiostaticElement
from SquidstatPyLibrary import AisConstantCurrentElement
app = QApplication()
tracker = AisDeviceTracker.Instance()
tracker.newDeviceConnected.connect(lambda deviceName: print("Device is Connected: %s" % deviceName))
tracker.connectToDeviceOnComPort("COM19")
handler = tracker.getInstrumentHandler("Ace1102");
"{:.9f}".format(data.workingElectrodeVoltage)))
handler.activeACDataReady.connect(lambda channel, data: print("frequency:", "{:.9f}".format(data.frequency), "absoluteImpedance: ", "{:.9f}".form
                                                                    ", "{:.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(cvElement, 1)
experiment.appendElement(cvElement, 1)
experiment.appendElement(subExperiment, 2)
experiment.appendElement(eisElement, 1)
handler.uploadExperimentToChannel(0, experiment)
handler.startUploadedExperiment(0)
sys.exit(app.exec_())
```

34 Python Example

Operating Squidstats Remotely

11.0.0.1 Introduction

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

11.0.0.2 Server Implementation

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

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

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

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

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

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

```
# This will build a start the Open Circuit Potential experiment
def start_ocp_experiment(handler, durationSec=60):
```

```
# Create an experiment with elements
experiment = AisExperiment()
ocpElement = AisOpenCircuitElement(durationSec, 1)
experiment.appendElement(ocpElement, 1)
# Upload the experiment to channel 0
error = handler.uploadExperimentToChannel(0, experiment)
if error.value() != AisErrorCode.ErrorCode.Success:
    return error
# Start the experiment
return(handler.startUploadedExperiment(0))
```

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

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

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

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

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

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

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

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

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

```
# Create the TCP/IP socket and bind it to our host
print("Starting server...")
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
activeSockets.append(server_socket)
server_socket.bind((HOST, PORT))
# Listen for incoming connections
server_socket.listen(1)
# Accept a client connection
client_socket, client_address = server_socket.accept()
activeSockets.append(client_socket)
# Connect the signals to send data to the client
handler.activeDCDataReady.connect(lambda channel, data: send_data_to_client(client_socket, "DCData", data)) handler.activeACDataReady.connect(lambda channel, data: send_data_to_client(client_socket, "ACData", data))
handler.experimentNewElementStarting.connect(lambda channel, data: send data to client(client socket,
       "NewElement", data))
handler.experimentStopped.connect(lambda channel: send_data_to_client(client_socket, "ExperimentCompleted",
# Start the listening process in a separate thread
listening_thread = threading.Thread(target=handle_client, args=(handler, client_socket))
listening_thread.start()
```

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

11.0.0.3 Client Implementation

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

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

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

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

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

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

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

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

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

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

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

11.0.0.4 Full Example

```
11.0.0.4.1 TCP_Server.py import os
import socket
import threading
from PvSide2.OtWidgets 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)
    \mbox{\tt\#} Upload the experiment to channel \mbox{\tt 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)
        #print("Invalid command:", command)
```

```
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:
         # Check if the client has closed the connection
         if not data:
         # 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):
    for socket in activeSockets:
        socket.close()
    app.quit()
    os._exit(1)
# Create the device tracker and connect to the Squidstat we will be using
print(f"Attempting to connect to the Squidstat {SQUIDNAME} on {SQUIDCOMPORT}...")
tracker = AisDeviceTracker.Instance()
tracker.newDeviceConnected.connect(lambda deviceName: print("Device is Connected: %s" % deviceName))
error = tracker.connectToDeviceOnComPort(SQUIDCOMPORT)
if error.value() != AisErrorCode.ErrorCode.Success:
    print(error.message())
    exit()
# Create the instrument handler
handler = tracker.getInstrumentHandler(SQUIDNAME)
print("Connection successful\n")
# Create the TCP/IP socket and bind it to our host
print("Starting server...")
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
activeSockets.append(server socket)
server_socket.bind((HOST, PORT))
# Listen for incoming connections
server_socket.listen(1)
print("Server started successfully. Waiting for client connection...")
terminal_thread = threading.Thread(target=terminate_program)
terminal_thread.start()
# Accept a client connection
client_socket, client_address = server_socket.accept()
activeSockets.append(client_socket)
\# Connect the signals to send data to the client
handler.activeDCDataReady.connect(lambda channel, data: send_data_to_client(client_socket, "DCData", data)) handler.activeACDataReady.connect(lambda channel, data: send_data_to_client(client_socket, "ACData", data))
```

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

Hierarchical Index

12.1 Class Hierarchy

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

isACData	47
isCompRange	48
isConstantCurrentElement	51
isConstantPotElement	61
isConstantPowerElement	69
isConstantResistanceElement	77
isCyclicVoltammetryElement	82
isDCCurrentSweepElement	92
isDCData	99
isDCPotentialSweepElement	99
isDiffPulseVoltammetryElement	111
isEISGalvanostaticElement	121
isEISPotentiostaticElement	126
isErrorCode	132
isExperiment	135
isExperimentNode	138
isNormalPulseVoltammetryElement	158
isOpenCircuitElement	167
isSquareWaveVoltammetryElement	173
Object	
AisDeviceTracker	107
AisInstrumentHandler	139

42 Hierarchical Index

Class Index

13.1 Class List

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

AisAC	Data	
	Structure containing AC data information	47
AisCo	mpRange	
	This class has advanced options controlling the device stability including the bandwidth index	
	and the stability factor	48
AisCo	nstantCurrentElement	
	Experiment that simulates a constant current flow with more advance options for stopping the experiment.	
	51	
AisCo	nstantPotElement	
	Experiment that simulates a constant applied voltage.	
	61	
AisCo	nstantPowerElement	
	This experiment simulates a constant power, charge or discharge".	
	69	
AisCo	nstantResistanceElement	
	This element/experiment simulates a constant resistance load.	
A:-C···	77	
AISCY	clicVoltammetryElement	
	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 num-	
	ber of cycles	82
AicDC	CurrentSweepElement	02
AISDO	This experiment performs a DC current sweep from the starting current to the ending current	
	which progresses linearly according to the scan rate	92
AisDC		32
711020	Structure containing DC data information	99
AisDC	PotentialSweepElement	
	This experiment performs a DC potential sweep from the starting current to the ending current	
	which progresses linearly according to the scan rate	99
AisDe	viceTracker	
	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	107

44 Class Index

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	111
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	121
AisEISPotentiostaticElement	121
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	100
number of frequency steps per decade	126
AisErrorCode	400
This class contains the possible error codes returned to the user when working with the API	132
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	135
AisExperimentNode	
Structure containing some information regarding the running element	138
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	139
AisNormalPulseVoltammetryElement	100
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	158
AisOpenCircuitElement	
This experiment observes the open circuit potential of the working electrode for a specific	
period of time.	
167	
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	173

File Index

14.1 File List

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

AisCompRange.h
AisDataPoints.h
AisDeviceTracker.h
AisErrorCode.h
AisExperiment.h
AisInstrumentHandler.h
AisSquidstatGlobal.h
AisAbstractElement.h
AisConstantCurrentElement.h
AisConstantPotElement.h
AisConstantPowerElement.h
AisConstantResistanceElement.h
AisCyclicVoltammetryElement.h
AisDCCurrentSweepElement.h
AisDCPotentialSweepElement.h
AisDiffPulseVoltammetryElement.h
AisEISGalvanostaticElement.h
AisEISPotentiostaticElement.h
AisNormalPulseVoltammetryElement.h
AisOpenCircuitElement.h
Ais Square Wayo Voltammetry Floment h

46 File Index

Class Documentation

15.1 AisACData Struct Reference

a structure containing AC data information.

#include <AisDataPoints.h>

Public Attributes

double timestamp

the time at which the AC data arrived.

· double frequency

the applied frequency in Hz.

• double absoluteImpedance

the magnitude of the complex impedance.

• double realImpedance

the real part of the complex impedance.

• double imagImpedance

the imaginary part of the complex impedance.

· double phaseAngle

the phase angle between the real and the imaginary parts of the impedance.

• double totalHarmonicDistortion

the percentage of the total harmonic distortion in the AC signal.

• double numberOfCycles

the number of cycles specific to the reported frequency.

double workingElectrodeDCVoltage

the DC working electrode voltage in volts.

double DCCurrent

the DC electric current value in Amps

• double currentAmplitude

the amplitude of the AC current.

• double voltageAmplitude

the amplitude of the AC voltage.

48 Class Documentation

15.1.1 Detailed Description

a structure containing AC data information.

15.1.2 Member Data Documentation

15.1.2.1 numberOfCycles

```
double AisACData::numberOfCycles
```

the number of cycles specific to the reported frequency.

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

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

· AisDataPoints.h

15.2 AisCompRange Class Reference

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

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

Public Member Functions

- AisCompRange (const QString &compRangeName, uint8_t bandwidthIndex, uint8_t stabilityFactor)
 constructor for the compensation-range object.
- AisCompRange (const AisCompRange &)

copy constructor for the compensation-range object.

uint8_t getBandwidthIndex () const

get the value set for the bandwidth index.

void setBandwidthIndex (uint8_t index)

set the index value for the bandwidth.

• uint8_t getStabilityFactor () const

get the value set for the stability factor.

void setStabilityFactor (uint8_t factor)

set a value for the stability factor.

void setCompRangeName (const QString &compRangeName)

set a name for the compensation range for reference purposes.

• const QString & getCompRangeName () const

get the name set for the compensation range.

15.2.1 Detailed Description

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

See also

```
setBandwidthIndex
setStabilityFactor
```

15.2.2 Constructor & Destructor Documentation

15.2.2.1 AisCompRange()

constructor for the compensation-range object.

Parameters

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

See also

```
setBandwidthIndex
setStabilityFactor
```

15.2.3 Member Function Documentation

15.2.3.1 getBandwidthIndex()

```
uint8_t AisCompRange::getBandwidthIndex ( ) const
get the value set for the bandwidth index.
```

Returns

the set value for the bandwidth index.

See also

setBandwidthIndex

50 Class Documentation

15.2.3.2 getCompRangeName()

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

get the name set for the compensation range.

Returns

the name set for the compensation range.

15.2.3.3 getStabilityFactor()

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

get the value set for the stability factor.

Returns

the value set for the stability factor.

15.2.3.4 setBandwidthIndex()

set the index value for the bandwidth.

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

Parameters

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

15.2.3.5 setCompRangeName()

set a name for the compensation range for reference purposes.

Parameters

15.2.3.6 setStabilityFactor()

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

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

· AisCompRange.h

15.3 AisConstantCurrentElement Class Reference

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

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

Inherits AisAbstractElement.

Public Member Functions

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

the constant current element constructor.

AisConstantCurrentElement (const AisConstantCurrentElement &)

copy constructor for the AisConstantCurrentElement object.

AisConstantCurrentElement & operator= (const AisConstantCurrentElement &)

overload equal to operator for the AisConstantCurrentElement object.

• QString getName () const override

get the name of the element.

QStringList getCategory () const override

get a list of applicable categories of the element.

• double getCurrent () const

get the value set for the current.

52 Class Documentation

void setCurrent (double current)

set the value for the current.

double getSamplingInterval () const

get how frequently we are sampling the data.

void setSamplingInterval (double samplingInterval)

set how frequently we are sampling the data.

· double getMinSamplingVoltageDifference () const

get the minimum sampling voltage difference for reporting the data.

• void setMinSamplingVoltageDifference (double minVoltageDifference)

set a minimum sampling voltage difference for reporting the voltage.

• double getMaxVoltage () const

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

• void setMaxVoltage (double maxVoltage)

set a maximum voltage to stop the experiment.

• double getMinVoltage () const

get the value set minimum for the voltage in volts.

• void setMinVoltage (double minVoltage)

set a minimum voltage to stop the experiment.

double getMaxDuration () const

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

void setMaxDuration (double maxDuration)

set the maximum duration for the experiment.

double getMaxCapacity () const

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

void setMaxCapacity (double maxCapacity)

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

· bool isAutoRange () const

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

void setAutoRange ()

set to auto-select the current range.

double getApproxMaxCurrent () const

get the value set for the expected maximum current.

void setApproxMaxCurrent (double approxMaxCurrent)

set maximum current expected, for manual current range selection.

• bool isAutoVoltageRange () const

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

void setAutoVoltageRange ()

set to auto-select the voltage range.

double getApproxMaxVoltage () const

get the value set for the expected maximum voltage.

void setApproxMaxVoltage (double approxMaxVoltage)

set maximum voltage expected, for manual voltage range selection.

15.3.1 Detailed Description

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

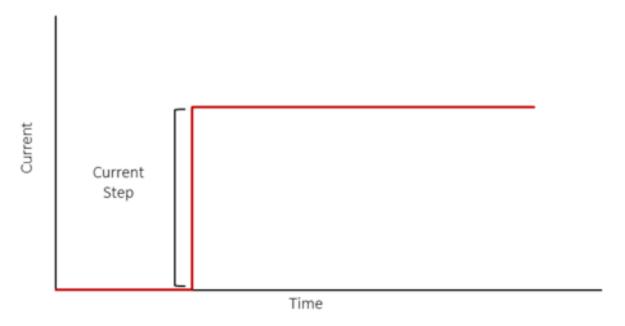


Figure 15.1 ConstantCurrent

15.3.2 Constructor & Destructor Documentation

15.3.2.1 AisConstantCurrentElement()

the constant current element constructor.

Parameters

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

15.3.3 Member Function Documentation

54 Class Documentation

15.3.3.1 getApproxMaxCurrent()

 $\verb|double AisConstantCurrentElement::getApproxMaxCurrent () const$

get the value set for the expected maximum current.

Returns

the value set for the expected maximum current in Amps.

Note

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

15.3.3.2 getApproxMaxVoltage()

double AisConstantCurrentElement::getApproxMaxVoltage () const

get the value set for the expected maximum voltage.

Returns

the value set for the expected maximum Voltage in volt.

Note

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

15.3.3.3 getCategory()

QStringList AisConstantCurrentElement::getCategory () const [override]

get a list of applicable categories of the element.

Returns

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

15.3.3.4 getCurrent()

 $\label{thm:double_AisConstantCurrentElement::getCurrent () const} \\ \\ \text{get the value set for the current.} \\$

Returns

the value for the current in Amps.

15.3.3.5 getMaxCapacity()

double AisConstantCurrentElement::getMaxCapacity () const

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

Returns

the value set for the maximum capacity in Coulomb.

Note

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

15.3.3.6 getMaxDuration()

double AisConstantCurrentElement::getMaxDuration () const

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

Returns

the maximum duration for the experiment in seconds.

15.3.3.7 getMaxVoltage()

 $\verb|double AisConstantCurrentElement::getMaxVoltage () const$

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

56 Class Documentation

15.3.3.8 getMinSamplingVoltageDifference()

double AisConstantCurrentElement::getMinSamplingVoltageDifference () const

get the minimum sampling voltage difference for reporting the data.

get the value set for the minimum sampling voltage difference.

Returns

the value set for the minimum sampling voltage difference.

See also

set Min Sampling Voltage Difference

Note

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

15.3.3.9 getMinVoltage()

double AisConstantCurrentElement::getMinVoltage () const

get the value set minimum for the voltage in volts.

Returns

the value set for the minimum voltage in volts.

Note

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

15.3.3.10 getName()

QString AisConstantCurrentElement::getName () const [override]

get the name of the element.

Returns

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

15.3.3.11 getSamplingInterval()

```
\verb|double AisConstantCurrentElement::getSamplingInterval () const|\\
```

get how frequently we are sampling the data.

Returns

the data sampling interval value in seconds.

15.3.3.12 isAutoRange()

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

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

Returns

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

15.3.3.13 isAutoVoltageRange()

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

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

Returns

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

15.3.3.14 setApproxMaxCurrent()

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

*MaxCurrent the value for the maximum current expected in Amps.

58 Class Documentation

15.3.3.15 setApproxMaxVoltage()

set maximum voltage expected, for manual voltage range selection.

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

Parameters

approxMaxVoltage	the value for the maximum current expected in V.
------------------	--

15.3.3.16 setAutoRange()

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

set to auto-select the current range.

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

15.3.3.17 setAutoVoltageRange()

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

set to auto-select the voltage range.

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

15.3.3.18 setCurrent()

```
void AisConstantCurrentElement::setCurrent ( \mbox{double } \mbox{\it current} \mbox{\ )}
```

set the value for the current.

Parameters

current	the value for the current in Amps.
current	the value for the current in Amps.

15.3.3.19 setMaxCapacity()

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

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

Parameters

maxCapacity the value to set for the	e cell maximum capacity.
--------------------------------------	--------------------------

15.3.3.20 setMaxDuration()

set the maximum duration for the experiment.

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

Parameters

maxDuration the maximum duration for the ex	periment in seconds.
---	----------------------

15.3.3.21 setMaxVoltage()

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.

may\/oltage	the maximum voltage value in volts at which the experiment will stop.
max voltage	the maximum voltage value in volts at which the experiment will stop.

15.3.3.22 setMinSamplingVoltageDifference()

```
\label{lement:setMinSamplingVoltageDifference} \mbox{ \begin{tabular}{ll} \label{lement:setMinSamplingVoltageDifference \end{tabular} } \mbox{ \end{tabular} } \mbox{ \begin{tabular}{ll} \label{lement:setMinSamplingVoltageDifference \end{tabular} } \mbox{ \end{tabular} } \mbox{ \begin{tabular}{ll} \label{lement:setMinSamplingVoltageDifference \end{tabular} } \mbox{ \end{tabular} } \mbox{ \begin{tabular}{ll} \label{lement:setMinSamplingVoltageDifference \end{tabular} } \mbox{ \begin{tabular}{ll} \label{lement:setMinSamplingVolt
```

set a minimum sampling voltage difference for reporting the voltage.

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

Note

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

Parameters

he minimum sampling voltage difference value in volts.
--

15.3.3.23 setMinVoltage()

set a minimum voltage to stop the experiment.

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

Parameters

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

15.3.3.24 setSamplingInterval()

set how frequently we are sampling the data.

samplingInterval	the data sampling interval value in seconds.
------------------	--

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

· AisConstantCurrentElement.h

15.4 AisConstantPotElement Class Reference

an experiment that simulates a constant applied voltage.

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

Inherits AisAbstractElement.

Public Member Functions

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

the constant potential element constructor.

AisConstantPotElement (const AisConstantPotElement &)

copy constructor for the AisConstantPotElement object.

AisConstantPotElement & operator= (const AisConstantPotElement &)

overload equal to operator for the AisConstantPotElement object.

• QString getName () const override

get the name of the element.

QStringList getCategory () const override

get a list of applicable categories of the element.

• double getPotential () const

get the value set for the potential in volts.

void setPotential (double potential)

set the value for the potential in volts.

• bool isVoltageVsOCP () const

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

void setVoltageVsOCP (bool vsOCP)

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

• double getSamplingInterval () const

get how frequently we are sampling the data.

void setSamplingInterval (double samplingInterval)

set how frequently we are sampling the data.

• double getMaxDuration () const

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

void setMaxDuration (double maxDuration)

set the maximum duration for the experiment.

• double getMaxCurrent () const

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

void setMaxCurrent (double maxCurrent)

set the maximum value for the absolute current in Amps.

double getMinCurrent () const

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

• void setMinCurrent (double minCurrent)

set the minimum value for the absolute current in Amps.

• double getMaxCapacity () const

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

void setMaxCapacity (double maxCapacity)

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

double getMindldt () const

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

• void setMindIdt (double mindIdt)

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

bool isAutoRange () const

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

• void setAutoRange ()

set to auto-select the current range.

double getApproxMaxCurrent () const

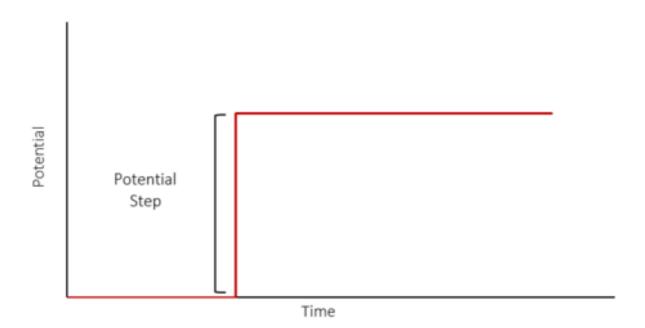
get the value set for the expected maximum current.

void setApproxMaxCurrent (double approxMaxCurrent)

set maximum current expected, for manual current range selection.

15.4.1 Detailed Description

an experiment that simulates a constant applied voltage.



15.4.2 Constructor & Destructor Documentation

15.4.2.1 AisConstantPotElement()

the constant potential element constructor.

Parameters

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

15.4.3 Member Function Documentation

15.4.3.1 getApproxMaxCurrent()

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

get the value set for the expected maximum current.

Returns

the value set for the expected maximum current in Amps.

Note

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

15.4.3.2 getCategory()

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

get a list of applicable categories of the element.

Returns

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

15.4.3.3 getMaxCapacity()

 $\verb|double AisConstantPotElement::getMaxCapacity () const|\\$

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

Returns

the value set for the maximum capacity in Coulomb.

Note

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

15.4.3.4 getMaxCurrent()

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

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

Returns

the maximum current value in Amps.

Note

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

15.4.3.5 getMaxDuration()

```
\verb|double AisConstantPotElement::getMaxDuration ( ) const|\\
```

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

Returns

the maximum duration for the experiment in seconds.

15.4.3.6 getMinCurrent()

