Squidstat API User Manual

Generated by Admiral Instruments LLC
April 1, 2025

1 Squidstat API User Manual	1
1.0.0.1 Where to go from here	1
2 Advanced Experiment Logic	3
2.0.1 Experiment Description	3
2.0.2 Building the Experiment	4
2.0.3 Starting the Experiment	5
3 Finding COM Ports	7
3.1 Windows	7
3.2 Mac	7
3.3 Linux	8
4 Extra Examples	9
4.1 Discussions	9
5 Updating Firmware	11
5.0.1 Introduction	11
5.0.2 Checking for and Updating Outdated Firmware	11
6 Guided Examples	15
6.1 About Qt	15
7 Manual Experiments	17
7.1 Manual Mode Commands	17
7.1.1 Start the Experiment	17
7.1.2 Manual Modes	18
7.1.3 Stop the Experiment	19
8 Running an Experiment	21
8.0.1 Setup	21
8.0.2 Building the experiment	22
8.0.3 Connecting to the Device	22
8.0.4 Starting Experiment	22
9 Installing with C++	25
9.1 Cloning the Repository	25
9.1.1 Mac ARM64 (Apple Silicon)	26
9.1.2 Linux	26
9.1.3 CMake Installation	26
9.1.3.1 Building the project	26
10 Installing the Python Wrapper	29
10.0.1 Mac ARM64 (Apple Silicon)	29
10.1 Linux	29
10.2 How To Use The SquidstatLibrary with Python	30

11 Handling Signals	31
11.0.1 Tracker Signals	31
11.0.2 Experiment Data Signals	31
11.0.3 Other Signals	32
11.0.4 Connecting Signals	32
11.0.5 References	33
12 Deprecated List	35
13 Topic Index	37
13.1 API Objects by Category	37
14 Class Index	39
14.1 All Classes	39
15 Topic Documentation	41
15.1 Experiment Elements	41
15.2 Instrument Control	42
15.3 Helpers	42
15.3.1 Enumeration Type Documentation	43
15.3.1.1 AisPulseType	43
16 Class Documentation	45
16.1 AisACData Struct Reference	45
16.1.1 Member Data Documentation	46
16.1.1.1 numberOfCycles	46
16.2 AisCompRange Class Reference	46
16.2.1 Constructor & Destructor Documentation	46
16.2.1.1 AisCompRange()	46
16.2.2 Member Function Documentation	47
16.2.2.1 getBandwidthIndex()	47
16.2.2.2 getCompRangeName()	47
16.2.2.3 getStabilityFactor()	47
16.2.2.4 setBandwidthIndex()	47
16.2.2.5 setCompRangeName()	48
16.2.2.6 setStabilityFactor()	48
16.3 AisConstantCurrentElement Class Reference	48
16.3.1 Constructor & Destructor Documentation	50
16.3.1.1 AisConstantCurrentElement()	50
16.3.2 Member Function Documentation	50
16.3.2.1 getApproxMaxCurrent()	50
16.3.2.2 getApproxMaxVoltage()	50
16.3.2.3 getCategory()	51
16.3.2.4 getCurrent()	51

16.3.2.5 getMaxCapacity()	51
16.3.2.6 getMaxDuration()	51
16.3.2.7 getMaxVoltage()	51
16.3.2.8 getMinSamplingVoltageDifference()	52
16.3.2.9 getMinVoltage()	52
16.3.2.10 getName()	52
16.3.2.11 getSamplingInterval()	52
16.3.2.12 isAutoRange()	53
16.3.2.13 isAutoVoltageRange()	53
16.3.2.14 setApproxMaxCurrent()	53
16.3.2.15 setApproxMaxVoltage()	53
16.3.2.16 setAutoRange()	53
16.3.2.17 setAutoVoltageRange()	53
16.3.2.18 setCurrent()	53
16.3.2.19 setMaxCapacity()	54
16.3.2.20 setMaxDuration()	54
16.3.2.21 setMaxVoltage()	54
16.3.2.22 setMinSamplingVoltageDifference()	54
16.3.2.23 setMinVoltage()	55
16.3.2.24 setSamplingInterval()	55
16.4 AisConstantPotElement Class Reference	55
16.4.1 Constructor & Destructor Documentation	57
16.4.1.1 AisConstantPotElement()	57
16.4.2 Member Function Documentation	57
16.4.2.1 getApproxMaxCurrent()	57
16.4.2.2 getCategory()	58
16.4.2.3 getMaxAbsoluteCurrent()	58
16.4.2.4 getMaxCapacity()	58
16.4.2.5 getMaxCurrent()	58
16.4.2.6 getMaxDuration()	59
16.4.2.7 getMinAbsoluteCurrent()	59
16.4.2.8 getMinCurrent()	59
16.4.2.9 getMindIdt()	59
16.4.2.10 getName()	60
16.4.2.11 getPotential()	60
16.4.2.12 getSamplingInterval()	60
16.4.2.13 getVoltageRange()	60
16.4.2.14 isAutoRange()	60
16.4.2.15 isVoltageVsOCP()	60
16.4.2.16 setApproxMaxCurrent()	60
16.4.2.17 setAutoRange()	61
16.4.2.18 setMaxAbsoluteCurrent()	61

16.4.2.19 setMaxCapacity()	61
16.4.2.20 setMaxCurrent()	61
16.4.2.21 setMaxDuration()	62
16.4.2.22 setMinAbsoluteCurrent()	62
16.4.2.23 setMinCurrent()	62
16.4.2.24 setMindldt()	63
16.4.2.25 setPotential()	63
16.4.2.26 setSamplingInterval()	63
16.4.2.27 setVoltageRange()	63
16.4.2.28 setVoltageVsOCP()	63
16.5 AisConstantPowerElement Class Reference	64
16.5.1 Constructor & Destructor Documentation	65
16.5.1.1 AisConstantPowerElement() [1/2]	65
16.5.1.2 AisConstantPowerElement() [2/2]	65
16.5.2 Member Function Documentation	66
16.5.2.1 getCategory()	66
16.5.2.2 getMaxCapacity()	66
16.5.2.3 getMaxCurrent()	66
16.5.2.4 getMaxDuration()	67
16.5.2.5 getMaxVoltage()	67
16.5.2.6 getMinCurrent()	67
16.5.2.7 getMinVoltage()	67
16.5.2.8 getName()	68
16.5.2.9 getPower()	68
16.5.2.10 getSamplingInterval()	68
16.5.2.11 isCharge()	68
16.5.2.12 isMaximumVoltageVsOCP()	68
16.5.2.13 isMinimumVoltageVsOCP()	69
16.5.2.14 setCharge()	69
16.5.2.15 setMaxCapacity()	69
16.5.2.16 setMaxCurrent()	69
16.5.2.17 setMaxDuration()	70
16.5.2.18 setMaximumVoltageVsOCP()	70
16.5.2.19 setMaxVoltage()	70
16.5.2.20 setMinCurrent()	70
16.5.2.21 setMinimumVoltageVsOCP()	71
16.5.2.22 setMinVoltage()	71
16.5.2.23 setPower()	71
16.5.2.24 setSamplingInterval()	71
16.6 AisConstantResistanceElement Class Reference	72
16.6.1 Constructor & Destructor Documentation	73
16.6.1.1 AicConstantRegistanceFlament/)	73

16.6.2 Member Function Documentation	73
16.6.2.1 getCategory()	73
16.6.2.2 getMaxCapacity()	74
16.6.2.3 getMaxCurrent()	74
16.6.2.4 getMaxDuration()	74
16.6.2.5 getMaxVoltage()	74
16.6.2.6 getMinCurrent()	75
16.6.2.7 getMinVoltage()	75
16.6.2.8 getName()	75
16.6.2.9 getResistance()	75
16.6.2.10 getSamplingInterval()	75
16.6.2.11 isMaximumVoltageVsOCP()	76
16.6.2.12 isMinimumVoltageVsOCP()	76
16.6.2.13 setMaxCapacity()	76
16.6.2.14 setMaxCurrent()	76
16.6.2.15 setMaxDuration()	77
16.6.2.16 setMaximumVoltageVsOCP()	77
16.6.2.17 setMaxVoltage()	77
16.6.2.18 setMinCurrent()	77
16.6.2.19 setMinimumVoltageVsOCP()	78
16.6.2.20 setMinVoltage()	78
16.6.2.21 setResistance()	78
16.6.2.22 setSamplingInterval()	78
16.7 AisCyclicVoltammetryElement Class Reference	79
16.7.1 Constructor & Destructor Documentation	81
16.7.1.1 AisCyclicVoltammetryElement()	81
16.7.2 Member Function Documentation	81
16.7.2.1 getAlphaFactor()	81
16.7.2.2 getApproxMaxCurrent()	81
16.7.2.3 getCategory()	82
16.7.2.4 getdEdt()	82
16.7.2.5 getEndVoltage()	82
16.7.2.6 getFirstVoltageLimit()	82
16.7.2.7 getName()	82
16.7.2.8 getNumberOfCycles()	83
16.7.2.9 getQuietTime()	83
16.7.2.10 getQuietTimeSamplingInterval()	83
16.7.2.11 getSamplingInterval()	83
16.7.2.12 getSecondVoltageLimit()	83
16.7.2.13 getStartVoltage()	84
16.7.2.14 isAutoRange()	84
16.7.2.15 isEndVoltageVsOCP()	84

	16.7.2.16 isFirstVoltageLimitVsOCP()	. 84
	16.7.2.17 isSecondVoltageLimitVsOCP()	. 85
	16.7.2.18 isStartVoltageVsOCP()	. 85
	16.7.2.19 setAlphaFactor()	. 85
	16.7.2.20 setApproxMaxCurrent()	. 85
	16.7.2.21 setAutoRange()	. 85
	16.7.2.22 setdEdt()	. 85
	16.7.2.23 setEndVoltage()	. 86
	16.7.2.24 setEndVoltageVsOCP()	. 86
	16.7.2.25 setFirstVoltageLimit()	. 86
	16.7.2.26 setFirstVoltageLimitVsOCP()	. 86
	16.7.2.27 setNumberOfCycles()	. 87
	16.7.2.28 setQuietTime()	. 87
	16.7.2.29 setQuietTimeSamplingInterval()	. 87
	16.7.2.30 setSamplingInterval()	. 87
	16.7.2.31 setSecondVoltageLimit()	. 87
	16.7.2.32 setSecondVoltageLimitVsOCP()	. 88
	16.7.2.33 setStartVoltage()	. 88
	16.7.2.34 setStartVoltageVsOCP()	. 88
16.8 AisDataN	Manipulator Class Reference	. 89
16.8.1 N	Member Function Documentation	. 89
	16.8.1.1 getBaseCurrent()	. 89
	16.8.1.2 getBaseVoltage()	. 90
	16.8.1.3 getFrequency()	. 90
	16.8.1.4 getPulseCurrent()	. 90
	16.8.1.5 getPulsePeriod()	. 90
	16.8.1.6 getPulseVoltage()	. 90
	16.8.1.7 getPulseWidth()	. 90
	16.8.1.8 isPulseCompleted()	. 91
	16.8.1.9 loadPrimaryData()	. 91
	16.8.1.10 setPulseType() [1/2]	. 91
	16.8.1.11 setPulseType() [2/2]	. 91
16.9 AisDCCu	urrentSweepElement Class Reference	. 92
16.9.1 C	Constructor & Destructor Documentation	. 93
	16.9.1.1 AisDCCurrentSweepElement()	. 93
16.9.2 N	Member Function Documentation	. 94
	16.9.2.1 getAlphaFactor()	. 94
	16.9.2.2 getCategory()	. 94
	16.9.2.3 getEndingCurrent()	. 94
	16.9.2.4 getMaxVoltage()	. 94
	16.9.2.5 getMinVoltage()	. 95
	16.9.2.6 getName()	. 95