```
\verb|double AisConstantPotElement::getMinCurrent () const|\\
```

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

Returns

the minimum current value in Amps.

Note

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

15.4.3.7 getMindldt()

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

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

Returns

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

Note

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

15.4.3.8 getName()

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

get the name of the element.

Returns

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

15.4.3.9 getPotential()

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

get the value set for the potential in volts.

Returns

the value set for the potential in volts.

15.4.3.10 getSamplingInterval()

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

get how frequently we are sampling the data.

Returns

the data sampling interval value in seconds.

15.4.3.11 isAutoRange()

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

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

Returns

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

15.4.3.12 isVoltageVsOCP()

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

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

Returns

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

See also

setVsOcp

15.4.3.13 setApproxMaxCurrent()

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

approximaxourrent the value for the maximum current expected in Amps.	approxMaxCurrent	the value for the maximum current expected in Amps.
---	------------------	---

15.4.3.14 setAutoRange()

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

set to auto-select the current range.

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

15.4.3.15 setMaxCapacity()

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

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

Parameters

	maxCapacity	the value to set for the cell maximum capacity.
--	-------------	---

15.4.3.16 setMaxCurrent()

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.

maxCurrent	the maximum current value in Amps.

15.4.3.17 setMaxDuration()

set the maximum duration for the experiment.

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

Parameters

maxDuration the maximum duration for the experiment in seconds	ds.
--	-----

15.4.3.18 setMinCurrent()

set the minimum value for the absolute current in Amps.

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

Parameters

```
minCurrent the value to set for the absolute minimum current.
```

15.4.3.19 setMindldt()

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.

mindldt the min	inimum value for the current rate o	of change with respect to time	(minimum di/dt).
-----------------	-------------------------------------	--------------------------------	------------------

15.4.3.20 setPotential()

set the value for the potential in volts.

Parameters

potential	the value to set for the potential in volts.
-----------	--

15.4.3.21 setSamplingInterval()

set how frequently we are sampling the data.

Parameters

samplingInterval	the data sampling interval value in seconds.
oagta.	tire data camping interval value in coconide.

15.4.3.22 setVoltageVsOCP()

```
void AisConstantPotElement::setVoltageVsOCP ( bool\ \textit{vsOCP}\ )
```

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

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

Parameters

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

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

· AisConstantPotElement.h

15.5 AisConstantPowerElement Class Reference

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

#include <AisConstantPowerElement.h>

Inherits AisAbstractElement.

Public Member Functions

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

the constant power element constructor

AisConstantPowerElement (const AisConstantPowerElement &)

copy constructor for the AisConstantPowerElement object.

AisConstantPowerElement & operator= (const AisConstantPowerElement &)

overload equal to operator for the AisConstantPowerElement object.

• QString getName () const override

get the name of the element.

· QStringList getCategory () const override

get a list of applicable categories of the element.

bool isCharge () const

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

void setCharge (bool isCharge)

set whether the experiment is to simulate charge or discharge.

• double getPower () const

get the value set for the power.

void setPower (double power)

set the value for the power.

• double getSamplingInterval () const

get how frequently we are sampling the data.

· void setSamplingInterval (double samplingInterval)

set how frequently we are sampling the data.

• double getMaxVoltage () const

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

void setMaxVoltage (double maxVoltage)

set a maximum voltage to stop the experiment.

• double getMinVoltage () const

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

• void setMinVoltage (double minVoltage)

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

• double getMaxDuration () const

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

• void setMaxDuration (double maxDuration)

set the maximum duration for the experiment.

• double getMaxCapacity () const

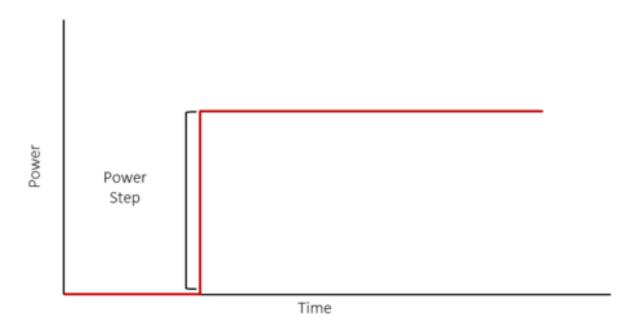
get the value set for the maximum capacity / cumulative charge.

void setMaxCapacity (double maxCapacity)

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

15.5.1 Detailed Description

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



15.5.2 Constructor & Destructor Documentation

15.5.2.1 AisConstantPowerElement()

the constant power element constructor

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

15.5.3 Member Function Documentation

15.5.3.1 getCategory()

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

get a list of applicable categories of the element.

Returns

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

15.5.3.2 getMaxCapacity()

double AisConstantPowerElement::getMaxCapacity () const

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

Returns

the value set for the maximum capacity in Coulomb.

Note

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

15.5.3.3 getMaxDuration()

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

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

Returns

the maximum duration for the experiment in seconds.

15.5.3.4 getMaxVoltage()

double AisConstantPowerElement::getMaxVoltage () const

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

Returns

the value set for the maximum voltage.

Note

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

15.5.3.5 getMinVoltage()

double AisConstantPowerElement::getMinVoltage () const

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

Returns

the minimum value set for the voltage in volts.

Note

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

15.5.3.6 getName()

QString AisConstantPowerElement::getName () const [override]

get the name of the element.

Returns

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

15.5.3.7 getPower()

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

get the value set for the power.

Returns

the value set for the power in watts.

15.5.3.8 getSamplingInterval()

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

get how frequently we are sampling the data.

Returns

the data sampling interval value in seconds.

15.5.3.9 isCharge()

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

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

Returns

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

15.5.3.10 setCharge()

```
\begin{tabular}{ll} \begin{tabular}{ll} void AisConstantPowerElement::setCharge ( \\ bool isCharge ) \end{tabular}
```

set whether the experiment is to simulate charge or discharge.

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

15.5.3.11 setMaxCapacity()

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

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

Parameters

	maxCapacity	the value to set for the cell maximum capacity.
--	-------------	---

15.5.3.12 setMaxDuration()

```
void AisConstantPowerElement::setMaxDuration ( \mbox{double } \textit{maxDuration} \ )
```

set the maximum duration for the experiment.

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

Parameters

maxDuration	the maximum duration for the experiment in seconds.
-------------	---

15.5.3.13 setMaxVoltage()

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.

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

15.5.3.14 setMinVoltage()

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

Parameters

minVoltage the value for the voltage in volts.	minVoltage	the value for the voltage in volts.
--	------------	-------------------------------------

Note

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

15.5.3.15 setPower()

set the value for the power.

Parameters

power the value set for the power in	watts.
--------------------------------------	--------

15.5.3.16 setSamplingInterval()

set how frequently we are sampling the data.

Parameters

samplingInterval the data sampling interval value	in seconds.

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

· AisConstantPowerElement.h

15.6 AisConstantResistanceElement Class Reference

This element/experiment simulates a constant resistance load.

#include <AisConstantResistanceElement.h>

Inherits AisAbstractElement.

Public Member Functions

AisConstantResistanceElement (double resistance, double duration, double samplingInterval)

the constant resistance element constructor.

• AisConstantResistanceElement (const AisConstantResistanceElement &)

copy constructor for the AisConstantResistanceElement object.

AisConstantResistanceElement & operator= (const AisConstantResistanceElement &)

overload equal to operator for the AisConstantResistanceElement object.

QString getName () const override

get the name of the element.

QStringList getCategory () const override

get a list of applicable categories of the element.

double getResistance () const

get the value set for the resistance as a load.

• void setResistance (double resistance)

set the value for the resistance as a load

• double getSamplingInterval () const

get how frequently we are sampling the data.

• void setSamplingInterval (double samplingInterval)

set how frequently we are sampling the data.

double getMinVoltage () const

get the value set minimum for the voltage in volts.

• void setMinVoltage (double minVoltage)

set a minimum voltage to stop the experiment.

• double getMaxDuration () const

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

void setMaxDuration (double maxDuration)

set the maximum duration for the experiment.

• double getMaxCapacity () const

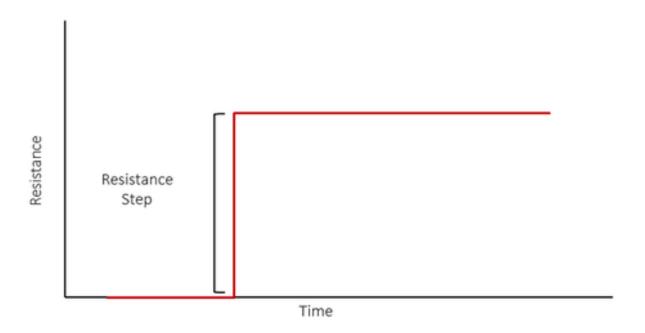
get the value set for the maximum capacity / cumulative charge.

void setMaxCapacity (double maxCapacity)

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

15.6.1 Detailed Description

This element/experiment simulates a constant resistance load.



15.6.2 Constructor & Destructor Documentation

15.6.2.1 AisConstantResistanceElement()

the constant resistance element constructor.

Parameters

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

15.6.3 Member Function Documentation

15.6.3.1 getCategory()

QStringList AisConstantResistanceElement::getCategory () const [override] get a list of applicable categories of the element.

Returns

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

15.6.3.2 getMaxCapacity()

double AisConstantResistanceElement::getMaxCapacity () const

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

Returns

the value set for the maximum capacity in Coulomb.

Note

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

15.6.3.3 getMaxDuration()

 $\verb|double AisConstantResistanceElement::getMaxDuration () const$

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

Returns

the maximum duration for the experiment in seconds.

15.6.3.4 getMinVoltage()

double AisConstantResistanceElement::getMinVoltage () const

get the value set minimum for the voltage in volts.

Returns

the value set for the minimum voltage in volts.

Note

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

15.6.3.5 getName()

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

get the name of the element.

Returns

The name of the element: "Constant Resistance".

15.6.3.6 getResistance()

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

get the value set for the resistance as a load.

Returns

the value in ohm of the load resistance.

15.6.3.7 getSamplingInterval()

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

get how frequently we are sampling the data.

Returns

the data sampling interval value in seconds.

15.6.3.8 setMaxCapacity()

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

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

Parameters

maxCapacity	the value to set for the cell maximum capacity.
-------------	---

15.6.3.9 setMaxDuration()

set the maximum duration for the experiment.

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

Parameters

maxDuration	the maximum duration for the experiment in seconds.

15.6.3.10 setMinVoltage()

```
void AisConstantResistanceElement::setMinVoltage ( \mbox{double } \textit{minVoltage} \ )
```

set a minimum voltage to stop the experiment.

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

Parameters

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

15.6.3.11 setResistance()

```
void AisConstantResistanceElement::setResistance ( \label{eq:constant} \mbox{double } resistance \ )
```

set the value for the resistance as a load

	Also control to along at the discolar statement
resistance	the value in ohm of the load resistance.

15.6.3.12 setSamplingInterval()

```
\label{local_problem} \mbox{void AisConstantResistanceElement::setSamplingInterval (} \\ \mbox{double } samplingInterval \mbox{)}
```

set how frequently we are sampling the data.

Parameters

samplingInterval	the data sampling interval value in seconds.
1- 3	9

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

· AisConstantResistanceElement.h

15.7 AisCyclicVoltammetryElement Class Reference

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

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

Inherits AisAbstractElement.

Public Member Functions

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

constructor of the cyclic voltammetry element.

AisCyclicVoltammetryElement (const AisCyclicVoltammetryElement &)

copy constructor for the AisCyclicVoltammetryElement object.

AisCyclicVoltammetryElement & operator= (const AisCyclicVoltammetryElement &)

overload equal to operator for the AisCyclicVoltammetryElement object.

· QString getName () const override

get the name of the element.

• QStringList getCategory () const override

get a list of applicable categories of the element.

• double getStartVoltage () const

get the value set for the start voltage

• void setStartVoltage (double startVoltage)

set the value for the start voltage.

• bool isStartVoltageVsOCP () const

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

void setStartVoltageVsOCP (bool startVoltageVsOCP)

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

• double getFirstVoltageLimit () const

get the value set for the first voltage-limit.

void setFirstVoltageLimit (double v1)

set the first voltage-limit

bool isFirstVoltageLimitVsOCP () const

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

void setFirstVoltageLimitVsOCP (bool firstVoltageLimitVsOCP)

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

• double getSecondVoltageLimit () const

get the value set for the second voltage-limit

void setSecondVoltageLimit (double v2)

set the second voltage-limit

bool isSecondVoltageLimitVsOCP () const

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

void setSecondVoltageLimitVsOCP (bool secondVoltageLimitVsOCP)

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

• double getNumberOfCycles ()

get the value set for the number of cycles

void setNumberOfCycles (int cycles)

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

• double getEndVoltage () const

get the value set for the ending potential value.

void setEndVoltage (double endVoltage)

set the ending potential value.

• bool isEndVoltageVsOCP () const

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

void setEndVoltageVsOCP (bool endVoltageVsOCP)

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

• double getdEdt () const

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

void setdEdt (double dEdt)

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

· double getSamplingInterval () const

get how frequently we are sampling the data.

void setSamplingInterval (double sampInterval)

set how frequently we are sampling the data.

• bool isAutoRange () const

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

• void setAutoRange ()

set to auto-select the current range.

• double getApproxMaxCurrent () const

get the value set for the expected maximum current.

void setApproxMaxCurrent (double approxMaxCurrent)

set maximum current expected, for manual current range selection.

15.7.1 Detailed Description

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

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

15.7.2 Constructor & Destructor Documentation

15.7.2.1 AisCyclicVoltammetryElement()

constructor of the cyclic voltammetry element.

Parameters

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

15.7.3 Member Function Documentation

15.7.3.1 getApproxMaxCurrent()

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

get the value set for the expected maximum current.

Returns

the value set for the expected maximum current in Amps.

Note

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

15.7.3.2 getCategory()

 ${\tt QStringList~AisCyclicVoltammetryElement::} {\tt getCategory~(~)~const~[override]}$

get a list of applicable categories of the element.

Returns

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

15.7.3.3 getdEdt()

double AisCyclicVoltammetryElement::getdEdt () const

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

Returns

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

15.7.3.4 getEndVoltage()

double AisCyclicVoltammetryElement::getEndVoltage () const

get the value set for the ending potential value.

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

Returns

the value set for the ending voltage in volts.

15.7.3.5 getFirstVoltageLimit()

double AisCyclicVoltammetryElement::getFirstVoltageLimit () const

get the value set for the first voltage-limit.

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

Returns

the first voltage-limit value in volts.

15.7.3.6 getName()

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

get the name of the element.

Returns

The name of the element: "Cyclic Voltammetry".

15.7.3.7 getNumberOfCycles()

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

get the value set for the number of cycles

Returns

the number of cycles set.

15.7.3.8 getSamplingInterval()

```
\verb|double AisCyclicVoltammetryElement::getSamplingInterval ( ) const
```

get how frequently we are sampling the data.

Returns

the data sampling interval value in seconds.

15.7.3.9 getSecondVoltageLimit()

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

get the value set for the second voltage-limit

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

Returns

the second voltage-limit value in volts.

15.7.3.10 getStartVoltage()

 $\verb|double AisCyclicVoltammetryElement::getStartVoltage () const$

get the value set for the start voltage

Returns

the value of the start voltage in volts

15.7.3.11 isAutoRange()

bool AisCyclicVoltammetryElement::isAutoRange () const

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

Returns

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

15.7.3.12 isEndVoltageVsOCP()

bool AisCyclicVoltammetryElement::isEndVoltageVsOCP () const

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

Returns

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

Note

if no value was set, the default is false

15.7.3.13 isFirstVoltageLimitVsOCP()

 $\verb|bool AisCyclicVoltammetryElement:: is First Voltage Limit Vs OCP () const$

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

Returns

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

Note

if no value was set, the default is false.

15.7.3.14 isSecondVoltageLimitVsOCP()

```
\verb|bool AisCyclicVoltammetryElement:: is SecondVoltageLimitVsOCP ( ) constitution of the property of the prop
```

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

Returns

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

Note

if no value was set, the default is false.

15.7.3.15 isStartVoltageVsOCP()

```
\verb|bool AisCyclicVoltammetryElement:: is StartVoltageVsOCP ( ) const
```

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

Returns

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

15.7.3.16 setApproxMaxCurrent()

set maximum current expected, for manual current range selection.

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

Parameters

approxMaxCurrent	the value for the maximum current expected in Amps.

15.7.3.17 setAutoRange()

```
\verb"void AisCyclicVoltammetryElement::setAutoRange ()\\
```

set to auto-select the current range.

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

15.7.3.18 setdEdt()

```
void AisCyclicVoltammetryElement::setdEdt ( \mbox{double } \mbox{$dEd$t })
```

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

Parameters

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

15.7.3.19 setEndVoltage()

```
void AisCyclicVoltammetryElement::setEndVoltage ( {\tt double} \ endVoltage \ )
```

set the ending potential value.

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

Parameters

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

15.7.3.20 setEndVoltageVsOCP()

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

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

Parameters

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

15.7.3.21 setFirstVoltageLimit()

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

set the first voltage-limit

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

Parameters

```
v1 first voltage-limit value in volts
```

15.7.3.22 setFirstVoltageLimitVsOCP()

```
\label{lem:voltageLimitVsOCP} \mbox{ void AisCyclicVoltammetryElement::setFirstVoltageLimitVsOCP (} \\ \mbox{ bool } firstVoltageLimitVsOCP )
```

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

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

Parameters

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

15.7.3.23 setNumberOfCycles()

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

Parameters

```
cycles the number of cycles to set
```

15.7.3.24 setSamplingInterval()

```
\label{lem:condition} \mbox{void AisCyclicVoltammetryElement::setSamplingInterval (} \\ \mbox{double } sampInterval \mbox{)}
```

set how frequently we are sampling the data.

Parameters

samplnterval the data sampling interval value in seconds.	
---	--

15.7.3.25 setSecondVoltageLimit()

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

set the second voltage-limit

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

Parameters

v2 the second voltage-limit value in volts

15.7.3.26 setSecondVoltageLimitVsOCP()

```
\label{lem:void} \begin{tabular}{ll} void A is Cyclic Voltammetry Element:: set Second Voltage Limit Vs OCP ( \\ bool second Voltage Limit Vs OCP ) \end{tabular}
```

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

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

Parameters

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

15.7.3.27 setStartVoltage()

set the value for the start voltage.

Parameters

startVoltage	the value of the start voltage in volts
--------------	---

15.7.3.28 setStartVoltageVsOCP()

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

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

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

Parameters

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

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

· AisCyclicVoltammetryElement.h

15.8 AisDCCurrentSweepElement Class Reference

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

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

Inherits AisAbstractElement.

Public Member Functions

AisDCCurrentSweepElement (double startCurrent, double endCurrent, double scanRate, double sampling
 —
 Interval)

the DC current sweep element.

AisDCCurrentSweepElement (const AisDCCurrentSweepElement &)

copy constructor for the AisDCCurrentSweepElement object.

AisDCCurrentSweepElement & operator= (const AisDCCurrentSweepElement &)

overload equal to operator for the AisDCCurrentSweepElement object.

· QString getName () const override

get the name of the element.

· QStringList getCategory () const override

get a list of applicable categories of the element.

double getStartingCurrent () const

get the value set for the starting current.

void setStartingCurrent (double startingCurrent)

set the value for the starting current.

• double getEndingCurrent () const

get the value set for the ending current.

void setEndingCurrent (double endingCurrent)

set the value for the ending current.

• double getScanRate () const

get the value set for the scan rate.

• void setScanRate (double scanRate)

set the value for the current scan rate.

• double getSamplingInterval () const

get how frequently we are sampling the data.

· void setSamplingInterval (double samplingInterval)

set how frequently we are sampling the data.

• double getMaxVoltage () const

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

• void setMaxVoltage (double maxVoltage)

set a maximum voltage to stop the experiment.

• double getMinVoltage () const

get the value set minimum for the voltage in volts.

• void setMinVoltage (double minVoltage)

set a minimum voltage to stop the experiment.

15.8.1 Detailed Description

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

DC Current Linear Sweep



15.8.2 Constructor & Destructor Documentation

15.8.2.1 AisDCCurrentSweepElement()

the DC current sweep element.

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

15.8.3 Member Function Documentation

15.8.3.1 getCategory()

```
QStringList AisDCCurrentSweepElement::getCategory ( ) const [override] get a list of applicable categories of the element.
```

Returns

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

get a list of applicable categories of the element.

Returns

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

15.8.3.2 getEndingCurrent()

```
double AisDCCurrentSweepElement::getEndingCurrent ( ) const
get the value set for the ending current.
```

Returns

the value for the ending current in Amps.

15.8.3.3 getMaxVoltage()

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

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

Returns

the value set for the maximum voltage.

Note

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

15.8.3.4 getMinVoltage()

```
\label{thm:constraint} \mbox{double AisDCCurrentSweepElement::getMinVoltage ( ) const}
```

get the value set minimum for the voltage in volts.

Returns

the value set for the minimum voltage in volts.

Note

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

15.8.3.5 getName()

```
QString AisDCCurrentSweepElement::getName ( ) const [override] get the name of the element.
```

Returns

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

15.8.3.6 getSamplingInterval()

```
double AisDCCurrentSweepElement::getSamplingInterval ( ) const
get how frequently we are sampling the data.
```

Returns

the data sampling interval value in seconds.

15.8.3.7 getScanRate()

```
\begin{tabular}{ll} \begin{tabular}{ll} double & AisDCCurrentSweepElement::getScanRate () const \\ \end{tabular} \label{table} get the value set for the scan rate.
```

Returns

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

See also

setScanRate

15.8.3.8 getStartingCurrent()

```
\verb|double AisDCCurrentSweepElement::getStartingCurrent ( ) const|\\
```

get the value set for the starting current.

Returns

the value set for the constant current in Amps.

15.8.3.9 setEndingCurrent()

set the value for the ending current.

Parameters

endingCurrent the value for the ending curr	ent in Amps
---	-------------

15.8.3.10 setMaxVoltage()

```
void AisDCCurrentSweepElement::setMaxVoltage ( \label{eq:double_maxVoltage} \ )
```

set a maximum voltage to stop the experiment.

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

Parameters

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

15.8.3.11 setMinVoltage()

```
void AisDCCurrentSweepElement::setMinVoltage ( \label{eq: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

15.8.3.12 setSamplingInterval()

set how frequently we are sampling the data.