16.9.2.7 getQuietTime()	95
16.9.2.8 getQuietTimeSamplingInterval()	95
16.9.2.9 getSamplingInterval()	95
16.9.2.10 getScanRate()	96
16.9.2.11 getStartingCurrent()	96
16.9.2.12 setAlphaFactor()	96
16.9.2.13 setEndingCurrent()	96
16.9.2.14 setMaxVoltage()	96
16.9.2.15 setMinVoltage()	97
16.9.2.16 setQuietTime()	97
16.9.2.17 setQuietTimeSamplingInterval()	97
16.9.2.18 setSamplingInterval()	97
16.9.2.19 setScanRate()	97
16.9.2.20 setStartingCurrent()	98
16.10 AisDCData Struct Reference	98
16.11 AisDCPotentialSweepElement Class Reference	99
16.11.1 Constructor & Destructor Documentation	100
16.11.1.1 AisDCPotentialSweepElement()	100
16.11.2 Member Function Documentation	100
16.11.2.1 getAlphaFactor()	100
16.11.2.2 getApproxMaxCurrent()	101
16.11.2.3 getCategory()	101
16.11.2.4 getEndingPot()	101
16.11.2.5 getMaxAbsoluteCurrent()	101
16.11.2.6 getMinAbsoluteCurrent()	102
16.11.2.7 getName()	102
16.11.2.8 getQuietTime()	102
16.11.2.9 getQuietTimeSamplingInterval()	102
16.11.2.10 getSamplingInterval()	102
16.11.2.11 getScanRate()	103
16.11.2.12 getStartingPot()	103
16.11.2.13 isAutoRange()	103
16.11.2.14 isEndVoltageVsOCP()	103
16.11.2.15 isStartVoltageVsOCP()	103
16.11.2.16 setAlphaFactor()	103
16.11.2.17 setApproxMaxCurrent()	104
16.11.2.18 setAutoRange()	104
16.11.2.19 setEndingPot()	104
16.11.2.20 setEndVoltageVsOCP()	104
16.11.2.21 setMaxAbsoluteCurrent()	105
16.11.2.22 setMinAbsoluteCurrent()	105
16.11.2.23 setQuietTime()	105

16.11.2.24 setQuietTimeSamplingInterval()	 105
16.11.2.25 setSamplingInterval()	 105
16.11.2.26 setScanRate()	 106
16.11.2.27 setStartingPot()	 106
16.11.2.28 setStartVoltageVsOCP()	 106
16.12 AisDeviceTracker Class Reference	 107
16.12.1 Member Function Documentation	 108
16.12.1.1 connectAllPluggedInDevices()	 108
16.12.1.2 connectToDeviceOnComPort()	 108
16.12.1.3 deviceDisconnected	 108
16.12.1.4 firmwareUpdateNotification	 109
16.12.1.5 getConnectedDevices()	 109
16.12.1.6 getInstrumentHandler()	 109
16.12.1.7 Instance()	 110
16.12.1.8 newDeviceConnected	 110
16.12.1.9 saveLogToFile()	 110
16.12.1.10 setLogFilePath()	 110
16.12.1.11 updateFirmwareOnAllAvailableDevices()	 111
16.12.1.12 updateFirmwareOnComPort()	 111
16.13 AisDiffPulseVoltammetryElement Class Reference	 112
16.13.1 Constructor & Destructor Documentation	 113
16.13.1.1 AisDiffPulseVoltammetryElement() [1/2]	 113
16.13.1.2 AisDiffPulseVoltammetryElement() [2/2]	 114
16.13.2 Member Function Documentation	 114
16.13.2.1 getAlphaFactor()	 114
16.13.2.2 getApproxMaxCurrent()	 114
16.13.2.3 getCategory()	 115
16.13.2.4 getEndVoltage()	 115
16.13.2.5 getName()	 115
16.13.2.6 getPulseHeight()	 115
16.13.2.7 getPulsePeriod()	 115
16.13.2.8 getPulseWidth()	 116
16.13.2.9 getQuietTime()	 116
16.13.2.10 getQuietTimeSamplingInterval()	 116
16.13.2.11 getStartVoltage()	 116
16.13.2.12 getVStep()	 116
16.13.2.13 isAutoRange()	 117
16.13.2.14 isEndVoltageVsOCP()	 117
16.13.2.15 isStartVoltageVsOCP()	 117
16.13.2.16 setAlphaFactor()	 117
16.13.2.17 setApproxMaxCurrent()	 118
16.13.2.18 setAutoRange()	 118

16.13.2.19 setEndVoltage()	118
16.13.2.20 setEndVoltageVsOCP()	118
16.13.2.21 setPulseHeight()	119
16.13.2.22 setPulsePeriod()	119
16.13.2.23 setPulseWidth()	119
16.13.2.24 setQuietTime()	119
16.13.2.25 setQuietTimeSamplingInterval()	120
16.13.2.26 setStartVoltage()	120
16.13.2.27 setStartVoltageVsOCP()	120
16.13.2.28 setVStep()	120
16.14 AisEISGalvanostaticElement Class Reference	121
16.14.1 Constructor & Destructor Documentation	122
16.14.1.1 AisEISGalvanostaticElement()	122
16.14.2 Member Function Documentation	122
16.14.2.1 getAmplitude()	122
16.14.2.2 getBiasCurrent()	123
16.14.2.3 getCategory()	123
16.14.2.4 getEndFreq()	123
16.14.2.5 getMinimumCycles()	123
16.14.2.6 getName()	123
16.14.2.7 getQuietTime()	123
16.14.2.8 getQuietTimeSamplingInterval()	124
16.14.2.9 getStartFreq()	124
16.14.2.10 getStepsPerDecade()	124
16.14.2.11 setAmplitude()	124
16.14.2.12 setBiasCurrent()	124
16.14.2.13 setEndFreq()	124
16.14.2.14 setMinimumCycles()	125
16.14.2.15 setQuietTime()	125
16.14.2.16 setQuietTimeSamplingInterval()	125
16.14.2.17 setStartFreq()	125
16.14.2.18 setStepsPerDecade()	125
16.15 AisEISPotentiostaticElement Class Reference	126
16.15.1 Constructor & Destructor Documentation	127
16.15.1.1 AisEISPotentiostaticElement()	127
16.15.2 Member Function Documentation	127
16.15.2.1 getAmplitude()	127
16.15.2.2 getBiasVoltage()	128
16.15.2.3 getCategory()	128
16.15.2.4 getEndFreq()	128
16.15.2.5 getMinimumCycles()	128
16.15.2.6 getName()	128

16.15.2.7 getQuietTime()	. 128
16.15.2.8 getQuietTimeSamplingInterval()	. 129
16.15.2.9 getStartFreq()	. 129
16.15.2.10 getStepsPerDecade()	. 129
16.15.2.11 isBiasVoltageVsOCP()	. 129
16.15.2.12 setAmplitude()	. 129
16.15.2.13 setBiasVoltage()	. 129
16.15.2.14 setBiasVoltageVsOCP()	. 130
16.15.2.15 setEndFreq()	. 130
16.15.2.16 setMinimumCycles()	. 130
16.15.2.17 setQuietTime()	. 130
16.15.2.18 setQuietTimeSamplingInterval()	. 130
16.15.2.19 setStartFreq()	. 131
16.15.2.20 setStepsPerDecade()	. 131
16.16 AisErrorCode Class Reference	. 131
16.16.1 Member Enumeration Documentation	. 132
16.16.1.1 ErrorCode	. 132
16.16.2 Member Function Documentation	. 134
16.16.2.1 message()	. 134
16.16.2.2 value()	. 134
16.17 AisExperiment Class Reference	. 134
16.17.1 Constructor & Destructor Documentation	. 135
16.17.1.1 AisExperiment()	. 135
16.17.2 Member Function Documentation	. 135
16.17.2.1 appendElement()	. 135
16.17.2.2 appendSubExperiment()	. 135
16.17.2.3 getCategory()	. 136
16.17.2.4 getDescription()	. 136
16.17.2.5 getExperimentName()	. 136
16.17.2.6 operator=()	. 136
16.17.2.7 setDescription()	. 137
16.17.2.8 setExperimentName()	. 137
16.18 AisExperimentNode Struct Reference	. 137
16.19 AisInstrumentHandler Class Reference	. 138
16.19.1 Member Function Documentation	. 140
16.19.1.1 activeACDataReady	. 140
16.19.1.2 activeDCDataReady	. 141
16.19.1.3 deviceError	. 141
16.19.1.4 eraseRecoverData()	. 142
16.19.1.5 experimentNewElementStarting	. 142
16.19.1.6 experimentPaused	. 142
16.19.1.7 experimentResumed	. 142

16.19.1.8 experimentStopped
16.19.1.9 getExperimentUTCStartTime()
16.19.1.10 getFreeChannels()
16.19.1.11 getLinkedChannels()
16.19.1.12 getManualModeCurrentRangeList()
16.19.1.13 getManualModeVoltageRangeList()
16.19.1.14 getNumberOfChannels()
16.19.1.15 groundFloatStateChanged
16.19.1.16 hasBipolarMode()
16.19.1.17 idleDCDataReady
16.19.1.18 isChannelBusy()
16.19.1.19 isChannelPaused()
16.19.1.20 pauseExperiment()
16.19.1.21 recoverDataErased
16.19.1.22 recoveryACDataReady
16.19.1.23 recoveryDCDataReady
16.19.1.24 resetChannelLimits()
16.19.1.25 resumeExperiment()
16.19.1.26 setBipolarLinkedChannels()
16.19.1.27 setChannelMaximumCurrent()
16.19.1.28 setChannelMaximumTemperature()
16.19.1.29 setChannelMaximumVoltage()
16.19.1.30 setChannelMinimumCurrent()
16.19.1.31 setChannelMinimumVoltage()
16.19.1.32 setCompRange()
16.19.1.33 setFanSpeedMaximum()
16.19.1.34 setFanSpeedVariable()
16.19.1.35 setIRComp()
16.19.1.36 setLinkedChannels()
16.19.1.37 setManualModeConstantCurrent()
16.19.1.38 setManualModeConstantVoltage() [1/2]
16.19.1.39 setManualModeConstantVoltage() [2/2]
16.19.1.40 setManualModeCurrentAutorange()
16.19.1.41 setManualModeCurrentRange()
16.19.1.42 setManualModeOCP()
16.19.1.43 setManualModeSamplingInterval()
16.19.1.44 setManualModeVoltageAutorange()
16.19.1.45 setManualModeVoltageRange()
16.19.1.46 skipExperimentStep()
16.19.1.47 startIdleSampling()
16.19.1.48 startManualExperiment()
16.19.1.49 startUploadedExperiment()