Parameters

samplingInterval	the data sampling interval value in seconds.
------------------	--

15.8.3.13 setScanRate()

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

coanPato	the value to set for the scan rate.
Scarinale	i lile value lo sel loi lile scali fale.

15.8.3.14 setStartingCurrent()

```
\label{local_current} \mbox{ void AisDCCurrentSweepElement::setStartingCurrent (} \\ \mbox{ double } startingCurrent \mbox{ )}
```

set the value for the starting current.

Parameters

startingCurrent	the value to set for the starting current in Amps

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

· AisDCCurrentSweepElement.h

15.9 AisDCData Struct Reference

a structure containing DC data information.

#include <AisDataPoints.h>

Public Attributes

· double timestamp

the time at which the DC data arrived.

· double workingElectrodeVoltage

the measured working electrode voltage in volts.

· double counterElectrodeVoltage

the measured counter electrode voltage in volts.

· double current

the measured electric current value in Amps

double temperature

the measured temperature in Celsius.

15.9.1 Detailed Description

a structure containing DC data information.

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

· AisDataPoints.h

15.10 AisDCPotentialSweepElement Class Reference

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

#include <AisDCPotentialSweepElement.h>

Inherits AisAbstractElement.

Public Member Functions

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

the potential sweep element constructor.

AisDCPotentialSweepElement (const AisDCPotentialSweepElement &)

copy constructor for the AisDCPotentialSweepElement object.

• AisDCPotentialSweepElement & operator= (const AisDCPotentialSweepElement &)

overload equal to operator for the AisDCPotentialSweepElement object.

QString getName () const override

get the name of the element.

QStringList getCategory () const override

get a list of applicable categories of the element.

• double getStartingPot () const

get the value set for the starting potential.

void setStartingPot (double startingPotential)

set the value for the starting potential.

• bool isStartVoltageVsOCP () const

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

void setStartVoltageVsOCP (bool startVoltageVsOCP)

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

• double getEndingPot () const

get the value set for the ending potential value.

void setEndingPot (double endingPotential)

set the ending potential value.

• bool isEndVoltageVsOCP () const

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

void setEndVoltageVsOCP (bool endVoltageVsOCP)

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

• double getScanRate () const

get the value set for the voltage scan rate.

void setScanRate (double scanRate)

set the value for the voltage scan rate.

double getSamplingInterval () const

get how frequently we are sampling the data.

void setSamplingInterval (double samplingInterval)

set how frequently we are sampling the data.

bool isAutoRange () const

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

void setAutoRange ()

set to auto-select the current range.

double getApproxMaxCurrent () const

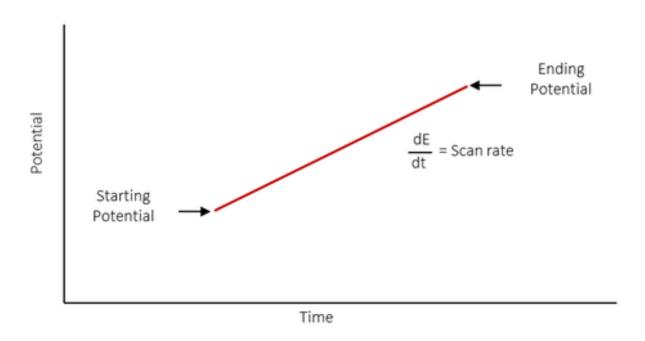
get the value set for the expected maximum current.

void setApproxMaxCurrent (double approxMaxCurrent)

set maximum current expected, for manual current range selection.

15.10.1 Detailed Description

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



15.10.2 Constructor & Destructor Documentation

15.10.2.1 AisDCPotentialSweepElement()

the potential sweep element constructor.

Parameters

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

15.10.3 Member Function Documentation

15.10.3.1 getApproxMaxCurrent()

double AisDCPotentialSweepElement::getApproxMaxCurrent () const

get the value set for the expected maximum current.

Returns

the value set for the expected maximum current in Amps.

Note

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

15.10.3.2 getCategory()

```
QStringList AisDCPotentialSweepElement::getCategory ( ) const [override] get a list of applicable categories of the element.
```

Returns

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

get a list of applicable categories of the element.

Returns

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

15.10.3.3 getEndingPot()

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

get the value set for the ending potential value.

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

Returns

the value set for the ending voltage in volts.

15.10.3.4 getName()

QString AisDCPotentialSweepElement::getName () const [override] get the name of the element.

Returns

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

15.10.3.5 getSamplingInterval()

double AisDCPotentialSweepElement::getSamplingInterval () const get how frequently we are sampling the data.

Returns

the data sampling interval value in seconds.

15.10.3.6 getScanRate()

double AisDCPotentialSweepElement::getScanRate () const
get the value set for the voltage scan rate.

Returns

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

See also

setScanRate

15.10.3.7 getStartingPot()

double AisDCPotentialSweepElement::getStartingPot () const
get the value set for the starting potential.

Returns

the value of the starting potential in volts.

15.10.3.8 isAutoRange()

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

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

Returns

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

15.10.3.9 isEndVoltageVsOCP()

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

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

Returns

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

See also

setEndVoltageVsOCP

15.10.3.10 isStartVoltageVsOCP()

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

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

Returns

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

See also

setStartVoltageVsOCP

15.10.3.11 setApproxMaxCurrent()

```
\label{thm:cond} \mbox{void AisDCPotentialSweepElement::setApproxMaxCurrent (} \\ \mbox{double } approxMaxCurrent \mbox{)}
```

set maximum current expected, for manual current range selection.

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

Parameters

approxMaxCurrent	the value for the maximum current expected in Amps.
------------------	---

15.10.3.12 setAutoRange()

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

set to auto-select the current range.

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

15.10.3.13 setEndingPot()

set the ending potential value.

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

Parameters

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

15.10.3.14 setEndVoltageVsOCP()

```
\label{local_problem} \begin{tabular}{ll} \b
```

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

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

Parameters

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

Note

by default, this is set to false.

15.10.3.15 setSamplingInterval()

```
\label{lem:condition} \mbox{ void AisDCPotentialSweepElement::setSamplingInterval (} \\ \mbox{ double } samplingInterval \mbox{ )}
```

set how frequently we are sampling the data.

Parameters

15.10.3.16 setScanRate()

set the value for the voltage scan rate.

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

Parameters

```
scanRate the value to set for the scan rate.
```

15.10.3.17 setStartingPot()

set the value for the starting potential.

Parameters

startingPotential	the value of the starting potential in volts
StartingPotential	the value of the starting potential in voits

15.10.3.18 setStartVoltageVsOCP()

```
\label{thm:cond} \mbox{void AisDCPotentialSweepElement::setStartVoltageVsOCP (} \\ \mbox{bool } startVoltageVsOCP \mbox{ )}
```

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

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

Parameters

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

Note

by default, this is set to false.

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

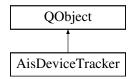
· AisDCPotentialSweepElement.h

15.11 AisDeviceTracker Class Reference

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

#include <AisDeviceTracker.h>

Inheritance diagram for AisDeviceTracker:



Signals

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

Public Member Functions

- AisErrorCode connectToDeviceOnComPort (const QString &comPort)
 - establish a connection with a device connected on a USB port.
- const AisInstrumentHandler & getInstrumentHandler (const QString &deviceName) const
 - get an instrument handler to control a specific device.
- const std::list< QString > getConnectedDevices () const
 - get a list of all the connected devices.
- int connectAllPluggedInDevices ()
 - connect all devices physically plugged to the computer.
- AisErrorCode updateFirmwareOnComPort (QString comport) const
 - update firmware on connected device at USB port.
- int updateFirmwareOnAllAvailableDevices ()
 - request firmware update for all available devices.

Static Public Member Functions

• static AisDeviceTracker * Instance ()

15.11.1 Detailed Description

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

15.11.2 Member Function Documentation

15.11.2.1 connectAllPluggedInDevices()

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

connect all devices physically plugged to the computer.

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

Returns

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

Note

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

15.11.2.2 connectToDeviceOnComPort()

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

Parameters

comPort the communication port to connect through.

Returns

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

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

Note

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

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

15.11.2.3 deviceDisconnected

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

Parameters

deviceName the name of the newly disconnected device.

15.11.2.4 getConnectedDevices()

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

get a list of all the connected devices.

Returns

a list of all the connected devices.

15.11.2.5 getInstrumentHandler()

get an instrument handler to control a specific device.

Parameters

Returns

the instrument handler that controls the specified device.

Note

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

See also

AisInstrumentHandler connectToDeviceOnComPort() getConnectedDevices()

15.11.2.6 newDeviceConnected

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

Parameters

Note

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

15.11.2.7 updateFirmwareOnAllAvailableDevices()

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

request firmware update for all available devices.

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

Returns

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

Note

emits firmwareUpdateNotification() signal will provide notification regarding firmware update of all devices. You can update firmware when you reset the device physically through reset button.

See also

updateFirmwareOnComPort

15.11.2.8 updateFirmwareOnComPort()

update firmware on connected device at USB port.

Parameters

comPort	the communication port to connect through.
---------	--

Returns

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

- · AisErrorCode::FirmwareUptodate
- AisErrorCode::ConnectionFailed

Note

emits firmware Update Notification() signal to provide firmware update progress.

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

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

· AisDeviceTracker.h

15.12 AisDiffPulseVoltammetryElement Class Reference

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

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

Inherits AisAbstractElement.

Public Member Functions

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

the differential pulse element constructor.

AisDiffPulseVoltammetryElement (const AisDiffPulseVoltammetryElement &)

copy constructor for the AisDiffPulseVoltammetryElement object.

AisDiffPulseVoltammetryElement & operator= (const AisDiffPulseVoltammetryElement &)

overload equal to operator for the AisDiffPulseVoltammetryElement object.

· QString getName () const override

get the name of the element.

QStringList getCategory () const override

get a list of applicable categories of the element.

double getStartVoltage () const

get the value set for the start voltage.

void setStartVoltage (double startVoltage)

set the value for the start voltage.

bool isStartVoltageVsOCP () const

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

void setStartVoltageVsOCP (bool startVoltageVsOCP)

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

• double getEndVoltage () const

get the value set for the ending potential value.

void setEndVoltage (double endVoltage)

set the ending potential value.

• bool isEndVoltageVsOCP () const

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

void setEndVoltageVsOCP (bool endVoltageVsOCP)

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

• double getVStep () const

get the value set for the potential step.

• void setVStep (double vStep)

set the value for the potential step.

double getPulseHeight () const

get the value set for the pulse height.

void setPulseHeight (double pulseHeight)

set the value for the pulse height.

• double getPulseWidth () const

get the value set for the pulse width.

void setPulseWidth (double pulseWidth)

set the value for the pulse width.

double getPulsePeriod () const

get the value set for the pulse period.

void setPulsePeriod (double pulsePeriod)

set the value for the pulse period.

• bool isAutoRange () const

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

void setAutoRange ()

set to auto-select the current range.

double getApproxMaxCurrent () const

get the value set for the expected maximum current.

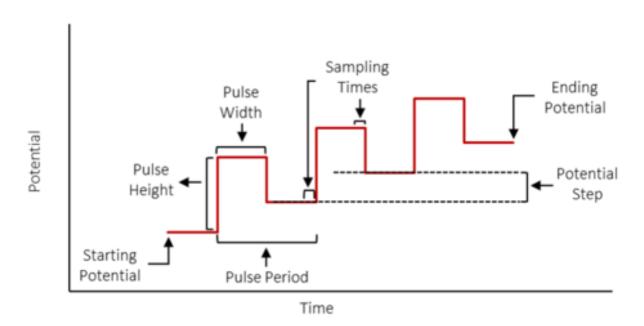
void setApproxMaxCurrent (double approxMaxCurrent)

set maximum current expected, for manual current range selection.

15.12.1 Detailed Description

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

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



15.12.2 Constructor & Destructor Documentation

15.12.2.1 AisDiffPulseVoltammetryElement()

the differential pulse element constructor.

Parameters

startVoltage	the value of the starting potential in volts
endVoltage	the value of the ending potential in volts
Generated & Admir	al therwatus ৬ধা for the voltage step in volts.
pulseHeight	the value for the pulse height in volts.
pulseWidth	the value for the pulse width in volts.
pulsePeriod	the value for the pulse period in volts.

15.12.3 Member Function Documentation

15.12.3.1 getApproxMaxCurrent()

 $\verb|double AisDiffPulseVoltammetryElement::getApproxMaxCurrent () const|\\$

get the value set for the expected maximum current.

Returns

the value set for the expected maximum current in Amps.

Note

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

15.12.3.2 getCategory()

```
QStringList AisDiffPulseVoltammetryElement::getCategory ( ) const [override] get a list of applicable categories of the element.
```

Returns

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

get a list of applicable categories of the element.

Returns

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

15.12.3.3 getEndVoltage()

```
\verb|double AisDiffPulseVoltammetryElement::getEndVoltage () const|\\
```

get the value set for the ending potential value.

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

Returns

the value set for the ending voltage in volts.

15.12.3.4 getName()

QString AisDiffPulseVoltammetryElement::getName () const [override]

get the name of the element.

Returns

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

15.12.3.5 getPulseHeight()

 $\label{thm:double_AisDiffPulseVoltammetryElement::getPulseHeight () const.}$

get the value set for the pulse height.

Returns

the value set for the pulse height in volts.

See also

setPulseHeight

15.12.3.6 getPulsePeriod()

double AisDiffPulseVoltammetryElement::getPulsePeriod () const

get the value set for the pulse period.

Returns

the value set for the pulse period in seconds.

See also

setPulsePeriod

15.12.3.7 getPulseWidth()

```
\label{thm:constraint} \mbox{double AisDiffPulseVoltammetryElement::} getPulseWidth \mbox{ ( ) const}
```

get the value set for the pulse width.

Returns

the value set for the pulse width in seconds.

See also

setPulseWidth

15.12.3.8 getStartVoltage()

```
\label{thm:const} \mbox{double AisDiffPulseVoltammetryElement::getStartVoltage ( ) const} \\ \mbox{get the value set for the start voltage}.
```

Returns

the value of the start voltage in volts.

15.12.3.9 getVStep()

```
\label{thm:const} \mbox{double AisDiffPulseVoltammetryElement::getVStep () const}
```

Returns

the value set for the potential step in volts.

get the value set for the potential step.

See also

setVStep

15.12.3.10 isAutoRange()

```
bool AisDiffPulseVoltammetryElement::isAutoRange ( ) const
```

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

Returns

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

15.12.3.11 isEndVoltageVsOCP()

```
bool AisDiffPulseVoltammetryElement::isEndVoltageVsOCP ( ) const
```

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

Returns

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

See also

setEndVoltageVsOCP

15.12.3.12 isStartVoltageVsOCP()

```
bool AisDiffPulseVoltammetryElement::isStartVoltageVsOCP ( ) const
```

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

Returns

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

See also

setStartVoltageVsOCP

15.12.3.13 setApproxMaxCurrent()

```
\label{lem:condition} \mbox{void AisDiffPulseVoltammetryElement::setApproxMaxCurrent (} \\ \mbox{double } approxMaxCurrent \mbox{)}
```

set maximum current expected, for manual current range selection.

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

Parameters

approxMaxCurrent	the value for the maximum current expected in Amps.
------------------	---

15.12.3.14 setAutoRange()

```
void AisDiffPulseVoltammetryElement::setAutoRange ( )
```

set to auto-select the current range.

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

15.12.3.15 setEndVoltage()

set the ending potential value.

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

Parameters

0.7.11	
endVoltage	the value to set for the ending voltage in volts.
ona ronago	ind talde to cot for the chang tollage in tolle.

15.12.3.16 setEndVoltageVsOCP()

```
void AisDiffPulseVoltammetryElement::setEndVoltageVsOCP ( bool\ endVoltageVsOCP\ )
```

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

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

Parameters

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

Note

by default, this is set to false.

15.12.3.17 setPulseHeight()

```
void AisDiffPulseVoltammetryElement::setPulseHeight ( \mbox{double } pulse\mbox{\it Height })
```

set the value for the pulse height.

For the first pulse, the pulse height is added to the starting potential. For the next pulse, the pulse height is added to the potential voltage and the potential step. In general, the pulse height is added to the potential step and the starting voltage of the last pulse.

Parameters

pulseHeight	the value to set for the pulse height in volts.
-------------	---

15.12.3.18 setPulsePeriod()

```
void AisDiffPulseVoltammetryElement::setPulsePeriod ( {\tt double}\ pulsePeriod\ )
```

set the value for the pulse period.

The pulse period is the time spent between the starts of two consecutive pulses.

Parameters

unido o Douis d	the control to the set for the more period in a consequent
puisererioa	the value to set for the pulse period in seconds.

15.12.3.19 setPulseWidth()

```
void AisDiffPulseVoltammetryElement::setPulseWidth ( \mbox{double } pulseWidth \ )
```

set the value for the pulse width.

The pulse width is the value in seconds for the time spent at the same voltage set for the pulse height.

Parameters

pulseWidth	the value to set for the pulse width in seconds.

See also

setPulseHeight

15.12.3.20 setStartVoltage()

```
\label{lem:condition} \mbox{void AisDiffPulseVoltammetryElement::setStartVoltage (} \\ \mbox{double } startVoltage \mbox{ )}
```

set the value for the start voltage.

Parameters

f the start voltage in volts	startVoltage
------------------------------	--------------

15.12.3.21 setStartVoltageVsOCP()

```
void AisDiffPulseVoltammetryElement::setStartVoltageVsOCP ( bool\ startVoltageVsOCP\ )
```

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

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

Parameters

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

Note

by default, this is set to false.

15.12.3.22 setVStep()

```
void AisDiffPulseVoltammetryElement::setVStep ( \mbox{double } vStep \mbox{ )}
```

set the value for the potential step.

The potential step is the difference between the starting potential of two consecutive pulses.

Parameters

vStep	the value to set for the potential step in volts.

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

· AisDiffPulseVoltammetryElement.h

15.13 AisEISGalvanostaticElement Class Reference

This experiment records the complex impedance of the experimental cell in galvanostatic mode, starting from the start frequency and sweeping through towards the end frequency, with a fixed number of frequency steps per decade.

#include <AisEISGalvanostaticElement.h>

Inherits AisAbstractElement.

Public Member Functions

 AisEISGalvanostaticElement (double startFrequency, double endFrequency, double stepsPerDecade, double currentBias, double currentAamplitude)

the EIS galvanostatic element constructor.

AisEISGalvanostaticElement (const AisEISGalvanostaticElement &)

copy constructor for the AisEISGalvanostaticElement object.

AisEISGalvanostaticElement & operator= (const AisEISGalvanostaticElement &)

overload equal to operator for the AisEISGalvanostaticElement object.

• QString getName () const override

get the name of the element.

• QStringList getCategory () const override

get a list of applicable categories of the element.

• double getStartFreq () const

get the value set for the current starting frequency

void setStartFreq (double startFreq)

set the value for the current starting frequency.

• double getEndFreq () const

the value set for the current ending frequency.

void setEndFreq (double endFreq)

set the value for the current end frequency.

• double getStepsPerDecade () const

get the value set for the current frequency steps per decade.

void setStepsPerDecade (double stepsPerDecade)

set the number of the current frequency steps per decade.

• double getBiasCurrent () const

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

void setBiasCurrent (double biasCurrent)

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

double getAmplitude () const

the value to set for the AC current amplitude.

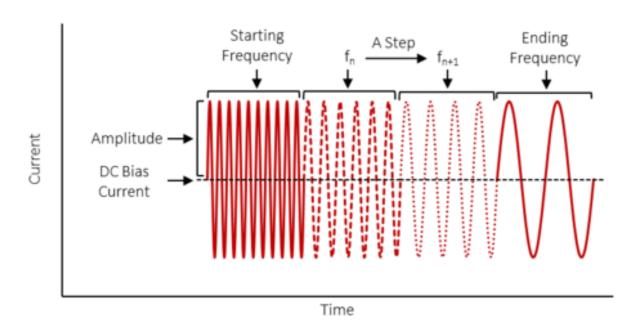
void setAmplitude (double amplitude)

set the value for the AC current amplitude.

15.13.1 Detailed Description

This experiment records the complex impedance of the experimental cell in galvanostatic mode, starting from the start frequency and sweeping through towards the end frequency, with a fixed number of frequency steps per decade.

Important parameters include the DC bias and the AC excitation amplitude.



15.13.2 Constructor & Destructor Documentation

15.13.2.1 AisEISGalvanostaticElement()

the EIS galvanostatic element constructor.

Parameters

startFrequency	the value for the current starting frequency
endFrequency	the value for the current ending frequency
stepsPerDecade	the value for the current frequency steps per decade.
currentBias	the value for the DC bias (DC offset).
currentAamplitude	the AC current amplitude.

15.13.3 Member Function Documentation

15.13.3.1 getAmplitude()

double AisEISGalvanostaticElement::getAmplitude () const

the value to set for the AC current amplitude.

Returns

the value set for the AC current amplitude in Amps.

15.13.3.2 getBiasCurrent()

 ${\tt double\ AisEISGalvanostaticElement::} {\tt getBiasCurrent\ (\)\ const}$

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

Returns

the value set for the DC bias in Amps.

15.13.3.3 getCategory()

QStringList AisEISGalvanostaticElement::getCategory () const [override]

get a list of applicable categories of the element.

Returns

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

get a list of applicable categories of the element.

Returns

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

15.13.3.4 getEndFreq()

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

the value set for the current ending frequency.

Returns

the value set for the current end frequency in Hz

15.13.3.5 getName()

```
QString AisEISGalvanostaticElement::getName ( ) const [override] get the name of the element.
```

Returns

The name of the element: "Galvanostatic EIS".

15.13.3.6 getStartFreq()

```
double AisEISGalvanostaticElement::getStartFreq ( ) const
get the value set for the current starting frequency
```

Returns

the value set for the current start frequency in Hz?

15.13.3.7 getStepsPerDecade()

```
double AisEISGalvanostaticElement::getStepsPerDecade ( ) const get the value set for the current frequency steps per decade.
```

Returns

the value set for the current frequency steps per decade. This is unit-less.

15.13.3.8 setAmplitude()

set the value for the AC current amplitude.

Parameters

amplitude the value to set for the AC current amplitude in Amps.

15.13.3.9 setBiasCurrent()

```
void AisEISGalvanostaticElement::setBiasCurrent ( {\tt double}\ biasCurrent\ )
```

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

Parameters

biasCurrent	the value to set for the DC bias in Amps.
-------------	---

15.13.3.10 setEndFreq()

```
void AisEISGalvanostaticElement::setEndFreq ( \mbox{double } \mbox{\it endFreq })
```

set the value for the current end frequency.

Parameters

endFreq the value to set for the current end frequency in Hz

15.13.3.11 setStartFreq()

set the value for the current starting frequency.

Parameters

startFreq the value to set the current starting frequency in Hz

15.13.3.12 setStepsPerDecade()

```
void AisEISGalvanostaticElement::setStepsPerDecade ( \mbox{double } stepsPerDecade \ )
```

set the number of the current frequency steps per decade.

Parameters

stepsPerDecade the value to set for the number of steps per decade.

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

AisEISGalvanostaticElement.h

15.14 AisEISPotentiostaticElement Class Reference

This experiment records the complex impedance of the experimental cell in potentiostatic mode, starting from the start frequency and sweeping through towards the end frequency, with a fixed number of frequency steps per decade.

```
#include <AisEISPotentiostaticElement.h>
```

Inherits AisAbstractElement.

Public Member Functions

 AisEISPotentiostaticElement (double startFrequency, double endFrequency, double stepsPerDecade, double voltageBias, double voltageAamplitude)

the EIS potentiostatic element

• AisEISPotentiostaticElement (const AisEISPotentiostaticElement &)

copy constructor for the AisEISPotentiostaticElement object.

• AisEISPotentiostaticElement & operator= (const AisEISPotentiostaticElement &)

overload equal to operator for the AisEISPotentiostaticElement object.

QString getName () const override

get the name of the element.

• QStringList getCategory () const override

get a list of applicable categories of the element.

• double getStartFreq () const

get the value set for the voltage starting frequency

void setStartFreq (double startFreq)

set the value for the voltage starting frequency.

• double getEndFreq () const

the value set for the voltage ending frequency.

void setEndFreq (double endFreq)

set the value for the voltage end frequency.

double getStepsPerDecade () const

get the value set for the voltage frequency steps per decade.

void setStepsPerDecade (double stepsPerDecade)

set the number of the voltage frequency steps per decade.

• double getBiasVoltage () const

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

• void setBiasVoltage (double biasVoltage)

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

• bool isBiasVoltageVsOCP () const

tells whether the DC-bias voltage is referenced against the open-circuit voltage or the reference cable.

void setBiasVoltageVsOCP (bool biasVsOCP)

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

• double getAmplitude () const

the value to set for the AC voltage amplitude.

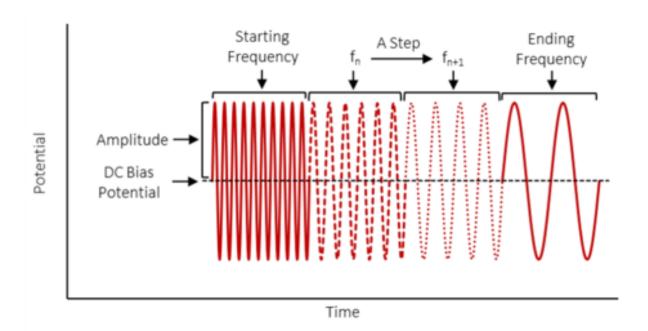
void setAmplitude (double amplitude)

set the value for the AC voltage amplitude.

15.14.1 Detailed Description

This experiment records the complex impedance of the experimental cell in potentiostatic mode, starting from the start frequency and sweeping through towards the end frequency, with a fixed number of frequency steps per decade.

Important parameters include the **DC bias** and the **AC excitation amplitude**.



15.14.2 Constructor & Destructor Documentation

15.14.2.1 AisEISPotentiostaticElement()

the EIS potentiostatic element

Parameters

startFrequency	the value for the voltage starting frequency
endFrequency	the value for the voltage ending frequency
stepsPerDecade	the value for the voltage frequency steps per decade.
voltageBias	the value for the DC bias (DC offset).
voltageAamplitude	the AC voltage amplitude.

15.14.3 Member Function Documentation

15.14.3.1 getAmplitude()

```
double AisEISPotentiostaticElement::getAmplitude ( ) const
```

the value to set for the AC voltage amplitude.

Returns

the value set for the AC voltage amplitude in volts.

15.14.3.2 getBiasVoltage()

```
\verb|double AisEISPotentiostaticElement::getBiasVoltage () const
```

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

Returns

the value set for the DC bias in volts.

get a list of applicable categories of the element.

15.14.3.3 getCategory()

QStringList AisEISPotentiostaticElement::getCategory () const [override]

Returns

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

15.14.3.4 getEndFreq()

double AisEISPotentiostaticElement::getEndFreq () const

the value set for the voltage ending frequency.

Returns

the value set for the voltage end frequency in Hz

15.14.3.5 getName()

QString AisEISPotentiostaticElement::getName () const [override]

get the name of the element.

Returns

The name of the element: "Potentiostatic EIS".

15.14.3.6 getStartFreq()

double AisEISPotentiostaticElement::getStartFreq () const

get the value set for the voltage starting frequency

Returns

the value set for the start frequency in Hz

15.14.3.7 getStepsPerDecade()

```
double AisEISPotentiostaticElement::getStepsPerDecade ( ) const
```

get the value set for the voltage frequency steps per decade.

Returns

the value set for the frequency steps per decade. This is unit-less.

15.14.3.8 isBiasVoltageVsOCP()

```
bool AisEISPotentiostaticElement::isBiasVoltageVsOCP ( ) const
```

tells whether the DC-bias voltage is referenced against the open-circuit voltage or the reference cable.

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

Returns

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

15.14.3.9 setAmplitude()

set the value for the AC voltage amplitude.

Parameters

ſ	amplitude	the value to set for the AC voltage amplitude in volts.
---	-----------	---

15.14.3.10 setBiasVoltage()

```
void AisEISPotentiostaticElement::setBiasVoltage ( {\tt double}\ biasVoltage\ )
```

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

Parameters

biasVoltage	the value to set for the DC bias in volts.
-------------	--

15.14.3.11 setBiasVoltageVsOCP()

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

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

Parameters

biasVsO0	P	true to if the DC-bias voltage is set to reference the open-circuit voltage and false if set against
		the reference terminal.

15.14.3.12 setEndFreq()

```
void AisEISPotentiostaticElement::setEndFreq ( \label{eq:condition} \texttt{double} \ \textit{endFreq} \ )
```

set the value for the voltage end frequency.

Parameters

endFrea	the value to set for the voltage end frequency in Hz

15.14.3.13 setStartFreq()

```
void AisEISPotentiostaticElement::setStartFreq ( \label{eq:condition} \texttt{double} \ \ \textit{startFreq} \ )
```

set the value for the voltage starting frequency.

otortEroa	the value to set the starting frequency Hz
siariFreq	I the value to set the starting frequency hz

15.14.3.14 setStepsPerDecade()

```
void AisEISPotentiostaticElement::setStepsPerDecade ( \mbox{double } stepsPerDecade \ )
```

set the number of the voltage frequency steps per decade.

Parameters

stepsPerDecade	the value to set for the number of steps per decade.
----------------	--

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

· AisEISPotentiostaticElement.h

15.15 AisErrorCode Class Reference

This class contains the possible error codes returned to the user when working with the API.

```
#include <AisErrorCode.h>
```

Public Types

```
    enum ErrorCode: uint8_t {
        Unknown = 255, Success = 0, ConnectionFailed = 1, FirmwareNotSupported = 2,
        FirmwareFileNotFound = 3, FirmwareUptodate = 4, InvalidChannel = 10, BusyChannel = 11,
        DeviceNotFound = 13, ManualExperimentNotRunning = 51, ExperimentNotUploaded = 52,
        ExperimentIsEmpty = 53,
        InvalidParameters = 54, ChannelNotBusy = 55, DeviceCommunicationFailed = 100, FailedToSetManualModeCurrentRange = 101,
        FailedToSetManualModeConstantVoltage = 102, FailedToPauseExperiment = 103, FailedToResumeExperiment = 104, FailedToStopExperiment = 105,
        FailedToUploadExperiment = 106, ExperimentAlreadyPaused = 107, ExperimentAlreadyRun = 108}
```

Public Member Functions

- AisErrorCode (ErrorCode error)
- AisErrorCode (ErrorCode error, QString message)
- QString message () const

a function to get a message explaining the error.

- int value () const
 - a function to get the error code.
- operator ErrorCode () const

15.15.1 Detailed Description

This class contains the possible error codes returned to the user when working with the API.

If a function has an AisErrorCode return type, then it needs to be checked for possible failures. The object of this class returned will contain an error code that can be accessed by calling value() member function and an error message that can be accessed by calling

See also

message.

15.15.2 Member Enumeration Documentation

15.15.2.1 ErrorCode

enum AisErrorCode::ErrorCode : uint8_t

Enumerator

Unknown	indicates that the command failed for an unknown reason.
Success	indicates success.
ConnectionFailed	indicates failure connecting the plugged in device when calling
	AisDeviceTracker::connectToDeviceOnComPort.
FirmwareNotSupported	indicates failure connecting the plugged in device when calling
	AisDeviceTracker::connectToDeviceOnComPort because firmware
	update require.
InvalidChannel	indicates that the given channel number is not valid.
BusyChannel	indicates that failure was due to the channel being busy.
DeviceNotFound	indicates that no device was detected to be connected.
ManualExperimentNotRunning	indicates that the given command applies when there is a manual
	experiment running on the channel but there is none.
ExperimentNotUploaded	indicates that the given command applies when an experiment
	has already been uploaded to the channel but there is none.
ExperimentIsEmpty	indicates that the given experiment has no elements. It need to
	contain at least one.
InvalidParameters	indicates that a given parameter is invalid. For example, it is out of
	the allowed range.
ChannelNotBusy	indicates that the given command applies when there is an
	experiment running or paused on the channel but there is none.
DeviceCommunicationFailed	indicates that there was failure in communication with the device.
FailedToSetManualModeCurrentRange	indicates failure of setting manual Mode current range for possible
	communication failure with the device.
FailedToSetManualModeConstantVoltage	indicates failure of setting manual Mode constant voltage for
	possible communication failure with the device.
FailedToPauseExperiment	indicates that pausing the experiment failed because either there
	is no running experiment or for possible communication failure
	with the device.

Enumerator

FailedToResumeExperiment	indicates that resuming the experiment failed because either there is no paused experiment or for possible communication failure with the device.
FailedToStopExperiment	indicates that stopping the experiment failed because either there is no running or paused experiment or for possible communication failure with the device.
FailedToUploadExperiment	indicates failure to communicate with the device to upload the experiment.
ExperimentAlreadyPaused	indicates that pausing the experiment failed because experiment is already pause.
ExperimentAlreadyRun	indicates that resuming the experiment failed because experiment is already running. it is not in pause state.

15.15.3 Member Function Documentation

15.15.3.1 message()

QString AisErrorCode::message () const

a function to get a message explaining the error.

Returns

a message that explains the error.

15.15.3.2 value()

int AisErrorCode::value () const

a function to get the error code.

Returns

the error code

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

· AisErrorCode.h

15.16 AisExperiment Class Reference

this class is used to create custom experiments. A custom experiment has a container of contains one or more elements. Once you create elements are set their parameters, you can add them to the container

```
#include <AisExperiment.h>
```

Public Member Functions

AisExperiment ()

this is the default constructor for the custom experiment.

AisExperiment (const AisExperiment &exp)

this is the copy constructor for the custom experiment.

void operator= (const AisExperiment &exp)

the assignment operator for the custom experiment.

QString getExperimentName () const

get the name of the custom experiment.

QString getDescription () const

get a brief description of the custom experiment.

• QStringList getCategory () const

get the category for the custom experiment.

void setExperimentName (QString name)

set a name for the custom experiment.

void setDescription (QString description)

set a description for the experiment.

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

append an element to the custom experiment.

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

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

Friends

· class AisInstrumentHandler

15.16.1 Detailed Description

this class is used to create custom experiments. A custom experiment has a container of contains one or more elements. Once you create elements are set their parameters, you can add them to the container

Note

we call the basic experiments -that are used to build more complex custom experiments- elements. In contexts where both elements and custom experiments are used, elements will be referred to as elements to make the digestion. In other contexts, elements may also be referred to as experiments as they may indeed be used as experiments.

15.16.2 Constructor & Destructor Documentation

15.16.2.1 AisExperiment()

```
AisExperiment::AisExperiment (  {\tt const~AisExperiment~\&~exp~)} \quad [{\tt explicit}]
```

this is the copy constructor for the custom experiment.

Parameters

exp	the custom experiment to copy from.

15.16.3 Member Function Documentation

15.16.3.1 appendElement()

append an element to the custom experiment.

Parameters

element	an elemental experiment to be appended to this/(the calling) custom experiment.
repeat	the number of times this element is to be repeated. This is an optional parameter and is defaulted to equal 1 when not set.

Returns

true if appending the element was successful and false otherwise.

Note

although an element is an experiment, in the context of custom experiments, it is referred to as an element to make a distinction between the two. In other contexts where such distinction is not needed, an element may still be referred to as an experiment.

15.16.3.2 appendSubExperiment()

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

subExp	a sub experiment to be appended to this/(the calling) custom experiment.
repeat	the number of times this sub experiment is to be repeated. This is an optional parameter and is
	defaulted to equal 1 when not set.

Returns

true if appending the sub experiment was successful and false otherwise.

15.16.3.3 getCategory()

```
QStringList AisExperiment::getCategory ( ) const
```

get the category for the custom experiment.

Returns

the category set for the custom experiment. If no category has been set, the default category returned is ("Custom").

15.16.3.4 getDescription()

```
QString AisExperiment::getDescription ( ) const
```

get a brief description of the custom experiment.

Returns

the description set for the custom experiment. If no description has been set, the default description returned is "Not Defined".

15.16.3.5 getExperimentName()

```
QString AisExperiment::getExperimentName ( ) const
```

get the name of the custom experiment.

Returns

the name set for the custom experiment. If no name has been set, the default name returned is "Custom Experiment"

15.16.3.6 operator=()

the assignment operator for the custom experiment.

Parameters

exp the custom experiment to copy from.

15.16.3.7 setDescription()

set a description for the experiment.

Parameters

description to be set for the custom experiment.

15.16.3.8 setExperimentName()

set a name for the custom experiment.

Parameters

name the name to be set for the custom experiment.

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

· AisExperiment.h

15.17 AisExperimentNode Struct Reference

a structure containing some information regarding the running element.

#include <AisDataPoints.h>

Public Attributes

QString stepName

This is the name of the current element running.

· int stepNumber

this number is the order of the element within the custom experiment.

• int substepNumber

this number is the order of the step within the element.

15.17.1 Detailed Description

a structure containing some information regarding the running element.

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

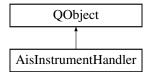
· AisDataPoints.h

15.18 AisInstrumentHandler Class Reference

this class provides control of the device including starting, pausing, resuming and stopping an experiment on a channel as well as reading the data and other controls of the device.

```
#include <AisInstrumentHandler.h>
```

Inheritance diagram for AisInstrumentHandler:



Signals

- void deviceDisconnected ()
 - a signal that is emitted if the device associated with this handler has been disconnected.
- void groundFloatStateChanged (bool grounded)
 - a signal that is emitted when the floating ground connection state has changed.
- void experimentNewElementStarting (uint8_t channel, const AisExperimentNode &stepInfo)
 - a signal that is emitted whenever a new elemental experiment has started.
- void activeDCDataReady (uint8_t channel, const AisDCData &DCData)
 - a signal that is emitted whenever new DC data for an active experiment are ready.
- void idleDCDataReady (uint8_t channel, const AisDCData &DCData)
 - a signal that is emitted whenever new DC data are ready when the device is in an idle state.
- void recoveryDCDataReady (uint8_t channel, const AisDCData &DCData)
 - a signal that is emitted whenever new DC recovery data are ready.
- void activeACDataReady (uint8_t channel, const AisACData &ACData)
 - a signal that is emitted whenever new AC data for an active experiment are ready.
- void recoveryACDataReady (uint8_t channel, const AisACData &ACData)
 - a signal that is emitted whenever new AC recovery data are ready.
- void experimentStopped (uint8_t channel)
 - a signal that is emitted whenever an experiment was stopped manually or has completed.
- void experimentPaused (uint8_t channel)
 - a signal that is emitted whenever an experiment was paused.
- void experimentResumed (uint8_t channel)
 - a signal that is emitted whenever an experiment was resumed.
- void recoverDataErased (bool successful)
 - a signal that is emitted whenever data erase process is completed.

Public Member Functions

AisErrorCode uploadExperimentToChannel (uint8_t channel, std::shared_ptr< AisExperiment > experiment)
const

upload an already created custom experiment to a specific channel on the device.

AisErrorCode uploadExperimentToChannel (uint8_t channel, const AisExperiment & experiment) const

upload an already created custom experiment to a specific channel on the device.

AisErrorCode startUploadedExperiment (uint8 t channel) const

start the previously uploaded experiment on the specific channel.

AisErrorCode skipExperimentStep (uint8 t channel) const

skip the current experiment step and proceed to the next.

AisErrorCode pauseExperiment (uint8 t channel) const

pause a running experiment on the channel.

AisErrorCode resumeExperiment (uint8_t channel) const

resume a paused experiment on the channel.

AisErrorCode stopExperiment (uint8_t channel) const

stop a running or a paused experiment on the channel.

double getExperimentUTCStartTime (uint8_t channel) const

get UTC time for the start of the experiment in seconds.

AisErrorCode setIRComp (uint8_t channel, double uncompensatedResistance, double compensationLevel)
const

set IR compensation.

• AisErrorCode setCompRange (uint8_t channel, const AisCompRange &compRange) const

set a compensation range with stability factor and bandwidth index.

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

connect several channels together in parallel mode.

int8_t setBipolarLinkedChannels (std::vector< uint8_t > channels) const

connect two channels together in bipolar mode.

bool hasBipolarMode (uint8_t channel) const

tells whether the given channel is bipolar mode

• std::vector< uint8_t > getLinkedChannels (uint8_t channel) const

get a list of channels linked to the given channel.

bool isChannelBusy (uint8_t channel) const

tells whether the given channel is busy or not.

bool isChannelPaused (uint8_t channel) const

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

std::vector< uint8_t > getFreeChannels () const

get a list of the currently free channels.

int getNumberOfChannels () const

get the number of all the channels on this device.

· AisErrorCode eraseRecoverData () const

delete the recover data from device.

AisErrorCode startManualExperiment (uint8_t channel) const

start a manual experiment.

AisErrorCode setManualModeSamplingInterval (uint8 t channel, double value) const

set an interval for sampling the data.

AisErrorCode setManualModeOCP (uint8_t channel) const

set open-circuit potential mode.

AisErrorCode setManualModeConstantVoltage (uint8 t channel, double value) const

set constant voltage for the manual experiment.

• AisErrorCode setManualModeConstantVoltage (uint8_t channel, double value, int currentRangeIndex) const

set constant voltage for the manual experiment and also set a manual current range.

- AisErrorCode setManualModeConstantCurrent (uint8_t channel, double value) const set constant current for the manual experiment.
- std::vector< std::pair< double, double >> getManualModeCurrentRangeList (uint8_t channel) const get a list of the applicable current ranges to the given channel specific to your device.

15.18.1 Detailed Description

this class provides control of the device including starting, pausing, resuming and stopping an experiment on a channel as well as reading the data and other controls of the device.

You may get an instrument handler instance of this class by calling AisDeviceTracker::getInstrumentHandler where you can get the device name either by calling AisDeviceTracker::getConnectedDevices or whenever the signal newDeviceConnected() is emitted.

15.18.2 Member Function Documentation

15.18.2.1 activeACDataReady

a signal that is emitted whenever new AC data for an active experiment are ready.

Parameters

channel	the channel number from which the AC data arrived.
ACData	the AC data that just arrived.

15.18.2.2 activeDCDataReady

a signal that is emitted whenever new DC data for an active experiment are ready.

channel	the channel number from which the DC data arrived.
DCData	the DC data that just arrived.

15.18.2.3 deviceDisconnected

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

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

15.18.2.4 eraseRecoverData()

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

delete the recover data from device.

Returns

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

- AisErrorCode::DeviceNotFound
- AisErrorCode::DeviceCommunicationFailed

15.18.2.5 experimentNewElementStarting

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

Parameters

channel	the channel number on which the experiment was started.
stepInfo	information regarding the current step.

See also

AisExperimentNode

15.18.2.6 experimentPaused

15.18 AisInstrumentHandler Class Reference 143 a signal that is emitted whenever an experiment was paused.

Parameters

channel	the channel on which the experiment was paused.
---------	---

15.18.2.7 experimentResumed

a signal that is emitted whenever an experiment was resumed.