16.19.1.50 stopExperiment()	158
16.19.1.51 uploadExperimentToChannel() [1/2]	158
16.19.1.52 uploadExperimentToChannel() [2/2]	159
16.20 AisMottSchottkyElement Class Reference	160
16.20.1 Constructor & Destructor Documentation	161
16.20.1.1 AisMottSchottkyElement() [1/2]	161
16.20.1.2 AisMottSchottkyElement() [2/2]	162
16.20.2 Member Function Documentation	162
16.20.2.1 getAmplitude()	162
16.20.2.2 getCategory()	162
16.20.2.3 getEndFrequency()	162
16.20.2.4 getEndingPotential()	163
16.20.2.5 getMinCycles()	163
16.20.2.6 getName()	163
16.20.2.7 getQuietTime()	163
16.20.2.8 getQuietTimeSampInterval()	163
16.20.2.9 getStartFrequency()	163
16.20.2.10 getStartingPotential()	164
16.20.2.11 getStepQuietSampInterval()	164
16.20.2.12 getStepQuietTime()	164
16.20.2.13 getStepsPerDecade()	164
16.20.2.14 getVoltageStep()	164
16.20.2.15 isEndVoltageVsOCP()	164
16.20.2.16 isStartVoltageVsOCP()	165
16.20.2.17 operator=()	165
16.20.2.18 setAmplitude()	165
16.20.2.19 setEndFrequency()	165
16.20.2.20 setEndingPotential()	165
16.20.2.21 setEndVoltageVsOCP()	166
16.20.2.22 setMinCycles()	166
16.20.2.23 setQuietTime()	166
16.20.2.24 setQuietTimeSampInterval()	166
16.20.2.25 setStartFrequency()	166
16.20.2.26 setStartingPotential()	167
16.20.2.27 setStartVoltageVsOCP()	167
16.20.2.28 setStepQuietSampInterval()	167
16.20.2.29 setStepQuietTime()	167
16.20.2.30 setStepsPerDecade()	167
16.20.2.31 setVoltageStep()	168
16.21 AisNormalPulseVoltammetryElement Class Reference	168
16.21.1 Constructor & Destructor Documentation	170
16.21.1.1 AisNormalPulseVoltammetryFlement() [1/2]	170

16.21.1.2 AlsNormalPulse voltammetry Element() [2/2]	170
16.21.2 Member Function Documentation	170
16.21.2.1 getAlphaFactor()	170
16.21.2.2 getApproxMaxCurrent()	171
16.21.2.3 getCategory()	171
16.21.2.4 getEndVoltage()	171
16.21.2.5 getName()	171
16.21.2.6 getPulsePeriod()	171
16.21.2.7 getPulseWidth()	172
16.21.2.8 getQuietTime()	172
16.21.2.9 getQuietTimeSamplingInterval()	172
16.21.2.10 getStartVoltage()	172
16.21.2.11 getVStep()	172
16.21.2.12 isAutoRange()	173
16.21.2.13 isEndVoltageVsOCP()	173
16.21.2.14 isStartVoltageVsOCP()	173
16.21.2.15 setAlphaFactor()	173
16.21.2.16 setApproxMaxCurrent()	174
16.21.2.17 setAutoRange()	174
16.21.2.18 setEndVoltage()	174
16.21.2.19 setEndVoltageVsOCP()	174
16.21.2.20 setPulsePeriod()	175
16.21.2.21 setPulseWidth()	175
16.21.2.22 setQuietTime()	175
16.21.2.23 setQuietTimeSamplingInterval()	175
16.21.2.24 setStartVoltage()	176
16.21.2.25 setStartVoltageVsOCP()	176
16.21.2.26 setVStep()	176
16.22 AisOpenCircuitElement Class Reference	177
16.22.1 Constructor & Destructor Documentation	178
16.22.1.1 AisOpenCircuitElement()	178
16.22.2 Member Function Documentation	178
16.22.2.1 getApproxMaxVoltage()	178
16.22.2.2 getCategory()	178
16.22.2.3 getMaxDuration()	178
16.22.2.4 getMaxVoltage()	179
16.22.2.5 getMindVdt()	179
16.22.2.6 getMinVoltage()	179
16.22.2.7 getName()	179
16.22.2.8 getSamplingInterval()	180
16.22.2.9 isAutoVoltageRange()	180
16.22.2.10 setApproxMaxVoltage()	180

16.22.2.11 setAutoVoltageRange()	180
16.22.2.12 setMaxDuration()	180
16.22.2.13 setMaxVoltage()	180
16.22.2.14 setMindVdt()	181
16.22.2.15 setMinVoltage()	181
16.22.2.16 setSamplingInterval()	181
16.23 AisSquareWaveVoltammetryElement Class Reference	182
16.23.1 Constructor & Destructor Documentation	183
16.23.1.1 AisSquareWaveVoltammetryElement() [1/2]	183
16.23.1.2 AisSquareWaveVoltammetryElement() [2/2]	184
16.23.2 Member Function Documentation	184
16.23.2.1 getAlphaFactor()	184
16.23.2.2 getApproxMaxCurrent()	184
16.23.2.3 getCategory()	185
16.23.2.4 getEndVoltage()	185
16.23.2.5 getName()	185
16.23.2.6 getPulseAmp()	185
16.23.2.7 getPulseFreq()	185
16.23.2.8 getQuietTime()	186
16.23.2.9 getQuietTimeSamplingInterval()	186
16.23.2.10 getStartVoltage()	186
16.23.2.11 getVStep()	186
16.23.2.12 isAutoRange()	186
16.23.2.13 isEndVoltageVsOCP()	187
16.23.2.14 isStartVoltageVsOCP()	187
16.23.2.15 setAlphaFactor()	187
16.23.2.16 setApproxMaxCurrent()	187
16.23.2.17 setAutoRange()	188
16.23.2.18 setEndVoltage()	188
16.23.2.19 setEndVoltageVsOCP()	188
16.23.2.20 setPulseAmp()	188
16.23.2.21 setPulseFreq()	189
16.23.2.22 setQuietTime()	189
16.23.2.23 setQuietTimeSamplingInterval()	189
16.23.2.24 setStartVoltage()	189
16.23.2.25 setStartVoltageVsOCP()	189
16.23.2.26 setVStep()	190
16.24 AisStaircasePotentialVoltammetryElement Class Reference	190
16.24.1 Constructor & Destructor Documentation	192
16.24.1.1 AisStaircasePotentialVoltammetryElement() [1/2]	192
16.24.1.2 AisStaircasePotentialVoltammetryElement() [2/2]	192
16.24.2 Member Function Documentation	193