Parameters

channel	the channel on which the experiment was resumed.
---------	--

15.18.2.8 experimentStopped

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

Parameters

channel	the channel on which the experiment has stopped.
---------	--

15.18.2.9 getExperimentUTCStartTime()

```
double AisInstrumentHandler::getExperimentUTCStartTime ( uint8\_t \ channel \ ) \ const
```

get UTC time for the start of the experiment in seconds.

This will give the time in seconds between the origin of UTC time and the start of the experiment aka Unix Epoch.

Returns

the Unix Epoch up to the start of the experiment in seconds.

15.18.2.10 getFreeChannels()

```
std::vector < uint8_t > AisInstrumentHandler::getFreeChannels ( ) const
```

get a list of the currently free channels.

Returns

a list of the currently free channels. If all channels are busy, an empty list is returned.

15.18.2.11 getLinkedChannels()

get a list of channels linked to the given channel.

Parameters

channel	a valid channel number to find which other channels are linked to it.
Criarinici	a valid chariner hamber to find which other charines are linked to it.

Returns

a list of channels linked to the channel parameter.

15.18.2.12 getManualModeCurrentRangeList()

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.

Returns

a list of the of the applicable current ranges to the given channel specific to your device.

15.18.2.13 getNumberOfChannels()

```
int AisInstrumentHandler::getNumberOfChannels ( ) const
```

get the number of all the channels on this device.

Returns

the number of channels on the connected device. If no device found, -1 will be returned.

15.18.2.14 groundFloatStateChanged

```
\begin{tabular}{ll} void AisInstrumentHandler:: groundFloatStateChanged ( \\ bool $grounded$ ) [signal] \end{tabular}
```

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

Parameters

grounded true if there is a connection to ground and false if the ground has disconnected.

15.18.2.15 hasBipolarMode()

tells whether the given channel is bipolar mode

Parameters

channel	the channel number to check if it is bipolar mode
---------	---

Returns

true only if given a valid channel number that has bipolar mode.

15.18.2.16 idleDCDataReady

a signal that is emitted whenever new DC data are ready when the device is in an idle state.

A manual experiment displays real time values. These values are displayed even if the channel does not have an experiment running on it.

Parameters

channel	the channel number from which the DC data arrived.
DCData	the DC data that just arrived.

15.18.2.17 isChannelBusy()

tells whether the given channel is busy or not.

Parameters

	channel	the channel number to check if it is busy or not.
--	---------	---

Returns

true only if given a valid channel number that has either a running or a paused experiment.

15.18.2.18 isChannelPaused()

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

channel	the channel number to check if it has a paused experiment.
---------	--

Returns

true only if given a valid channel number that has an experiment that has been paused.

15.18.2.19 pauseExperiment()

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

pause a running experiment on the channel.

Parameters

	channel	the channel number to pause the experiment on.
--	---------	--

Returns

true if an experiment was successfully paused on the channel and false otherwise. If not successful, possible returned errors are:

- AisErrorCode::FailedToPauseExperiment
- AisErrorCode::DeviceNotFound
- AisErrorCode::InvalidChannel
- · AisErrorCode::ChannelNotBusy

This will return AisErrorCode::Success only if there is currently a running experiment on a valid channel on a connected device.

15.18.2.20 recoverDataErased

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

Parameters

```
successful is true on erased correctly, and false on data is not erased.
```

15.18.2.21 recoveryACDataReady

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

Parameters

channel	the channel number from which the AC data are recovered from.
ACData	the AC data that just arrived.

15.18.2.22 recoveryDCDataReady

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

Parameters

channel	the channel number from which the DC data are recovered from.
DCData	the DC data that just arrived.

15.18.2.23 resumeExperiment()

resume a paused experiment on the channel.

Parameters

channel	the channel number to resume the experiment on.
---------	---

Returns

AisErrorCode::Success if an experiment was successfully resumed on the channel. If not successful, possible returned errors are:

- · AisErrorCode::FailedToResumeExperiment
- · AisErrorCode::DeviceNotFound
- AisErrorCode::InvalidChannel
- · AisErrorCode::ChannelNotBusy

This will return AisErrorCode::Success only if there is currently a paused experiment on a valid channel on a connected device.

15.18.2.24 setBipolarLinkedChannels()

```
int8_t AisInstrumentHandler::setBipolarLinkedChannels ( std::vector < \ uint8\_t \ > \ channels \ ) \ const
```

connect two channels together in bipolar mode.

You may combine two channels to expand the voltage range to include negative voltages. Note that this is only applicable to the cycler model. For 4 channel Cycler models, you can combine channels 1 and 2 or channels 3 and 4. You cannot use any other channel combinations.

Parameters

channels a list of two channels to be oprate in bipolar mode.	channels	a list of two channels to be oprate in bipolar mode.
---	----------	--

Returns

the master channel out of the given list of two channels. The master channel is your interface to upload an experiment to and then control it. If not successful set in bipolar mode, possible returned errors as -1.

Note

this functionality is only applicable to the cycler model.

15.18.2.25 setCompRange()

set a compensation range with stability factor and bandwidth index.

Parameters

channel	the channel for which to set the compensation range.
compRange	an object of type compRange that is initialized with a stability factor (0-10) and a bandwidth index (0-10).

Returns

AisErrorCode::Success if setting the IR compensation was successful. If not successful, possible returned errors are:

- AisErrorCode::DeviceNotFound
- AisErrorCode::InvalidChannel
- · AisErrorCode::InvalidParameters

See also

AisCompRange

15.18.2.26 setIRComp()

set IR compensation.

Parameters

channel	the channel for which to set the IR compensation.
uncompensatedResistance	the value of the uncompensated resistance in Ohms.
compensationLevel	the compensation percentage (0%-100%). This is unit-less.

Returns

AisErrorCode::Success if setting the IR compensation was successful. If not successful, possible returned errors are:

• AisErrorCode::DeviceNotFound

• AisErrorCode::InvalidChannel

· AisErrorCode::InvalidParameters

15.18.2.27 setLinkedChannels()

connect several channels together in parallel mode.

You may connect a list of channels so you can get a higher combined output current of all channels. Note that this is only applicable to the cycler model.

Parameters

```
channels a list of channels to be linked.
```

Returns

the master channel out of the given list of channels. The master channel is your interface to upload an experiment to and then control it.

Note

this functionality is only applicable to the cycler model.

15.18.2.28 setManualModeConstantCurrent()

set constant current for the manual experiment.

Parameters

channel	a valid channel number to set a constant voltage for.
value	the value to set the constant current in Amps.

Returns

AisErrorCode::Success if setting the constant current was successful. If not successful, possible returned errors are:

- AisErrorCode::ManualExperimentNotRunning
- AisErrorCode::DeviceNotFound
- AisErrorCode::InvalidChannel
- · AisErrorCode::DeviceCommunicationFailed

15.18.2.29 setManualModeConstantVoltage() [1/2]

set constant voltage for the manual experiment.

Parameters

channel	a valid channel number to set a constant voltage for.
value	the value to set the constant voltage in volts.

Returns

AisErrorCode::Success if setting the constant voltage was successful. If not successful, possible returned errors are:

• AisErrorCode::FailedToSetManualModeConstantVoltage

· AisErrorCode::ManualExperimentNotRunning

• AisErrorCode::DeviceNotFound

AisErrorCode::InvalidChannel

15.18.2.30 setManualModeConstantVoltage() [2/2]

set constant voltage for the manual experiment and also set a manual current range.

Parameters

channel	a valid channel number to set a constant voltage for.
value	the value to set the constant voltage in volts.
currentRangeIndex	the index of the desired current range.

Returns

AisErrorCode::Success if setting the constant voltage was successful. You can get a list of the available ranges for your model by calling getManualModeCurrentRangeList. If not successful, possible returned errors are:

- AisErrorCode::FailedToSetManualModeConstantVoltage
- AisErrorCode::FailedToSetManualModeCurrentRange
- · AisErrorCode::ManualExperimentNotRunning
- AisErrorCode::DeviceNotFound
- AisErrorCode::InvalidChannel

15.18.2.31 setManualModeOCP()

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()

channel	a valid channel number to set open circuit mode on.

Returns

AisErrorCode::Success if turning on the open circuit mode was successful. If not successful, possible returned errors are:

• AisErrorCode::ManualExperimentNotRunning

• AisErrorCode::DeviceNotFound

• AisErrorCode::InvalidChannel

• AisErrorCode::DeviceCommunicationFailed

15.18.2.32 setManualModeSamplingInterval()

set an interval for sampling the data.

Parameters

channel	the channel to set the sampling interval for.
value	the value for the sampling interval in seconds.

Returns

AisErrorCode::Success if the operation was set successfully. If not successful, possible returned errors are:

· AisErrorCode::DeviceNotFound

• AisErrorCode::Unknown

• AisErrorCode::InvalidChannel

15.18.2.33 skipExperimentStep()

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.

channel	a valid channel number with an experiment to skip the step.

Returns

AisErrorCode::Success the experiment step was successfully skipped If not successful, possible returned errors are:

- AisErrorCode::DeviceNotFound
- AisErrorCode::InvalidChannel
- · AisErrorCode::ChannelNotBusy
- AisErrorCode::DeviceCommunicationFailed

15.18.2.34 startManualExperiment()

start a manual experiment.

With manual experiments, users can turn on any connected channel and toggle between open circuit mode and voltage or current setpoints that can be changed in real-time and run for indefinite periods.

Parameters

channel	a valid channel number to run the manual experiment on.
---------	---

Returns

AisErrorCode::Success if the manual experiment was successfully started. If not successful, possible returned errors are:

- AisErrorCode::FailedManualModeStartExperiment
- AisErrorCode::DeviceNotFound
- AisErrorCode::InvalidChannel
- AisErrorCode::BusyChannel

15.18.2.35 startUploadedExperiment()

start the previously uploaded experiment on the specific channel.

channel	the channel number to start the experiment on.

Returns

AisErrorCode::Success if the experiment was successfully started on the channel. If not successful, possible returned errors are:

- · AisErrorCode::DeviceCommunicationFailed
- · AisErrorCode::ExperimentNotUploaded
- · AisErrorCode::DeviceNotFound
- AisErrorCode::InvalidChannel
- AisErrorCode::BusyChannel

See also

```
uploadExperimentToChannel isChannelBusy
```

15.18.2.36 stopExperiment()

stop a running or a paused experiment on the channel.

Parameters

channel	the channel number to stop the experiment on.

Returns

AisErrorCode::Success if an experiment was successfully stopped on the channel. If not successful, possible returned errors are:

- · AisErrorCode::FailedToStopExperiment
- · AisErrorCode::DeviceNotFound
- AisErrorCode::InvalidChannel

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

15.18.2.37 uploadExperimentToChannel() [1/2]

upload an already created custom experiment to a specific channel on the device.

Any running experiment is run on a specific device on a specific channel. This function uploads an experiment to a channel so that you may start, pause, resume and stop the experiment. All of these four control functionalities and others require a channel number to control the experiment. Therefore, if we have several channels, we need to keep track of which experiment is on which channel.

Parameters

channel	the channel number to upload the experiment to.
experiment	the custom experiment to be uploaded to the channel.

Returns

AisErrorCode::Success if the experiment was successfully uploaded to the channel. If not successful, possible returned errors are:

- AisErrorCode::FailedToUploadExperiment
- · AisErrorCode::ExperimentIsEmpty
- AisErrorCode::DeviceNotFound
- AisErrorCode::InvalidChannel
- AisErrorCode::BusyChannel
- AisErrorCode::InvalidParameters

This returns AisErrorCode::Success only when given a valid channel number that is not busy on a connected device.

See also

isChannelBusy

15.18.2.38 uploadExperimentToChannel() [2/2]

upload an already created custom experiment to a specific channel on the device.

Any running experiment is run on a specific device on a specific channel. This function uploads an experiment to a channel so that you may start, pause, resume and stop the experiment. All of these four control functionalities and others require a channel number to control the experiment. Therefore, if we have several channels, we need to keep track of which experiment is on which channel.

Parameters

channel	the channel number to upload the experiment to.
experiment	the custom experiment to be uploaded to the channel.

Returns

AisErrorCode::Success if the experiment was successfully uploaded to the channel. If not successful, possible returned errors are:

- AisErrorCode::FailedToUploadExperiment
- · AisErrorCode::ExperimentIsEmpty

- · AisErrorCode::DeviceNotFound
- AisErrorCode::InvalidChannel
- AisErrorCode::BusyChannel
- · AisErrorCode::InvalidParameters

This returns AisErrorCode::Success only when given a valid channel number that is not busy on a connected device.

See also

isChannelBusy

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

· AisInstrumentHandler.h

15.19 AisNormalPulseVoltammetryElement Class Reference

This experiment holds the working electrode at a **baseline potential** during the **quiet time**, then applies a train of pulses, which increase in amplitude until the **final potential** is reached.

#include <AisNormalPulseVoltammetryElement.h>

Inherits AisAbstractElement.

Public Member Functions

AisNormalPulseVoltammetryElement (double startVoltage, double endVoltage, double voltageStep, double pulseWidth, double pulsePeriod)

the normal-pulse-voltammetry element constructor

• AisNormalPulseVoltammetryElement (const AisNormalPulseVoltammetryElement &)

copy constructor for the AisNormalPulseVoltammetryElement object.

AisNormalPulseVoltammetryElement & operator= (const AisNormalPulseVoltammetryElement &)

overload equal to operator for the AisNormalPulseVoltammetryElement object.

• QString getName () const override

get the name of the element.

QStringList getCategory () const override

get a list of applicable categories of the element.

· double getStartVoltage () const

get the value set for the start voltage.

void setStartVoltage (double startVoltage)

set the value for the start voltage.

bool isStartVoltageVsOCP () const

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

void setStartVoltageVsOCP (bool startVoltageVsOCP)

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

• double getEndVoltage () const

get the value set for the ending potential value.

void setEndVoltage (double endVoltage)

set the ending potential value.

• bool isEndVoltageVsOCP () const

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

void setEndVoltageVsOCP (bool endVoltageVsOcp)

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

• double getVStep () const

get the value set for the voltage step.

void setVStep (double vStep)

set the value for the voltage step.

• double getPulseWidth () const

get the value for the voltage pulse width.

void setPulseWidth (double pulseWidth)

set the value for the voltage pulse width.

· double getPulsePeriod () const

get the value set for the pulse period.

void setPulsePeriod (double pulsePeriod)

set the value for the pulse period.

• bool isAutoRange () const

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

void setAutoRange ()

set to auto-select the current range.

double getApproxMaxCurrent () const

get the value set for the expected maximum current.

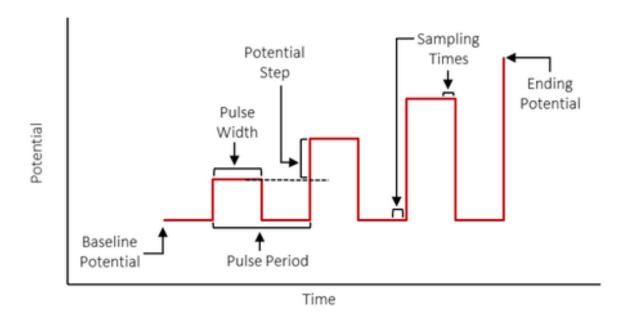
void setApproxMaxCurrent (double approxMaxCurrent)

set maximum current expected, for manual current range selection.

15.19.1 Detailed Description

This experiment holds the working electrode at a **baseline potential** during the **quiet time**, then applies a train of pulses, which increase in amplitude until the **final potential** is reached.

The **potential step** is the magnitude of this incremental increase. The **pulse width** is the amount of time between the rising and falling edge of a pulse. The **pulse period** is the amount of time between the beginning of one pulse and the beginning of the next.



15.19.2 Constructor & Destructor Documentation

15.19.2.1 AisNormalPulseVoltammetryElement()

the normal-pulse-voltammetry element constructor

Parameters

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

15.19.3 Member Function Documentation

15.19.3.1 getApproxMaxCurrent()

 $\label{thm:constraint} \mbox{double AisNormalPulseVoltammetryElement::getApproxMaxCurrent () const} \\ \mbox{get the value set for the expected maximum current.} \\$

Returns

the value set for the expected maximum current in Amps.

Note

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

15.19.3.2 getCategory()

QStringList AisNormalPulseVoltammetryElement::getCategory () const [override] get a list of applicable categories of the element.

Returns

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

15.19.3.3 getEndVoltage()

 $\verb|double AisNormalPulseVoltammetryElement::getEndVoltage () const$

get the value set for the ending potential value.

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

Returns

the value set for the ending voltage in volts.

15.19.3.4 getName()

QString AisNormalPulseVoltammetryElement::getName () const [override] get the name of the element.

Returns

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

15.19.3.5 getPulsePeriod()

```
double AisNormalPulseVoltammetryElement::getPulsePeriod ( ) const
get the value set for the pulse period.
```

Returns

the value set for the pulse period in seconds.

See also

setPulsePeriod

15.19.3.6 getPulseWidth()

```
\label{thm:const} \mbox{double AisNormalPulseVoltammetryElement::} \mbox{getPulseWidth () const} \\ \mbox{get the value for the voltage pulse width.}
```

Returns

the value for the voltage pulse width in seconds.

See also

setPulseWidth

15.19.3.7 getStartVoltage()

```
\label{thm:const} \mbox{double AisNormalPulseVoltammetryElement::getStartVoltage ( ) const} \\ \mbox{get the value set for the start voltage}.
```

Returns

the value of the start voltage in volts.

15.19.3.8 getVStep()

double AisNormalPulseVoltammetryElement::getVStep () const

get the value set for the voltage step.

Returns

the value set for the voltage step in volts.

See also

setVStep

15.19.3.9 isAutoRange()

bool AisNormalPulseVoltammetryElement::isAutoRange () const

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

Returns

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

15.19.3.10 isEndVoltageVsOCP()

bool AisNormalPulseVoltammetryElement::isEndVoltageVsOCP () const

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

Returns

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

Note

if nothing is set, the default is false.

15.19.3.11 isStartVoltageVsOCP()

```
bool AisNormalPulseVoltammetryElement::isStartVoltageVsOCP ( ) const
```

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

Returns

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

Note

if nothing is set, the default is false.

See also

setStartVoltageVsOCP

15.19.3.12 setApproxMaxCurrent()

```
\label{lem:condition} \mbox{void AisNormalPulseVoltammetryElement::setApproxMaxCurrent (} \\ \mbox{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

annray May Current	the value for the maximum current expected in Amne
approxiviaxCurrent	the value for the maximum current expected in Amps.

15.19.3.13 setAutoRange()

```
void AisNormalPulseVoltammetryElement::setAutoRange ( )
```

set to auto-select the current range.

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

15.19.3.14 setEndVoltage()

set the ending potential value.

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

Parameters

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

15.19.3.15 setEndVoltageVsOCP()

```
void AisNormalPulseVoltammetryElement::setEndVoltageVsOCP ( bool\ endVoltageVsOcp\ )
```

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

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

Parameters

endVoltageVsOcp	true to set the end voltage to be referenced against the open-circuit voltage and false if set	1
	against the reference terminal.	

Note

by default, this is set to false.

15.19.3.16 setPulsePeriod()

```
void AisNormalPulseVoltammetryElement::setPulsePeriod ( {\tt double} \ pulsePeriod \ )
```

set the value for the pulse period.

The pulse period is the time spent between the starts of two consecutive pulses.

Parameters

pulsePeriod the value to set for the pulse period in seconds.

15.19.3.17 setPulseWidth()

```
\label{lem:condition} \mbox{void AisNormalPulseVoltammetryElement::setPulseWidth (} \\ \mbox{double } pulseWidth \mbox{)}
```

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.
--

15.19.3.18 setStartVoltage()

```
void AisNormalPulseVoltammetryElement::setStartVoltage ( {\tt double} \  \, startVoltage \ )
```

set the value for the start voltage.

Parameters

startVoltage	the value of the start voltage in volts
--------------	---

15.19.3.19 setStartVoltageVsOCP()

```
\label{lem:condition} \mbox{void AisNormalPulseVoltammetryElement::setStartVoltageVsOCP (} \\ \mbox{bool } startVoltageVsOCP \mbox{ )}
```

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

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

Parameters

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

Note

by default, this is set to false.

15.19.3.20 setVStep()

```
void AisNormalPulseVoltammetryElement::setVStep ( \mbox{double } vStep \mbox{ )} \label{eq:vStep}
```

set the value for the voltage step.

The voltage step is the voltage difference between the heights of two consecutive pulses.

Parameters

vStep	the value for the voltage step in volts.
· Ciop	ino raido ioi ino rollago olop in rollo.

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

· AisNormalPulseVoltammetryElement.h

15.20 AisOpenCircuitElement Class Reference

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

```
#include <AisOpenCircuitElement.h>
```

Inherits AisAbstractElement.

Public Member Functions

• AisOpenCircuitElement (double duration, double samplingInterval)

the open-circuit element constructor.

AisOpenCircuitElement (const AisOpenCircuitElement &)

copy constructor for the AisOpenCircuitElement object.

AisOpenCircuitElement & operator= (const AisOpenCircuitElement &)

overload equal to operator for the AisOpenCircuitElement object.

• QString getName () const override

get the name of the element.

· QStringList getCategory () const override

get a list of applicable categories of the element.

• double getSamplingInterval () const

get how frequently we are sampling the data.

void setSamplingInterval (double samplingInterval)

set how frequently we are sampling the data.

• double getMaxDuration () const

get the value set for the duration of the experiment.

void setMaxDuration (double maxDuration)

set the value set for the duration of the experiment.

• double getMaxVoltage () const

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

void setMaxVoltage (double maxVoltage)

set a maximum voltage to stop the experiment.

• double getMinVoltage () const

get the value set minimum for the voltage in volts.

void setMinVoltage (double minVoltage)

set a minimum voltage to stop the experiment.

• double getMindVdt () const

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

• void setMindVdt (double mindVdt)

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

• bool isAutoVoltageRange () const

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

• void setAutoVoltageRange ()

set to auto-select the voltage range.

• double getApproxMaxVoltage () const

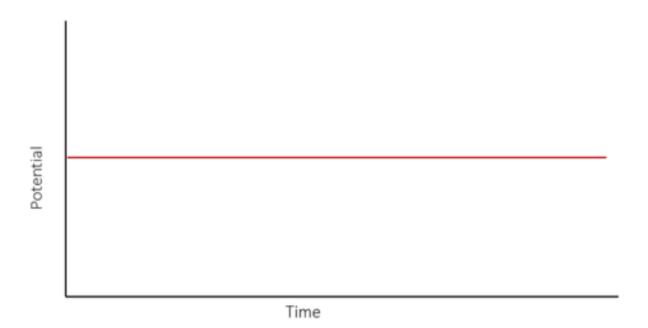
get the value set for the expected maximum voltage.

void setApproxMaxVoltage (double approxMaxVoltage)

set maximum voltage expected, for manual voltage range selection.

15.20.1 Detailed Description

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



15.20.2 Constructor & Destructor Documentation

15.20.2.1 AisOpenCircuitElement()

the open-circuit element constructor.

Parameters

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

15.20.3 Member Function Documentation

15.20.3.1 getApproxMaxVoltage()

double AisOpenCircuitElement::getApproxMaxVoltage () const

get the value set for the expected maximum voltage.

Returns

the value set for the expected maximum Voltage in volt.

Note

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

15.20.3.2 getCategory()

QStringList AisOpenCircuitElement::getCategory () const [override] get a list of applicable categories of the element.

Returns

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

15.20.3.3 getMaxDuration()

 $\label{thm:double_AisOpenCircuitElement::getMaxDuration () const} \\$ get the value set for the duration of the experiment.

Returns

the value set for the duration of the experiment in seconds.

15.20.3.4 getMaxVoltage()

```
double AisOpenCircuitElement::getMaxVoltage ( ) const
```

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

Returns

the value set for the maximum voltage.

Note

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

15.20.3.5 getMindVdt()

```
double AisOpenCircuitElement::getMindVdt ( ) const
```

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

Returns

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

Note

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

15.20.3.6 getMinVoltage()

```
double AisOpenCircuitElement::getMinVoltage ( ) const
```

get the value set minimum for the voltage in volts.

Returns

the value set for the minimum voltage in volts.

Note

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

15.20.3.7 getName()

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

get the name of the element.

Returns

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

15.20.3.8 getSamplingInterval()

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

get how frequently we are sampling the data.

Returns

the data sampling interval value in seconds.

15.20.3.9 isAutoVoltageRange()

```
bool AisOpenCircuitElement::isAutoVoltageRange ( ) const
```

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

Returns

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

15.20.3.10 setApproxMaxVoltage()

set maximum voltage expected, for manual voltage range selection.

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

Parameters

approxMaxVoltage	the value for the maximum current expected in V.

15.20.3.11 setAutoVoltageRange()

```
void AisOpenCircuitElement::setAutoVoltageRange ( )
```

set to auto-select the voltage range.

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

15.20.3.12 setMaxDuration()

set the value set for the duration of the experiment.

Parameters

maxDuration the value to set for the duration of the	experiment in seconds.
--	------------------------

15.20.3.13 setMaxVoltage()

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

- 4		
	maylaltaga	the maximum voltage value in volts at which the experiment will stop.
	max vonace	i me maximum vollage value in volls at which me experiment will stop
	martionago	the maximum vertage value in verte at which the experiment will etep.

15.20.3.14 setMindVdt()

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

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

Parameters

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

15.20.3.15 setMinVoltage()

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

15.20.3.16 setSamplingInterval()

set how frequently we are sampling the data.

Parameters

camplingInterval	the data sampling interval value in seconds.
Sampling interval	the data sampling interval value in seconds.

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

· AisOpenCircuitElement.h

15.21 AisSquareWaveVoltammetryElement Class Reference

This experiment holds the working electrode at the **starting potential** during the **quiet time**. Then it applies a train of square pulses superimposed on a staircase waveform with a uniform **potential step** magnitude.

```
#include <AisSquareWaveVoltammetryElement.h>
```

Inherits AisAbstractElement.

Public Member Functions

 AisSquareWaveVoltammetryElement (double startVoltage, double endVoltage, double voltageStep, double pulseAmp, double pulseFrequency)

the square wave element constructor

AisSquareWaveVoltammetryElement (const AisSquareWaveVoltammetryElement &)

copy constructor for the AisSquareWaveVoltammetryElement object.

AisSquareWaveVoltammetryElement & operator= (const AisSquareWaveVoltammetryElement &)

overload equal to operator for the AisSquareWaveVoltammetryElement object.

QString getName () const override

get the name of the element.

QStringList getCategory () const override

get a list of applicable categories of the element.

double getStartVoltage () const

get the value set for the start voltage.

void setStartVoltage (double startVoltage)

set the value for the start voltage.

bool isStartVoltageVsOCP () const

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

void setStartVoltageVsOCP (bool startVoltageVsOcp)

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

• double getEndVoltage () const

get the value set for the ending potential value.

void setEndVoltage (double endVoltage)

set the ending potential value.

• bool isEndVoltageVsOCP () const

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

void setEndVoltageVsOCP (bool endVoltageVsOcp)

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

• double getVStep () const

get the value set for the voltage step.

void setVStep (double vStep)

set the value for the voltage step.

• double getPulseAmp () const

get the value set for the pulse amplitude.

void setPulseAmp (double pulseAmp)

set the value for the pulse amplitude.

double getPulseFreq () const

get the value set for the pulse frequency.

void setPulseFreq (double pulseFreq)

set the value for the pulse frequency.

bool isAutoRange () const

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

• void setAutoRange ()

set to auto-select the current range.

double getApproxMaxCurrent () const

get the value set for the expected maximum current.

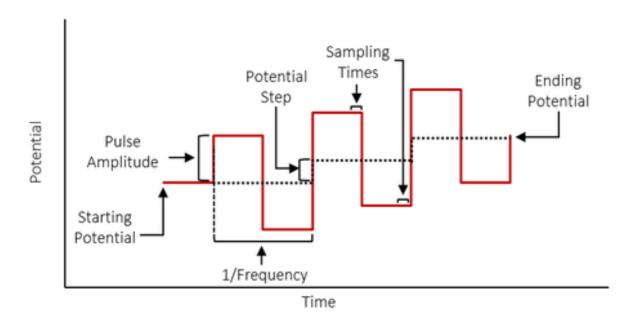
void setApproxMaxCurrent (double approxMaxCurrent)

set maximum current expected, for manual current range selection.

15.21.1 Detailed Description

This experiment holds the working electrode at the **starting potential** during the **quiet time**. Then it applies a train of square pulses superimposed on a staircase waveform with a uniform **potential step** magnitude.

The potential continues to step until the **final potential** is reached. Each square pulse consists of a forward pulse and a reverse pulse of equal in **amplitude** but opposite in direction. **Frequency** is the inverse of the total duration of a square pulse. Current responses are sampled at two points, one at the end of the forward pulse (if) and another at the end of the reverse pulse (ir). The difference in current sampled at these two points is plotted against the potential of the corresponding staircase tread.



15.21.2 Constructor & Destructor Documentation

15.21.2.1 AisSquareWaveVoltammetryElement()

the square wave element constructor

Parameters

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

15.21.3 Member Function Documentation

15.21.3.1 getApproxMaxCurrent()

double AisSquareWaveVoltammetryElement::getApproxMaxCurrent () const

get the value set for the expected maximum current.

Returns

the value set for the expected maximum current in Amps.

Note

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

15.21.3.2 getCategory()

QStringList AisSquareWaveVoltammetryElement::getCategory () const [override]

get a list of applicable categories of the element.

Returns

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

15.21.3.3 getEndVoltage()

 $\verb|double AisSquareWaveVoltammetryElement::getEndVoltage () const$

get the value set for the ending potential value.

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

Returns

the value set for the ending voltage in volts.

15.21.3.4 getName()

QString AisSquareWaveVoltammetryElement::getName () const [override] get the name of the element.

Returns

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

15.21.3.5 getPulseAmp()

 $\label{thm:const} \mbox{double AisSquareWaveVoltammetryElement::getPulseAmp () const} \\ \mbox{get the value set for the pulse amplitude.} \\$

Returns

the value set for the pulse amplitude in volts.

See also

setPulseAmp

15.21.3.6 getPulseFreq()

 $\label{thm:const} \mbox{double AisSquareWaveVoltammetryElement::getPulseFreq () const} \\ \mbox{get the value set for the pulse frequency.} \\$

Returns

the value set for the frequency in Hz.

15.21.3.7 getStartVoltage()

```
\label{thm:constraint} \mbox{double AisSquareWaveVoltammetryElement::getStartVoltage ( ) const} \\ \mbox{get the value set for the start voltage}.
```

Returns

the value of the start voltage in volts.

15.21.3.8 getVStep()

```
\label{thm:const} \mbox{double AisSquareWaveVoltammetryElement::getVStep ( ) const} \\ \mbox{get the value set for the voltage step.} \\
```

Returns

the value set for the voltage step in volts.

See also

setVStep

15.21.3.9 isAutoRange()

```
bool AisSquareWaveVoltammetryElement::isAutoRange ( ) const
```

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

Returns

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

15.21.3.10 isEndVoltageVsOCP()

```
bool AisSquareWaveVoltammetryElement::isEndVoltageVsOCP ( ) const
```

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

Returns

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

Note

if nothing is set, the default is false.

15.21.3.11 isStartVoltageVsOCP()

```
bool AisSquareWaveVoltammetryElement::isStartVoltageVsOCP ( ) const
```

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

Returns

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

Note

if nothing is set, the default is false.

See also

setStartVoltageVsOCP

15.21.3.12 setApproxMaxCurrent()

```
\label{lem:condition} \mbox{void AisSquareWaveVoltammetryElement::setApproxMaxCurrent (} \\ \mbox{double } \mbox{approxMaxCurrent} \mbox{)}
```

set maximum current expected, for manual current range selection.

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

Parameters

approxMaxCurrent the value for the maximum current expected in Amps.

15.21.3.13 setAutoRange()

```
void AisSquareWaveVoltammetryElement::setAutoRange ( )
```

set to auto-select the current range.

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

15.21.3.14 setEndVoltage()

```
void AisSquareWaveVoltammetryElement::setEndVoltage ( {\tt double} \ endVoltage \ )
```

set the ending potential value.

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

Parameters

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

15.21.3.15 setEndVoltageVsOCP()

```
void AisSquareWaveVoltammetryElement::setEndVoltageVsOCP ( bool endVoltageVsOcp )
```

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

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

Parameters

endVoltageVsOcp	true to set the end voltage to be referenced against the open-circuit voltage and false if set	1
	against the reference terminal.	

Note

by default, this is set to false.

15.21.3.16 setPulseAmp()

```
void AisSquareWaveVoltammetryElement::setPulseAmp ( \mbox{double } pulseAmp \mbox{ )} \label{eq:pulseAmp}
```

set the value for the pulse amplitude.

The voltage pulse goes up in hight by the given amplitude in addition to the starting potential (of the previous pulse). It then goes back down twice the amplitude to end up one amplitude below the starting potential (of the previous pulse).

Parameters

pulseAmp the value to set for the pulse amplitude in volts.

15.21.3.17 setPulseFreq()

```
void AisSquareWaveVoltammetryElement::setPulseFreq ( \mbox{double } pulseFreq \ )
```

15.21 AisSquareWaveVoltammetryElement Class Reference 181 set the value for the pulse frequency.

Parameters

pulseFrea	the value to set for the pulse frequency in Hz.
100	

15.21.3.18 setStartVoltage()

```
void AisSquareWaveVoltammetryElement::setStartVoltage ( \label{eq:condition} \mbox{double } startVoltage \mbox{ )}
```

set the value for the start voltage.

Parameters

15.21.3.19 setStartVoltageVsOCP()

```
void AisSquareWaveVoltammetryElement::setStartVoltageVsOCP ( bool\ startVoltageVsOcp\ )
```

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

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

Parameters

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

Note

by default, this is set to false.

15.21.3.20 setVStep()

```
void AisSquareWaveVoltammetryElement::setVStep ( \label{eq:condition} \mbox{double } vStep \mbox{ )}
```

set the value for the voltage step.

The voltage step is added to the value of the starting potential of the previous pulse to start the new pulse.

Parameters

vStep	the value for the voltage step in volts.
1-	

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

• AisSquareWaveVoltammetryElement.h

Chapter 16

File Documentation

16.1 AisCompRange.h

```
1 #ifndef SQUIDSTATLIBRARY_AISCOMPRANGE_H
2 #define SQUIDSTATLIBRARY_AISCOMPRANGE_H
4 #include "AisSquidstatGlobal.h"
5 #include <QString>
6 #include <memory>
8 class AisCompRangePrivate;
15 class SQUIDSTATLIBRARY_EXPORT AisCompRange final {
16 public:
       explicit AisCompRange(const QString& compRangeName, uint8_t bandwidthIndex, uint8_t stabilityFactor);
30
       AisCompRange(const AisCompRange&);
31
32
38
       uint8_t getBandwidthIndex() const;
39
       void setBandwidthIndex(uint8_t index);
48
53
       uint8_t getStabilityFactor() const;
54
       void setStabilityFactor(uint8_t factor);
62
63
       void setCompRangeName(const QString& compRangeName);
74
       const QString& getCompRangeName() const;
7.5
76 private:
       std::shared_ptr<AisCompRangePrivate> m_data;
78 };
80 #endif
```

16.2 AisDataPoints.h

```
1 #ifndef SQUIDSTATLIBRARY_AISDATAPOINTS_H
2 #define SQUIDSTATLIBRARY_AISDATAPOINTS_H
7 struct AisDCData {
12
      double timestamp;
      double workingElectrodeVoltage;
17
18
      double counterElectrodeVoltage;
22
      double current;
28
32
       double temperature;
33 };
34
38 struct AisACData {
```

186 File Documentation

```
43
       double timestamp;
44
48
       double frequency;
49
5.3
       double absoluteImpedance;
54
58
       double realImpedance;
59
63
       double imagImpedance;
64
68
       double phaseAngle;
69
73
       double totalHarmonicDistortion;
74
81
       double numberOfCycles;
82
       double workingElectrodeDCVoltage;
86
87
       double DCCurrent;
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
```

16.3 AisDeviceTracker.h

```
1 #ifndef SQUIDSTATLIBRARY_AISDEVICETRACKER_H
2 #define SQUIDSTATLIBRARY_AISDEVICETRACKER_H
4 #include "AisErrorCode.h"
5 #include "AisSquidstatGlobal.h"
6 #include <QObject>
7 #include <memory>
10
11 class AisDeviceTrackerPrivate;
12 class AisInstrumentHandler:
13
18 class SQUIDSTATLIBRARY_EXPORT AisDeviceTracker final : public QObject
19 {
2.0
       Q_OBJECT
21 public:
       ~AisDeviceTracker() override:
22
23
       static AisDeviceTracker *Instance();
24
37
       AisErrorCode connectToDeviceOnComPort(const QString &comPort);
38
50
       const AisInstrumentHandler &getInstrumentHandler(const QString &deviceName) const;
51
       const std::list<QString> getConnectedDevices() const;
56
       int connectAllPluggedInDevices();
80
       AisErrorCode updateFirmwareOnComPort(QString comport) const;
81
       int updateFirmwareOnAllAvailableDevices();
93
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();
        AisDeviceTracker(const AisDeviceTracker &);
113
114
        void operator=(const AisDeviceTracker &);
115
116
        std::unique_ptr<AisDeviceTrackerPrivate> m_data;
```

16.4 AisErrorCode.h

```
117
118
119 };
120
```

16.4 AisErrorCode.h

```
2 #ifndef AIS_ERROR_CODE_H
3 #define AIS_ERROR_CODE_H
5 #include "AisSquidstatGlobal.h"
6 #include <qstring.h>
15 class SQUIDSTATLIBRARY_EXPORT AisErrorCode {
17 public:
       enum ErrorCode : uint8_t {
18
           Unknown = 255,
Success = 0,
19
            ConnectionFailed = 1,
22
           FirmwareNotSupported = 2,
           FirmwareFileNotFound = 3,
23
2.4
           FirmwareUptodate = 4,
25
            InvalidChannel = 10,
            BusyChannel = 11,
28
            DeviceNotFound = 13,
30
            ManualExperimentNotRunning = 51,
31
            ExperimentNotUploaded = 52,
32
            ExperimentIsEmptv = 53.
            InvalidParameters = 54,
33
34
            ChannelNotBusy = 55,
36
            DeviceCommunicationFailed = 100,
38
            FailedToSetManualModeCurrentRange = 101,
            FailedToSetManualModeConstantVoltage = 102,
39
40
            FailedToPauseExperiment = 103,
            FailedToResumeExperiment = 104,
41
42
            FailedToStopExperiment = 105,
            FailedToUploadExperiment = 106,
ExperimentAlreadyPaused = 107,
43
44
45
            ExperimentAlreadyRun = 108,
46
       };
       AisErrorCode();
49
       AisErrorCode (ErrorCode error);
       AisErrorCode (ErrorCode error, QString message);
50
51
56
       OString message() const;
62
       int value() const;
64
       operator ErrorCode() const;
6.5
66 private:
       ErrorCode code;
67
68
       QString errorMessage;
69 };
71 #endif // ! AIS_ERROR_CODE_H
```

16.5 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
19 class SQUIDSTATLIBRARY_EXPORT AisExperiment final {
20 public:
24 explicit AisExperiment();
25
30 explicit AisExperiment (const AisExperiment& exp);
```

188 File Documentation

```
31
36
       void operator=(const AisExperiment& exp);
37
38
       ~AisExperiment();
39
45
       OString getExperimentName() const:
46
52
       QString getDescription() const;
53
59
       QStringList getCategory() const;
60
65
       void setExperimentName(QString name);
66
71
       void setDescription(QString description);
72
81
       bool appendElement(AisAbstractElement& element, uint repeat = 1);
82
89
       bool appendSubExperiment(const AisExperiment& subExp, uint repeat = 1);
90
91 private:
       friend class AisInstrumentHandler;
92
93
       std::shared_ptr<AisExperimentPrivate> m_data;
94 };
9.5
96 #endif //SQUIDSTATLIBRARY_AISCUSTOMEXPERIMENTRUNNER_H
```

16.6 AisInstrumentHandler.h

```
1 #ifndef SQUIDSTATLIBRARY_AISINSTRUMENTHANDLER_H
2 #define SQUIDSTATLIBRARY_AISINSTRUMENTHANDLER_H
4 #include <ctime>
6 #include <QObject>
8 #include "AisCompRange.h"
9 #include "AisDataPoints.h"
10 #include "AisErrorCode.h"
11 #include "AisSquidstatGlobal.h"
13 class AisInstrumentHandlerPrivate;
14 class AisExperiment;
15
24 class SOUIDSTATLIBRARY EXPORT AisInstrumentHandler final : public OObject {
25
       Q_OBJECT
       AisInstrumentHandlerPrivate* m_data;
27 public:
29
       explicit AisInstrumentHandler(AisInstrumentHandlerPrivate* privateData);
31
       ~AisInstrumentHandler();
32
55
       AisErrorCode uploadExperimentToChannel(uint8_t channel, std::shared_ptr<AisExperiment> experiment)
      const;
56
79
       AisErrorCode uploadExperimentToChannel(uint8_t channel, const AisExperiment& experiment) const;
80
       AisErrorCode startUploadedExperiment (uint8_t channel) const;
94
95
112
        AisErrorCode skipExperimentStep(uint8_t channel) const;
113
126
        AisErrorCode pauseExperiment(uint8_t channel) const;
127
140
        AisErrorCode resumeExperiment (uint8_t channel) const;
141
153
        AisErrorCode stopExperiment(uint8_t channel) const;
154
162
        double getExperimentUTCStartTime(uint8_t channel) const;
163
176
        AisErrorCode setIRComp(uint8_t channel, double uncompensatedResistance, double compensationLevel)
      const:
177
189
        AisErrorCode setCompRange(uint8_t channel, const AisCompRange& compRange) const;
190
201
        int8_t setLinkedChannels(std::vector<uint8_t> channels) const;
202
214
        int8 t setBipolarLinkedChannels(std::vector<uint8 t> channels) const;
215
216
222
        bool hasBipolarMode(uint8_t channel) const;
223
        std::vector<uint8_t> getLinkedChannels(uint8_t channel) const;
229
230
        bool isChannelBusy(uint8 t channel) const;
236
237
```

```
243
        bool isChannelPaused(uint8_t channel) const;
244
249
        std::vector<uint8_t> getFreeChannels() const;
250
255
        int getNumberOfChannels() const;
256
264
        AisErrorCode eraseRecoverData() const;
265
279
        AisErrorCode startManualExperiment(uint8_t channel) const;
280
291
        AisErrorCode setManualModeSamplingInterval (uint8_t channel, double value) const;
292
307
        AisErrorCode setManualModeOCP(uint8_t channel) const;
308
320
        AisErrorCode setManualModeConstantVoltage(uint8_t channel, double value) const;
321
336
        AisErrorCode setManualModeConstantVoltage(uint8_t channel, double value, int currentRangeIndex)
337
350
        AisErrorCode setManualModeConstantCurrent(uint8_t channel, double value) const;
351
360
        std::vector<std::pair<double, double» getManualModeCurrentRangeList(uint8_t channel) const;</pre>
361
362 signals:
363
368
        void deviceDisconnected();
369
374
        void groundFloatStateChanged(bool grounded);
375
382
        void experimentNewElementStarting(uint8_t channel, const AisExperimentNode& stepInfo);
383
389
        void activeDCDataReady(uint8_t channel, const AisDCData& DCData);
390
398
        void idleDCDataReady(uint8_t channel, const AisDCData& DCData);
399
405
        void recoveryDCDataReady(uint8_t channel, const AisDCData& DCData);
406
412
        void activeACDataReady(uint8_t channel, const AisACData& ACData);
413
419
        void recoveryACDataReady(uint8_t channel, const AisACData& ACData);
420
425
        void experimentStopped(uint8_t channel);
426
431
        void experimentPaused(uint8_t channel);
432
437
        void experimentResumed(uint8_t channel);
438
443
        void recoverDataErased(bool successful);
444
445 private slots:
446
        void onActiveExperimentNodeBeginning(uint8_t channel, const AisExperimentNode&);
447
        void onRecoveryExperimentNodeBeginning(uint8_t channel, const AisExperimentNode&);
448
        void onDeviceDisconnected();
449
450 private:
451
        void connectWithOperatorSignals();
454 #endif //SQUIDSTATLIBRARY_AISINSTRUMENTHANDLER_H
```