16.24.2.1 getApproxMaxCurrent()	93
16.24.2.2 getCategory()	93
16.24.2.3 getEndVoltage()	93
16.24.2.4 getFirstVoltageLimit()	93
16.24.2.5 getName()	93
16.24.2.6 getNumberOfCycles()	94
16.24.2.7 getQuietTime()	94
16.24.2.8 getQuietTimeSamplingInterval()	94
16.24.2.9 getSamplingInterval()	94
16.24.2.10 getSecondVoltageLimit()	94
16.24.2.11 getStartVoltage()	94
16.24.2.12 getStepDuration()	95
16.24.2.13 getStepSize()	95
16.24.2.14 isAutorange()	95
16.24.2.15 isEndVoltageVsOCP()	95
16.24.2.16 isFirstVoltageLimitVsOCP()	95
16.24.2.17 isSecondVoltageLimitVsOCP()	95
16.24.2.18 isStartVoltageVsOCP()	96
16.24.2.19 operator=()	96
16.24.2.20 setApproxMaxCurrent()	96
16.24.2.21 setEndVoltage()	96
16.24.2.22 setEndVoltageVsOCP()	96
16.24.2.23 setFirstVoltageLimit()	97
16.24.2.24 setFirstVoltageLimitVsOCP()	97
16.24.2.25 setNumberOfCycles()	97
16.24.2.26 setQuietTime()	97
16.24.2.27 setQuietTimeSamplingInterval()	97
16.24.2.28 setSamplingInterval()	98
16.24.2.29 setSecondVoltageLimit()	98
16.24.2.30 setSecondVoltageLimitVsOCP()	98
16.24.2.31 setStartVoltage()	98
16.24.2.32 setStartVoltageVsOCP()	98
16.24.2.33 setStepDuration()	99
16.24.2.34 setStepSize()	99
16.25 AisSteppedCurrentElement Class Reference	99
16.25.1 Constructor & Destructor Documentation	00
16.25.1.1 AisSteppedCurrentElement()	00
16.25.2 Member Function Documentation	01
16.25.2.1 getApproxMaxVoltage()	01
16.25.2.2 getApproxMinVoltage()	01
16.25.2.3 getCategory()	01
16.25.2.4 getEndCurrent()	02

16.25.2.5 getName()	12
16.25.2.6 getSamplingInterval()	12
16.25.2.7 getStartCurrent()	12
16.25.2.8 getStepDuration()	12
16.25.2.9 getStepSize()	12
16.25.2.10 setApproxMaxVoltage()	12
16.25.2.11 setApproxMinVoltage()	13
16.25.2.12 setEndCurrent()	13
16.25.2.13 setSamplingInterval()	13
16.25.2.14 setStartCurrent()	13
16.25.2.15 setStepDuration()	13
16.25.2.16 setStepSize()	14
16.26 AisSteppedVoltageElement Class Reference	14
16.26.1 Constructor & Destructor Documentation	15
16.26.1.1 AisSteppedVoltageElement() [1/2]	15
16.26.1.2 AisSteppedVoltageElement() [2/2]	16
16.26.2 Member Function Documentation	16
16.26.2.1 getApproxMaxCurrent()	16
16.26.2.2 getCategory()	16
16.26.2.3 getEndVoltage()	16
16.26.2.4 getName()	16
16.26.2.5 getSamplingInterval()	17
16.26.2.6 getStartVoltage()	17
16.26.2.7 getStepDuration()	17
16.26.2.8 getStepSize()	17
16.26.2.9 isAutoRange()	17
16.26.2.10 isEndVoltageVsOCP()	17
16.26.2.11 isStartVoltageVsOCP()	8
16.26.2.12 operator=()	8
16.26.2.13 setApproxMaxCurrent()	8
16.26.2.14 setEndVoltage()	8
16.26.2.15 setEndVoltageVsOCP()	8
16.26.2.16 setSamplingInterval()	19
16.26.2.17 setStartVoltage()	19
16.26.2.18 setStartVoltageVsOCP()	19
16.26.2.19 setStepDuration()	19
16.26.2.20 setStepSize()	19
17 Examples 21	1
17.1 advancedControlFlow.cpp	
17.2 advancedExperiment.cpp	
17.3 basicExperiment.cpp	
17.3 basicExperiment.cpp	4

Inc	dex	241
	17.21 writeinCSV.py	237
	17.20 tcpServer.py	235
	17.19 tcpClient.py	233
	17.18 pulseData.py	231
	17.17 nonblockingExperiment.py	230
	17.16 manualExperiment.py	229
	17.15 linkedChannels.py	228
	17.14 firmwareUpdate.py	227
	17.13 dataOutput.py	226
	17.12 basicExperiment.py	225
	17.11 async.py	224
	17.10 advancedExperiment.py	223
	17.9 pulseData.cpp	221
	17.8 nonblockingExperiment.cpp	220
	17.7 manualExperiment.cpp	218
	17.6 linkedChannels.cpp	217
	17.5 firmwareUpdate.cpp	217
	17.4 dataOutput.cpp	215

Squidstat API User Manual

The Admiral Instruments API provides you with more control of our potentiostats. Through asynchronous signals you can capture events such as experiments starting, stopping, and pausing to seamlessly integrate these controls into your pipeline and workflow automation.