16.7 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
```

16.8 AisAbstractElement.h

1 #ifndef SQUIDSTATLIBRARY_AISABSTRACTELEMENT_H

190 File Documentation

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

16.9 AisConstantCurrentElement.h

```
1 #pragma once
3 #include "AisAbstractElement.h"
4 #include "AisSquidstatGlobal.h"
5 #include <QString>
7 class ConstantCurrentAdvElement;
15 class SQUIDSTATLIBRARY_EXPORT AisConstantCurrentElement final : public AisAbstractElement {
16 public:
2.3
       explicit AisConstantCurrentElement(
24
           double current,
25
           double samplingInterval,
           double duration);
26
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
       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
        void setMaxVoltage(double maxVoltage);
114
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:
        std::shared_ptr<ConstantCurrentAdvElement> m_dataDerived;
225
226 };
```

16.10 AisConstantPotElement.h

```
1 #pragma once
  #include "AisAbstractElement.h"
3
  #include "AisSquidstatGlobal.h"
5 #include <QString>
7 class ConstantPotAdvElement;
15 class SQUIDSTATLIBRARY EXPORT AisConstantPotElement final: public AisAbstractElement {
16 public:
      explicit AisConstantPotElement(
           double voltage,
25
           double samplingInterval,
2.6
           double duration);
       explicit AisConstantPotElement(const AisConstantPotElement&);
30
31
35
       AisConstantPotElement& operator=(const AisConstantPotElement&);
36
37
       ~AisConstantPotElement() override;
38
43
       QString getName() const override;
44
49
       OStringList getCategory() const override;
50
55
       double getPotential() const;
56
61
       void setPotential(double potential);
62
68
       bool isVoltageVsOCP() const;
76
       void setVoltageVsOCP(bool vsOCP);
77
82
       double getSamplingInterval() const;
83
88
       void setSamplingInterval(double samplingInterval);
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
        void setMinCurrent(double minCurrent);
141
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
        void setAutoRange();
188
189
195
        double getApproxMaxCurrent() const;
196
204
        void setApproxMaxCurrent(double approxMaxCurrent);
205
206 private:
207
        std::shared ptr<ConstantPotAdvElement> m dataDerived;
208 };
```

192 File Documentation

16.11 AisConstantPowerElement.h

```
1 #pragma once
3 #include "AisAbstractElement.h"
4 #include "AisSquidstatGlobal.h"
5 #include <QString>
7 class ConstantPowerElement;
15 class SQUIDSTATLIBRARY_EXPORT AisConstantPowerElement final : public AisAbstractElement {
16 public:
       explicit AisConstantPowerElement(
          bool isCharge,
26
           double power,
27
           double duration,
2.8
           double smaplingInterval);
29
33
       explicit AisConstantPowerElement(const AisConstantPowerElement&);
34
38
       AisConstantPowerElement& operator=(const AisConstantPowerElement&);
39
40
       ~AisConstantPowerElement() override;
41
46
       QString getName() const override;
52
       QStringList getCategory() const override;
53
58
       bool isCharge() const;
59
       void setCharge(bool isCharge);
64
65
70
       double getPower() const;
71
76
       void setPower(double power);
77
82
       double getSamplingInterval() const;
83
       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:
        std::shared_ptr<ConstantPowerElement> m_dataDerived;
158 };
```

16.12 AisConstantResistanceElement.h

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

```
48
       QStringList getCategory() const override;
49
54
       double getResistance() const;
5.5
60
       void setResistance(double resistance);
61
66
       double getSamplingInterval() const;
72
       void setSamplingInterval(double samplingInterval);
73
79
       double getMinVoltage() const;
80
       void setMinVoltage(double minVoltage);
89
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 };
```

16.13 AisCyclicVoltammetryElement.h

```
1 #pragma once
3 #include "AisAbstractElement.h"
4 #include "AisSquidstatGlobal.h"
5 #include <QString>
7 class CyclicVoltammetryElement;
22 class SQUIDSTATLIBRARY_EXPORT AisCyclicVoltammetryElement final : public AisAbstractElement {
23 public:
       explicit AisCyclicVoltammetryElement(
33
          double startVoltage,
35
           double firstVoltageLimit,
36
           double secondVoltageLimit,
37
           double endVoltage,
38
           double dEdt,
39
           double samplingInterval);
       explicit AisCyclicVoltammetryElement(const AisCyclicVoltammetryElement&);
45
49
       AisCyclicVoltammetryElement& operator=(const AisCyclicVoltammetryElement&);
50
51
       ~AisCvclicVoltammetrvElement() override;
52
       QString getName() const override;
58
63
       QStringList getCategory() const override;
64
       double getStartVoltage() const;
69
70
75
       void setStartVoltage(double startVoltage);
76
81
       bool isStartVoltageVsOCP() const;
82
       void setStartVoltageVsOCP(bool startVoltageVsOCP);
89
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
        void setSecondVoltageLimit(double v2);
140
141
147
        bool isSecondVoltageLimitVsOCP() const;
148
155
        void setSecondVoltageLimitVsOCP(bool secondVoltageLimitVsOCP);
156
        double getNumberOfCycles();
161
162
167
        void setNumberOfCycles(int cycles);
```

194 File Documentation

```
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 };
```

16.14 AisDCCurrentSweepElement.h

```
1 #pragma once
  #include "AisAbstractElement.h"
  #include "AisSquidstatGlobal.h"
5 #include <QString>
7 class DCCurrentSweepElement:
17 class SQUIDSTATLIBRARY_EXPORT AisDCCurrentSweepElement : public AisAbstractElement {
26
       explicit AisDCCurrentSweepElement(
2.7
           double startCurrent,
28
           double endCurrent.
29
           double scanRate.
30
           double samplingInterval
31
32
36
       explicit AisDCCurrentSweepElement(const AisDCCurrentSweepElement&);
40
       AisDCCurrentSweepElement& operator=(const AisDCCurrentSweepElement&);
41
       ~AisDCCurrentSweepElement() override;
42
43
48
       QString getName() const override;
49
59
       QStringList getCategory() const override;
60
65
       double getStartingCurrent() const;
       void setStartingCurrent(double startingCurrent);
72
77
       double getEndingCurrent() const;
78
83
       void setEndingCurrent(double endingCurrent);
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
        void setMaxVoltage(double maxVoltage);
128
129
135
        double getMinVoltage() const;
136
145
        void setMinVoltage(double minVoltage);
146
147 private:
148
        std::shared_ptr<DCCurrentSweepElement> m_dataDerived;
149 };
```

16.15 AisDCPotentialSweepElement.h

```
1 #pragma once
3 #include "AisAbstractElement.h"
4 #include "AisSquidstatGlobal.h"
5 #include <QString>
7 class DCPotentialSweepElement;
17 class SQUIDSTATLIBRARY_EXPORT AisDCPotentialSweepElement final : public AisAbstractElement {
18 public:
       explicit AisDCPotentialSweepElement(
          double startPotential,
           double endPotential,
29
           double scanRate,
30
           double samplingInterval);
       explicit AisDCPotentialSweepElement(const AisDCPotentialSweepElement&);
34
38
       AisDCPotentialSweepElement& operator=(const AisDCPotentialSweepElement&);
40
       ~AisDCPotentialSweepElement() override;
41
46
       QString getName() const override;
       QStringList getCategory() const override;
56
57
62
       double getStartingPot() const;
68
       void setStartingPot(double startingPotential);
69
       bool isStartVoltageVsOCP() const;
75
76
       void setStartVoltageVsOCP(bool startVoltageVsOCP);
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);
174 private:
175
        std::shared_ptr<DCPotentialSweepElement> m_dataDerived;
176 };
```

16.16 AisDiffPulseVoltammetryElement.h

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

196 File Documentation

```
48
       ~AisDiffPulseVoltammetryElement() override;
49
54
       QString getName() const override;
5.5
6.5
       QStringList getCategory() const override;
66
71
       double getStartVoltage() const;
72
77
       void setStartVoltage(double startVoltage);
78
       bool isStartVoltageVsOCP() const;
84
85
93
       void setStartVoltageVsOCP(bool startVoltageVsOCP);
94
101
        double getEndVoltage() const;
102
        void setEndVoltage(double endVoltage);
109
110
116
        bool isEndVoltageVsOCP() const;
117
125
        void setEndVoltageVsOCP(bool endVoltageVsOCP);
126
132
        double getVStep() const;
133
140
        void setVStep(double vStep);
141
147
        double getPulseHeight() const;
148
        void setPulseHeight(double pulseHeight);
157
158
164
        double getPulseWidth() const;
165
173
        void setPulseWidth(double pulseWidth);
174
180
        double getPulsePeriod() const;
181
188
        void setPulsePeriod(double pulsePeriod);
189
194
        bool isAutoRange() const;
195
201
        void setAutoRange();
202
208
        double getApproxMaxCurrent() const;
209
217
        void setApproxMaxCurrent(double approxMaxCurrent);
218
219 private:
220
        \verb|std::shared_ptr<DiffPulseVoltammetryElement>| m_dataDerived;|\\
221 };
```

16.17 AisEISGalvanostaticElement.h

```
1 #pragma once
3 #include "AisAbstractElement.h"
4 #include "AisSquidstatGlobal.h"
5 #include <QString>
7 class EISGalvanostaticElement;
20 class SQUIDSTATLIBRARY_EXPORT AisEISGalvanostaticElement final : public AisAbstractElement {
21 public:
       explicit AisEISGalvanostaticElement(
30
31
          double startFrequency,
           double endFrequency,
33
           double stepsPerDecade,
34
           double currentBias,
           double currentAamplitude);
3.5
       explicit AisEISGalvanostaticElement(const AisEISGalvanostaticElement&);
39
43
       AisEISGalvanostaticElement& operator=(const AisEISGalvanostaticElement&);
44
45
       ~AisEISGalvanostaticElement() override;
46
51
       QString getName() const override;
52
62
       QStringList getCategory() const override;
63
68
       double getStartFreq() const;
69
74
       void setStartFreq(double startFreq);
75
       double getEndFreg() const;
80
```

```
86
       void setEndFreq(double endFreq);
92
       double getStepsPerDecade() const;
93
98
       void setStepsPerDecade(double stepsPerDecade);
99
104
       double getBiasCurrent() const;
105
110
        void setBiasCurrent(double biasCurrent);
111
        double getAmplitude() const;
116
117
122
        void setAmplitude(double amplitude);
123
124 private:
125
        std::shared_ptr<EISGalvanostaticElement> m_dataDerived;
126 1:
```

16.18 AisEISPotentiostaticElement.h

```
1 #pragma once
3 #include "AisAbstractElement.h"
4 #include "AisSquidstatGlobal.h"
5 #include <OString>
7 class EISPotentiostaticElement;
20 class SQUIDSTATLIBRARY_EXPORT AisEISPotentiostaticElement final : public AisAbstractElement {
21 public:
       explicit AisEISPotentiostaticElement(
30
          double startFrequency.
31
32
          double endFrequency,
33
          double stepsPerDecade,
34
          double voltageBias,
35
           double voltageAamplitude);
       explicit AisEISPotentiostaticElement(const AisEISPotentiostaticElement&);
39
43
       AisEISPotentiostaticElement& operator=(const AisEISPotentiostaticElement&);
44
45
       ~AisEISPotentiostaticElement() override;
46
51
       QString getName() const override;
52
57
       QStringList getCategory() const override;
58
       double getStartFreq() const;
69
       void setStartFreq(double startFreq);
70
       double getEndFreq() const;
75
76
81
       void setEndFreq(double endFreq);
87
       double getStepsPerDecade() const;
88
       void setStepsPerDecade(double stepsPerDecade);
93
94
99
       double getBiasVoltage() const;
100
105
        void setBiasVoltage(double biasVoltage);
106
        bool isBiasVoltageVsOCP() const;
113
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 };
```

16.19 AisNormalPulseVoltammetryElement.h

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

198 File Documentation

```
5 #include <QString>
7 class NormalPulseVoltammetryElement;
21 class SQUIDSTATLIBRARY_EXPORT AisNormalPulseVoltammetryElement final : public AisAbstractElement {
22 public:
31
       explicit AisNormalPulseVoltammetryElement(
          double startVoltage,
33
           double endVoltage,
34
           double voltageStep,
35
           double pulseWidth,
36
           double pulsePeriod);
       explicit AisNormalPulseVoltammetryElement(const AisNormalPulseVoltammetryElement&);
40
       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;
6.5
70
       void setStartVoltage(double startVoltage);
71
       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
        void setEndVoltageVsOCP(bool endVoltageVsOcp);
119
120
126
        double getVStep() const;
127
134
        void setVStep(double vStep);
135
        double getPulseWidth() const;
142
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 };
```

16.20 AisOpenCircuitElement.h

```
1 #pragma once
3 #include "AisAbstractElement.h"
4 #include "AisSquidstatGlobal.h"
5 #include <QString>
7 class OpenCircuitElement;
15 class SQUIDSTATLIBRARY_EXPORT AisOpenCircuitElement final : public AisAbstractElement {
16 public:
2.2
       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;
```

```
46
       QStringList getCategory() const override;
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);
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;
```

16.21 AisSquareWaveVoltammetryElement.h

```
1 #pragma once
3 #include "AisAbstractElement.h"
  #include "AisSquidstatGlobal.h"
5 #include <QString>
7 class SquareWaveVoltammetrvElement:
23 class SQUIDSTATLIBRARY_EXPORT AisSquareWaveVoltammetryElement final : public AisAbstractElement {
      explicit AisSquareWaveVoltammetryElement(
34
           double startVoltage,
35
           double endVoltage,
          double voltageStep
36
37
           double pulseAmp,
          double pulseFrequency);
39
43
       explicit AisSquareWaveVoltammetryElement(const AisSquareWaveVoltammetryElement&);
47
       AisSquareWaveVoltammetryElement& operator=(const AisSquareWaveVoltammetryElement&);
48
49
       ~AisSquareWaveVoltammetryElement() override;
50
       QString getName() const override;
56
61
       QStringList getCategory() const override;
62
       double getStartVoltage() const;
67
68
       void setStartVoltage(double startVoltage);
74
81
       bool isStartVoltageVsOCP() const;
82
90
       void setStartVoltageVsOCP(bool startVoltageVsOcp);
91
       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
        double getPulseAmp() const;
```

200 File Documentation

```
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;
186
194 void setApproxMaxCurrent(double approxMaxCurrent);
195
196 private:
197 std::shared_ptr<SquareWaveVoltammetryElement> m_dataDerived;
198 };
```

Index

activeACDataReady	getMaxDuration, 64
AisInstrumentHandler, 141	getMinCurrent, 64
activeDCDataReady	getMinOdrient, 64
AisInstrumentHandler, 141	getName, 65
AisAbstractElement.h, 189	getPotential, 65
AisACData, 47	getSamplingInterval, 66
numberOfCycles, 48	isAutoRange, 66
AisCompRange, 48	isVoltageVsOCP, 66
AisCompRange, 49	setApproxMaxCurrent, 66
· -	• •
getBandwidthIndex, 49	setAutoRange, 67
getCompRangeName, 49	setMaxCapacity, 67
getStabilityFactor, 50	setMaxCurrent, 67
setBandwidthIndex, 50	setMaxDuration, 67
setCompRangeName, 50	setMinCurrent, 68
setStabilityFactor, 51	setMindIdt, 68
AisCompRange.h, 185	setPotential, 68
AisConstantCurrentElement, 51	setSamplingInterval, 69
AisConstantCurrentElement, 53	setVoltageVsOCP, 69
getApproxMaxCurrent, 53	AisConstantPotElement.h, 191
getApproxMaxVoltage, 54	AisConstantPowerElement, 69
getCategory, 54	AisConstantPowerElement, 71
getCurrent, 54	getCategory, 72
getMaxCapacity, 55	getMaxCapacity, 72
getMaxDuration, 55	getMaxDuration, 72
getMaxVoltage, 55	getMaxVoltage, 72
getMinSamplingVoltageDifference, 55	getMinVoltage, 73
getMinVoltage, 56	getName, 73
getName, 56	getPower, 73
getSamplingInterval, 56	getSamplingInterval, 74
isAutoRange, 57	isCharge, 74
isAutoVoltageRange, 57	setCharge, 74
setApproxMaxCurrent, 57	setMaxCapacity, 75
setApproxMaxVoltage, 58	setMaxDuration, 75
setAutoRange, 58	setMaxVoltage, 75
setAutoVoltageRange, 58	setMinVoltage, 76
setCurrent, 58	setPower, 76
setMaxCapacity, 58	setSamplingInterval, 76
setMaxDuration, 59	
setMaxVoltage, 59	AisConstantPowerElement.h, 192 AisConstantResistanceElement, 77
setMinSamplingVoltageDifference, 59	AisConstantResistanceElement, 77 AisConstantResistanceElement, 78
setMinVoltage, 60	getCategory, 78
setSamplingInterval, 60	getMaxCapacity, 79
AisConstantCurrentElement.h, 190	getMaxDuration, 79
AisConstantPotElement, 61	getMinVoltage, 79
AisConstantPotElement, 62	getName, 79
getApproxMaxCurrent, 63	getResistance, 80
getCategory, 63	getSamplingInterval, 80
getMaxCapacity, 63	setMaxCapacity, 80
getMaxCurrent, 64	setMaxDuration, 81

setMinVoltage, 81	getEndingPot, 102
setResistance, 81	getName, 102
setSamplingInterval, 82	getSamplingInterval, 103
AisConstantResistanceElement.h, 192	getScanRate, 103
AisCyclicVoltammetryElement, 82	getStartingPot, 103
AisCyclicVoltammetryElement, 84	isAutoRange, 103
getApproxMaxCurrent, 84	isEndVoltageVsOCP, 104
getCategory, 84	isStartVoltageVsOCP, 104
getdEdt, 85	setApproxMaxCurrent, 104
getEndVoltage, 85	setAutoRange, 105
getFirstVoltageLimit, 85	setEndingPot, 105
getName, 85	setEndVoltageVsOCP, 105
getNumberOfCycles, 86	setSamplingInterval, 105
getSamplingInterval, 86	setScanRate, 106
getSecondVoltageLimit, 86	setStartingPot, 106
getStartVoltage, 86	setStartVoltageVsOCP, 106
isAutoRange, 87	AisDCPotentialSweepElement.h, 195
isEndVoltageVsOCP, 87	AisDeviceTracker, 107
isFirstVoltageLimitVsOCP, 87	connectAllPluggedInDevices, 108
isSecondVoltageLimitVsOCP, 87	connectToDeviceOnComPort, 108
isStartVoltageVsOCP, 88	deviceDisconnected, 109
setApproxMaxCurrent, 88	getConnectedDevices, 109
setAutoRange, 88	getInstrumentHandler, 109
setdEdt, 88	newDeviceConnected, 110
setEndVoltage, 89	updateFirmwareOnAllAvailableDevices, 110
setEndVoltageVsOCP, 89	updateFirmwareOnComPort, 111
setFirstVoltageLimit, 89	AisDeviceTracker.h, 186
setFirstVoltageLimitVsOCP, 90	AisDiffPulseVoltammetryElement, 111
setNumberOfCycles, 90	AisDiffPulseVoltammetryElement, 113
setSamplingInterval, 90	getApproxMaxCurrent, 114
setSecondVoltageLimit, 91	getCategory, 114
setSecondVoltageLimitVsOCP, 91	getEndVoltage, 114
setStartVoltage, 91	getName, 114
setStartVoltageVsOCP, 92	getPulseHeight, 115
AisCyclicVoltammetryElement.h, 193	getPulsePeriod, 115
AisDataPoints.h, 185	getPulseWidth, 115
AisDCCurrentSweepElement, 92	getStartVoltage, 116
AisDCCurrentSweepElement, 94	getVStep, 116
getCategory, 95	isAutoRange, 116
getEndingCurrent, 95	isEndVoltageVsOCP, 117
getMaxVoltage, 95	isStartVoltageVsOCP, 117
getMinVoltage, 95	setApproxMaxCurrent, 117
getName, 96	setAutoRange, 118
getSamplingInterval, 96	setEndVoltage, 118
getScanRate, 96	setEndVoltageVsOCP, 118
getStartingCurrent, 96	setPulseHeight, 118
setEndingCurrent, 97	setPulsePeriod, 119
setMaxVoltage, 97	setPulseWidth, 119
setMinVoltage, 97	setStartVoltage, 119
setSamplingInterval, 98	setStartVoltageVsOCP, 120
setScanRate, 98	setVStep, 120
setStartingCurrent, 98	AisDiffPulseVoltammetryElement.h, 195
AisDCCurrentSweepElement.h, 194	AisEISGalvanostaticElement, 121
AisDCData, 99	AisEISGalvanostaticElement, 122
AisDCPotentialSweepElement, 99	getAmplitude, 123
AisDCPotentialSweepElement, 101	getBiasCurrent, 123
getApproxMaxCurrent, 102	getCategory, 123
getCategory, 102	getEndFreq, 123
<u> </u>	U 12

getName, 124	getExperimentName, 137
getStartFreq, 124	operator=, 137
getStepsPerDecade, 124	setDescription, 138
setAmplitude, 124	setExperimentName, 138
setBiasCurrent, 125	AisExperiment.h, 187
setEndFreq, 125	AisExperimentNode, 138
setStartFreq, 125	AisInstrumentHandler, 139
setStepsPerDecade, 125	activeACDataReady, 141
AisEISGalvanostaticElement.h, 196	activeDCDataReady, 141
AisEISPotentiostaticElement, 126	deviceDisconnected, 142
AisEISPotentiostaticElement, 127	eraseRecoverData, 142
getAmplitude, 128	experimentNewElementStarting, 142
getBiasVoltage, 128	experimentPaused, 142
getCategory, 128	experimentResumed, 144
getEndFreq, 129	experimentStopped, 144
getName, 129	getExperimentUTCStartTime, 144
getStartFreq, 129	getFreeChannels, 145
getStepsPerDecade, 129	getLinkedChannels, 145
isBiasVoltageVsOCP, 130	getManualModeCurrentRangeList, 145
setAmplitude, 130	getNumberOfChannels, 146
setBiasVoltage, 130	groundFloatStateChanged, 146
setBiasVoltageVsOCP, 131	hasBipolarMode, 146
setEndFreq, 131	idleDCDataReady, 146
setStartFreq, 131	isChannelBusy, 147
setStepsPerDecade, 131	isChannelPaused, 147
AisEISPotentiostaticElement.h, 197	pauseExperiment, 148
AisErrorCode, 132	recoverDataErased, 148
BusyChannel, 133	recoveryACDataReady, 148
ChannelNotBusy, 133	recoveryDCDataReady, 149
ConnectionFailed, 133	resumeExperiment, 149
DeviceCommunicationFailed, 133	setBipolarLinkedChannels, 149
DeviceNotFound, 133	setCompRange, 150
ErrorCode, 133	setIRComp, 151
ExperimentAlreadyPaused, 134	setLinkedChannels, 151
ExperimentAlreadyRun, 134	setManualModeConstantCurrent, 152
ExperimentlsEmpty, 133	setManualModeConstantVoltage, 152, 153
ExperimentNotUploaded, 133	setManualModeOCP, 153
FailedToPauseExperiment, 133	setManualModeSamplingInterval, 154
FailedToResumeExperiment, 134	skipExperimentStep, 154
FailedToSetManualModeConstantVoltage, 133	startManualExperiment, 155
FailedToSetManualModeCurrentRange, 133	startUploadedExperiment, 155
FailedToStopExperiment, 134	stopExperiment, 156
FailedToUploadExperiment, 134	uploadExperimentToChannel, 156, 157
FirmwareNotSupported, 133	AisInstrumentHandler.h, 188
InvalidChannel, 133	AisNormalPulseVoltammetryElement, 158
InvalidParameters, 133	AisNormalPulseVoltammetryElement, 160
ManualExperimentNotRunning, 133	getApproxMaxCurrent, 160
message, 134	getCategory, 161
Success, 133	getEndVoltage, 161
Unknown, 133	getName, 161
value, 134	getPulsePeriod, 161
AisErrorCode.h, 187	getPulseWidth, 162
AisExperiment, 135	getStartVoltage, 162
AisExperiment, 135	getVStep, 162
appendElement, 136	isAutoRange, 163
appendSubExperiment, 136	isEndVoltageVsOCP, 163
getCategory, 137	isStartVoltageVsOCP, 163
getDescription, 137	setApproxMaxCurrent, 164
gernescription, 137	seinpproximaxourrent, 104

setAutoRange, 164	ChannelNotBusy
setEndVoltage, 164	AisErrorCode, 133
setEndVoltageVsOCP, 165	connectAllPluggedInDevices
setPulsePeriod, 165	AisDeviceTracker, 108
setPulseWidth, 165	ConnectionFailed
setStartVoltage, 166	AisErrorCode, 133
setStartVoltageVsOCP, 166	connectToDeviceOnComPort
setVStep, 166	AisDeviceTracker, 108
AisNormalPulseVoltammetryElement.h, 197	,
AisOpenCircuitElement, 167	DeviceCommunicationFailed
AisOpenCircuitElement, 168	AisErrorCode, 133
getApproxMaxVoltage, 169	deviceDisconnected
getCategory, 169	AisDeviceTracker, 109
getMaxDuration, 169	AisInstrumentHandler, 142
getMaxVoltage, 169	DeviceNotFound
getMindVdt, 170	AisErrorCode, 133
getMinVoltage, 170	,
getName, 170	eraseRecoverData
getSamplingInterval, 171	AisInstrumentHandler, 142
isAutoVoltageRange, 171	ErrorCode
setApproxMaxVoltage, 171	AisErrorCode, 133
setAutoVoltageRange, 172	ExperimentAlreadyPaused
setMaxDuration, 172	AisErrorCode, 134
setMaxVoltage, 172	ExperimentAlreadyRun
_	AisErrorCode, 134
setMinVoltage 172	ExperimentIsEmpty
setMinVoltage, 173	AisErrorCode, 133
setSamplingInterval, 173	experimentNewElementStarting
AisOpenCircuitElement.h, 198	AisInstrumentHandler, 142
AisSquareWaveVoltammetryElement, 173	ExperimentNotUploaded
AisSquareWaveVoltammetryElement, 175	AisErrorCode, 133
getApproxMaxCurrent, 176	experimentPaused
getCategory, 176	AisInstrumentHandler, 142
getEndVoltage, 176	experimentResumed
getName, 177	AisInstrumentHandler, 144
getPulseAmp, 177	experimentStopped
getPulseFreq, 177	AisInstrumentHandler, 144
getStartVoltage, 177	Alsinstiumentinantier, 144
getVStep, 178	FailedToPauseExperiment
isAutoRange, 178	AisErrorCode, 133
isEndVoltageVsOCP, 178	FailedToResumeExperiment
isStartVoltageVsOCP, 178	AisErrorCode, 134
setApproxMaxCurrent, 179	FailedToSetManualModeConstantVoltage
setAutoRange, 179	
setEndVoltage, 179	AisErrorCode, 133
setEndVoltageVsOCP, 180	FailedToSetManualModeCurrentRange
setPulseAmp, 180	AisErrorCode, 133
setPulseFreq, 180	FailedToStopExperiment
setStartVoltage, 182	AisErrorCode, 134
setStartVoltageVsOCP, 182	FailedToUploadExperiment
setVStep, 182	AisErrorCode, 134
AisSquareWaveVoltammetryElement.h, 199	FirmwareNotSupported
AisSquidstatGlobal.h, 189	AisErrorCode, 133
appendElement	and American
AisExperiment, 136	getAmplitude
appendSubExperiment	AisEISGalvanostaticElement, 123
AisExperiment, 136	AisEISPotentiostaticElement, 128
AGEAPORITORS, 100	getApproxMaxCurrent
BusyChannel	AisConstantCurrentElement, 53
AisErrorCode, 133	AisConstantPotElement, 63

AisCyclicVoltammetryElement, 84	AisInstrumentHandler, 145
AisDCPotentialSweepElement, 102	getInstrumentHandler
AisDiffPulseVoltammetryElement, 114	AisDeviceTracker, 109
AisNormalPulseVoltammetryElement, 160	getLinkedChannels
AisSquareWaveVoltammetryElement, 176	AisInstrumentHandler, 145
getApproxMaxVoltage	getManualModeCurrentRangeList
AisConstantCurrentElement, 54	AisInstrumentHandler, 145
AisOpenCircuitElement, 169	getMaxCapacity
getBandwidthIndex	AisConstantCurrentElement, 55
AisCompRange, 49	AisConstantPotElement, 63
getBiasCurrent	AisConstantPowerElement, 72
AisEISGalvanostaticElement, 123	AisConstantResistanceElement, 79
getBiasVoltage	getMaxCurrent
AisEISPotentiostaticElement, 128	AisConstantPotElement, 64
getCategory	getMaxDuration
AisConstantCurrentElement, 54	AisConstantCurrentElement, 55
AisConstantPotElement, 63	AisConstantPotElement, 64
AisConstantPowerElement, 72	AisConstantPowerElement, 72
AisConstantResistanceElement, 78	AisConstantResistanceElement, 79
AisCyclicVoltammetryElement, 84	AisOpenCircuitElement, 169
AisDCCurrentSweepElement, 95	getMaxVoltage
AisDCPotentialSweepElement, 102	AisConstantCurrentElement, 55
AisDiffPulseVoltammetryElement, 114	AisConstantPowerElement, 72
AisEISGalvanostaticElement, 123	AisDCCurrentSweepElement, 95
AisEISPotentiostaticElement, 128	AisOpenCircuitElement, 169
AisExperiment, 137	getMinCurrent
AisNormalPulseVoltammetryElement, 161	AisConstantPotElement, 64
AisOpenCircuitElement, 169	getMindldt
AisSquareWaveVoltammetryElement, 176	AisConstantPotElement, 65
getCompRangeName	getMindVdt
AisCompRange, 49	AisOpenCircuitElement, 170
getConnectedDevices	getMinSamplingVoltageDifference
AisDeviceTracker, 109	AisConstantCurrentElement, 55
getCurrent	getMinVoltage
AisConstantCurrentElement, 54	AisConstantCurrentElement, 56
getdEdt	AisConstantPowerElement, 73
AisCyclicVoltammetryElement, 85	AisConstantResistanceElement, 79
getDescription	AisDCCurrentSweepElement, 95
AisExperiment, 137	AisOpenCircuitElement, 170
getEndFreq	getName
AisEISGalvanostaticElement, 123	AisConstantCurrentElement, 56
AisEISPotentiostaticElement, 129	AisConstantPotElement, 65
getEndingCurrent	AisConstantPowerElement, 73
AisDCCurrentSweepElement, 95	AisConstantResistanceElement, 79
getEndingPot	AisCyclicVoltammetryElement, 85
AisDCPotentialSweepElement, 102	AisDCCurrentSweepElement, 96
getEndVoltage	AisDCPotentialSweepElement, 102
AisCyclicVoltammetryElement, 85	AisDiffPulseVoltammetryElement, 114
AisDiffPulseVoltammetryElement, 114	AisEISGalvanostaticElement, 124
AisNormalPulseVoltammetryElement, 161	AisEISPotentiostaticElement, 129
AisSquareWaveVoltammetryElement, 176	AisNormalPulseVoltammetryElement, 161
getExperimentName	AisOpenCircuitElement, 170
AisExperiment, 137	AisSquareWaveVoltammetryElement, 177
getExperimentUTCStartTime	getNumberOfChannels
AisInstrumentHandler, 144	AisInstrumentHandler, 146
getFirstVoltageLimit	getNumberOfCycles
AisCyclicVoltammetryElement, 85	AisCyclicVoltammetryElement, 86
getFreeChannels	getPotential

AisConstantPotElement, 65	AisInstrumentHandler, 146
getPower	InvalidChannel
AisConstantPowerElement, 73	AisErrorCode, 133
getPulseAmp	InvalidParameters
AisSquareWaveVoltammetryElement, 177	AisErrorCode, 133
getPulseFreq	isAutoRange
AisSquareWaveVoltammetryElement, 177	AisConstantCurrentElement, 57
getPulseHeight	AisConstantPotElement, 66
AisDiffPulseVoltammetryElement, 115	AisCyclicVoltammetryElement, 87
getPulsePeriod	AisDCPotentialSweepElement, 103
AisDiffPulseVoltammetryElement, 115	AisDiffPulseVoltammetryElement, 116
AisNormalPulseVoltammetryElement, 161	AisNormalPulseVoltammetryElement, 163
getPulseWidth	AisSquareWaveVoltammetryElement, 178
AisDiffPulseVoltammetryElement, 115	isAutoVoltageRange
AisNormalPulseVoltammetryElement, 162	AisConstantCurrentElement, 57
getResistance	AisOpenCircuitElement, 171
AisConstantResistanceElement, 80	isBiasVoltageVsOCP
getSamplingInterval	AisEISPotentiostaticElement, 130
AisConstantCurrentElement, 56	isChannelBusy
AisConstantPotElement, 66	AisInstrumentHandler, 147
AisConstantPowerElement, 74	isChannelPaused
AisConstantResistanceElement, 80	AisInstrumentHandler, 147
AisCyclicVoltammetryElement, 86	isCharge
AisDCCurrentSweepElement, 96	AisConstantPowerElement, 74
AisDCPotentialSweepElement, 103	isEndVoltageVsOCP
AisOpenCircuitElement, 171	AisCyclicVoltammetryElement, 87
getScanRate	AisDCPotentialSweepElement, 104
AisDCCurrentSweepElement, 96	AisDiffPulseVoltammetryElement, 117
AisDCPotentialSweepElement, 103	AisNormalPulseVoltammetryElement, 163
getSecondVoltageLimit	AisSquareWaveVoltammetryElement, 178
AisCyclicVoltammetryElement, 86	isFirstVoltageLimitVsOCP
getStabilityFactor	AisCyclicVoltammetryElement, 87
AisCompRange, 50	isSecondVoltageLimitVsOCP
getStartFreq	AisCyclicVoltammetryElement, 87
AisEISGalvanostaticElement, 124	isStartVoltageVsOCP
AisEISPotentiostaticElement, 129	AisCyclicVoltammetryElement, 88
getStartingCurrent	AisDCPotentialSweepElement, 104
AisDCCurrentSweepElement, 96	AisDiffPulseVoltammetryElement, 117
getStartingPot	AisNormalPulseVoltammetryElement, 163
AisDCPotentialSweepElement, 103	AisSquareWaveVoltammetryElement, 178
getStartVoltage	isVoltageVsOCP
AisCyclicVoltammetryElement, 86	AisConstantPotElement, 66
AisDiffPulseVoltammetryElement, 116	
AisNormalPulseVoltammetryElement, 162	ManualExperimentNotRunning
AisSquareWaveVoltammetryElement, 177	AisErrorCode, 133
getStepsPerDecade	message
AisEISGalvanostaticElement, 124	AisErrorCode, 134
AisEISPotentiostaticElement, 129	now Device Commented
getVStep	newDeviceConnected
AisDiffPulseVoltammetryElement, 116	AisDeviceTracker, 110
AisNormalPulseVoltammetryElement, 162	numberOfCycles
AisSquareWaveVoltammetryElement, 178	AisACData, 48
groundFloatStateChanged	operator=
AisInstrumentHandler, 146	AisExperiment, 137
	AUCENPORTION, 107
hasBipolarMode	pauseExperiment
AisInstrumentHandler, 146	AisInstrumentHandler, 148
idleDCDataReady	recoverDataErased

AisInstrumentHandler, 148	AisDCCurrentSweepElement, 97
recoveryACDataReady	setEndingPot
AisInstrumentHandler, 148	AisDCPotentialSweepElement, 105
recoveryDCDataReady	setEndVoltage
AisInstrumentHandler, 149	AisCyclicVoltammetryElement, 89
resumeExperiment	AisDiffPulseVoltammetryElement, 118
AisInstrumentHandler, 149	AisNormalPulseVoltammetryElement, 164
	AisSquareWaveVoltammetryElement, 179
setAmplitude	setEndVoltageVsOCP
AisEISGalvanostaticElement, 124	AisCyclicVoltammetryElement, 89
AisEISPotentiostaticElement, 130	AisDCPotentialSweepElement, 105
setApproxMaxCurrent	AisDiffPulseVoltammetryElement, 118
AisConstantCurrentElement, 57	AisNormalPulseVoltammetryElement, 165
AisConstantPotElement, 66	AisSquareWaveVoltammetryElement, 180
AisCyclicVoltammetryElement, 88	setExperimentName
AisDCPotentialSweepElement, 104	AisExperiment, 138
AisDiffPulseVoltammetryElement, 117	setFirstVoltageLimit
AisNormalPulseVoltammetryElement, 164	AisCyclicVoltammetryElement, 89
AisSquareWaveVoltammetryElement, 179	setFirstVoltageLimitVsOCP
setApproxMaxVoltage	AisCyclicVoltammetryElement, 90
AisConstantCurrentElement, 58	setIRComp
AisOpenCircuitElement, 171	AisInstrumentHandler, 151
setAutoRange	setLinkedChannels
AisConstantCurrentElement, 58	AisInstrumentHandler, 151
AisConstantPotElement, 67	setManualModeConstantCurrent
AisCyclicVoltammetryElement, 88	AisInstrumentHandler, 152
AisDCPotentialSweepElement, 105	setManualModeConstantVoltage
AisDiffPulseVoltammetryElement, 118	AisInstrumentHandler, 152, 153
AisNormalPulseVoltammetryElement, 164	setManualModeOCP
AisSquareWaveVoltammetryElement, 179	
setAutoVoltageRange	AisInstrumentHandler, 153
AisConstantCurrentElement, 58	setManualModeSamplingInterval
AisOpenCircuitElement, 172	AisInstrumentHandler, 154
setBandwidthIndex	setMaxCapacity
AisCompRange, 50	AisConstantCurrentElement, 58
setBiasCurrent	AisConstantPotElement, 67
AisEISGalvanostaticElement, 125	AisConstantPowerElement, 75
setBiasVoltage	AisConstantResistanceElement, 80
AisEISPotentiostaticElement, 130	setMaxCurrent
setBiasVoltageVsOCP	AisConstantPotElement, 67
AisEISPotentiostaticElement, 131	setMaxDuration
setBipolarLinkedChannels	AisConstantCurrentElement, 59
AisInstrumentHandler, 149	AisConstantPotElement, 67
	AisConstantPowerElement, 75
setCharge	AisConstantResistanceElement, 81
AisConstantPowerElement, 74	AisOpenCircuitElement, 172
setCompRange	setMaxVoltage
AisInstrumentHandler, 150	AisConstantCurrentElement, 59
setCompRangeName	AisConstantPowerElement, 75
AisCompRange, 50	AisDCCurrentSweepElement, 97
setCurrent	AisOpenCircuitElement, 172
AisConstantCurrentElement, 58	setMinCurrent
setdEdt	AisConstantPotElement, 68
AisCyclicVoltammetryElement, 88	setMindldt
setDescription	AisConstantPotElement, 68
AisExperiment, 138	setMindVdt
setEndFreq	AisOpenCircuitElement, 172
AisEISGalvanostaticElement, 125	setMinSamplingVoltageDifference
AisEISPotentiostaticElement, 131	AisConstantCurrentElement, 59
setEndingCurrent	,

setMinVoltage	AisDCPotentialSweepElement, 106
AisConstantCurrentElement, 60	AisDiffPulseVoltammetryElement, 120
AisConstantPowerElement, 76	AisNormalPulseVoltammetryElement, 166
AisConstantResistanceElement, 81	AisSquareWaveVoltammetryElement, 182
AisDCCurrentSweepElement, 97	setStepsPerDecade
AisOpenCircuitElement, 173	AisEISGalvanostaticElement, 125
setNumberOfCycles	AisEISPotentiostaticElement, 131
AisCyclicVoltammetryElement, 90	setVoltageVsOCP
setPotential	AisConstantPotElement, 69
AisConstantPotElement, 68	setVStep
setPower	AisDiffPulseVoltammetryElement, 120
AisConstantPowerElement, 76	AisNormalPulseVoltammetryElement, 166
setPulseAmp	AisSquareWaveVoltammetryElement, 182
AisSquareWaveVoltammetryElement, 180	skipExperimentStep
setPulseFreq	AisInstrumentHandler, 154
AisSquareWaveVoltammetryElement, 180	startManualExperiment
setPulseHeight	AisInstrumentHandler, 155
AisDiffPulseVoltammetryElement, 118	startUploadedExperiment
setPulsePeriod	AisInstrumentHandler, 155
AisDiffPulseVoltammetryElement, 119	stopExperiment
AisNormalPulseVoltammetryElement, 165	AisInstrumentHandler, 156
setPulseWidth	Success
AisDiffPulseVoltammetryElement, 119	AisErrorCode, 133
AisNormalPulseVoltammetryElement, 165	
setResistance	Unknown
AisConstantResistanceElement, 81	AisErrorCode, 133
setSamplingInterval	updateFirmwareOnAllAvailableDevices
AisConstantCurrentElement, 60	AisDeviceTracker, 110
AisConstantPotElement, 69	updateFirmwareOnComPort
AisConstantPowerElement, 76	AisDeviceTracker, 111
AisConstantResistanceElement, 82	uploadExperimentToChannel
AisCyclicVoltammetryElement, 90	AisInstrumentHandler, 156, 157
AisDCCurrentSweepElement, 98	
AisDCPotentialSweepElement, 105	value
AisOpenCircuitElement, 173	AisErrorCode, 134
setScanRate	
AisDCCurrentSweepElement, 98	
AisDCPotentialSweepElement, 106	
setSecondVoltageLimit	
AisCyclicVoltammetryElement, 91	
setSecondVoltageLimitVsOCP	
AisCyclicVoltammetryElement, 91	
setStabilityFactor	
AisCompRange, 51	
setStartFreq	
AisEISGalvanostaticElement, 125	
AisEISPotentiostaticElement, 131	
setStartingCurrent	
AisDCCurrentSweepElement, 98	
setStartingPot	
AisDCPotentialSweepElement, 106	
setStartVoltage	
AisCyclicVoltammetryElement, 91	
AisDiffPulseVoltammetryElement, 119	
AisNormalPulseVoltammetryElement, 166	
AisSquareWaveVoltammetryElement, 182	
setStartVoltageVsOCP	
AisCyclicVoltammetryElement, 92	