Some common use cases include:

- · Chaining multiple experiments together for long term testing.
- · Manipulating experiment parameters based on the result of the previous experiment.
- · Starting and stopping an experiment when an external device detects a specific temperature.

You can find it on our GitHub repository including examples for C++ and Python.

This documentation will guide you through how to install and set up our API as well as provide you with examples with explanations to help support your workflow.

If you have any difficulty using our API, please reach out to us through the discussions page on our GitHub.

1.0.0.1 Where to go from here

- Install and Setup C++
- · Install and Setup Python
- Guided Examples
- API Objects

Advanced Experiment Logic

This example demonstrates how to create a custom experiment using loops and subexperiments for added complexity.

2.0.1 Experiment Description

The API allows us to build experiments with multiple elements, which can loop individually or as a group. Let's consider building an experimental sequence with the following structure:

- 1. Open Circuit Potential (OCP) element
- 2. Constant Potential Element (loops 4 times, increasing the voltage by 100 mV each loop)
- 3. Galvanostatic EIS element and OCP element (loops 3 times)
- 4. Constant Current element that loops 2 times

· C++

```
auto customExperiment = std::make_shared<AisExperiment>();
AisOpenCircuitElement ocpElement(1, 10);
success &= customExperiment->appendElement(ocpElement);
int voltage = 0;
for (int i = 0; i < 4; i++) {</pre>
   AisConstantPotElement cvElement (voltage, 0.1, 5);
    success &= customExperiment->appendElement(cvElement, 1);
    voltage += 0.1; // Adding 100 mV
AisExperiment eisSubExperiment;
AisEISGalvanostaticElement galvEISElement(10, 10000, 10, 0.01, 0.1);
AisOpenCircuitElement ocpElement2(1, 5);
success &= eisSubExperiment.appendElement(galvEISElement, 1);
success &= eisSubExperiment.appendElement(ocpElement2, 1);
success &= customExperiment->appendSubExperiment(eisSubExperiment, 3):
AisConstantCurrentElement ccElement(0.1, 1, 10);
success &= customExperiment->appendElement(ccElement, 2);
if (!success) {
    qDebug() « "Error building experiment";
    return 0;
```

• Python

```
customExperiment = AisExperiment()
# Step 1
ocpElement = AisOpenCircuitElement(1, 10)
success &= customExperiment.appendElement(ocpElement)
for i in range(0, 4):
    cvElement = AisConstantPotElement(voltage, 0.1, 5)
    success &= customExperiment.appendElement(cvElement, 1)
    voltage = voltage + 0.1
eisSubExperiment = AisExperiment()
galvEISElement = AisEISGalvanostaticElement(10, 10000, 10, 0.01, 0.1)
ocpElement2 = AisOpenCircuitElement(1, 5)
success &= eisSubExperiment.appendElement(galvEISElement, 1)
success &= eisSubExperiment.appendElement(ocpElement2, 1)
success &= customExperiment.appendSubExperiment(eisSubExperiment, 3)
ccElement = AisConstantCurrentElement(0.1, 1, 10)
success &= customExperiment.appendElement(ccElement, 2)
   print("Error building experiment")
    sys.exit()
```

Let's go over the code in more detail:

2.0.2 Building the Experiment

Here, we create the Open Circuit Element and add it to the experiment without specifying the number of loops, so it runs once.

```
    C++
        AisOpenCircuitElement ocpElement(1, 10);
        success &= customExperiment->appendElement(ocpElement);
    Python
        ocpElement = AisOpenCircuitElement(1, 10)
        success &= customExperiment.appendElement(ocpElement)
```

Next, we create the Constant Potential Elements, using a for loop to increment the voltage for each element, and then adding it to the created subExperiment.

```
'C++
    int voltage = 0;
    for (int i = 0; i < 4; i++) {
        AisConstantPotElement cvElement(voltage, 0.1, 5);
        success &= customExperiment->appendElement(cvElement, 1);
        voltage += 0.1; // Adding 100 mV
}

* Python
voltage = 0
for i in range(0, 4):
    cvElement = AisConstantPotElement(voltage, 0.1, 5)
    success &= customExperiment.appendElement(cvElement, 1)
    voltage = voltage + 0.1
```

For the looped EIS and OCP elements, we create another AisExperiment (eisSubExperiment), add the elements, and use AisExperiment::appendSubExperiment with 3 passed as the loop argument. This makes the two elements loop consecutively 3 times.

```
· C++
```

```
AisEISGalvanostaticElement galvEISElement (10, 10000, 10, 0.01, 0.1);
AisOpenCircuitElement ocpElement2(1, 5);

success &= eisSubExperiment.appendElement (galvEISElement, 1);
success &= eisSubExperiment.appendElement (ocpElement2, 1);

success &= customExperiment->appendSubExperiment (eisSubExperiment, 3);

• Python
eisSubExperiment = AisExperiment()

galvEISElement = AisEISGalvanostaticElement (10, 10000, 10, 0.01, 0.1)
ocpElement2 = AisOpenCircuitElement (1, 5)

success &= eisSubExperiment.appendElement (galvEISElement, 1)
success &= eisSubExperiment.appendElement (ocpElement2, 1)
success &= customExperiment.appendSubExperiment (eisSubExperiment, 3)
```

Lastly, we create the Constant Current Element, adding it to the main experiment using AisExperiment::appendElement with the loop argument set to 2. This loops the element twice at the end.

```
· C++
```

```
AisExperiment eisSubExperiment;

AisEISGalvanostaticElement galvEISElement (10, 10000, 10, 0.01, 0.1);
AisOpenCircuitElement ocpElement2(1, 5);

success &= eisSubExperiment.appendElement (galvEISElement, 1);
success &= eisSubExperiment.appendElement (ocpElement2, 1);

success &= customExperiment->appendSubExperiment (eisSubExperiment, 3);

* Python
eisSubExperiment = AisExperiment()

galvEISElement = AisEISGalvanostaticElement (10, 10000, 10, 0.01, 0.1)
ocpElement2 = AisOpenCircuitElement (1, 5)

success &= eisSubExperiment.appendElement (galvEISElement, 1)
success &= eisSubExperiment.appendElement (ocpElement2, 1)

success &= customExperiment.appendSubExperiment (eisSubExperiment, 3)
```

2.0.3 Starting the Experiment

The experiment is now ready and can be started like any other experiment.

• C++

```
QObject::connect(tracker, &AisDeviceTracker::newDeviceConnected, [=](const QString& deviceName) {
    qDebug() « "New Device Connected: " « deviceName;
    auto& handler = tracker->getInstrumentHandler(INSTRUMENT_NAME);
    connectSignals(handler);
    AisErrorCode error = handler.uploadExperimentToChannel(CHANNEL, customExperiment);
    if (error) {
         qDebug() « error.message();
         return 0;
    error = handler.startUploadedExperiment(CHANNEL);
    if (error) {
        qDebug() « error.message();
         return 0;
});
AisErrorCode error = tracker->connectToDeviceOnComPort(COMPORT);
if (error != error.Success) {
    qDebug() « error.message();
    return 0;
}
```

Python

```
def startExperiment():
    handler = tracker.getInstrumentHandler(INSTRUMENT_NAME)

    connectSignals(handler)

    error = handler.uploadExperimentToChannel(CHANNEL, customExperiment)
    if error.value() != AisErrorCode.Success:
        print(error.message())
        app.quit()

    error = handler.startUploadedExperiment(CHANNEL)
    if error.value() != AisErrorCode.Success:
        print(error.message())
        app.quit()

tracker.newDeviceConnected.connect(startExperiment)

error = tracker.connectToDeviceOnComPort(COMPORT)

if error.value() != AisErrorCode.Success:
    print(error.message())
    sys.exit()
```

See the full example here

Now that we have seen how create complex custom experiments, lets move on to the next example, where we will utilize manual mode operations to provide real-time control of the Squidstat.

Previous	Next
Handling Signals	Manual Experiments

Finding COM Ports

COM ports are used by the API to communicate with Squidstats. It is crucial to know which serial port the instrument is connected to in order to establish the correct communication pathway. Locate the serial port in the Squidstat User Interface (SUI) software in the "More Options" tab under "Device Information." If the SUI is not installed, you can determine the serial port using different methods based on your operating system

3.1 Windows

- 1. Connect the device via USB and power it on.
- 2. Open Device Manager using the search function (Windows key > then type "Device Manager") or through Control Panel via: Control Panel > Hardware and Sound > Devices and Printers > Device Manager
- 3. Expand 'Ports' to view the connected devices (e.g. USB Serial Port (COM3)). If multiple COM ports are listed, power cycle or disconnect the device to identify the correct one.
- 4. When referring to the serial port from the example above in the API, the format for the entry would be COM3.

3.2 Mac

- 1. Connect the device via USB and power it on.
- 2. Open Terminal via Applications > Utilities > Terminal or use Spotlight Search (Cmd + Space, then type "Terminal").
- 3. List serial devices by entering ls /dev/cu.* in Terminal.
- 4. Identify the USB serial port from the list.

Example output:

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

In this example, /dev/cu.usbmodem14201 and /dev/cu.usbserial-Admiral_1409 are the USB serial ports associated with the connected devices. If there are multiple entries, power cycle or disconnect the device and rerun the command.

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

8 Finding COM Ports

3.3 Linux

- 1. Connect the device via USB and power it on.
- 2. Open Terminal (Ctrl + Alt + T).
- 3. Run ls /dev | grep tty in Terminal
- 4. Identify the USB serial port. Look for the lines that refer to USB devices (e.g., "ttyACM0" or "ttyUSB1"). For multiple entries, power cycle or disconnect the device and rerun the command.
- 5. When referring to the serial port from the example above in the API, the format for the entry would be $tty \leftarrow ACM0$.

Extra Examples

The previous examples cover all the API features required to run most experiments.

Below are additional examples showcasing specific features and use cases of the API.

	C++	Python
Writing Data to File	Link	Link
Nonblocking Experiment	Link	Link
Cycler Linked Channels	Link	Link
Handle Pulse Data	Link	Link
Advanced Control Flow	Link	
Async Example		Link
Remote Squidstat Operation		Client & Server

4.1 Discussions

If you have any other questions or requests regarding the API, please refer to the Admiral Instruments discussions page. Here you can ask questions, share your ongoing projects, and engage in discussions with other users and the Admiral Instruments team.

10 Extra Examples

Updating Firmware

5.0.1 Introduction

Each API version includes the necessary firmware for all supported Squidstats to connect properly. When a new API is released, it may include updated firmware with new features or fixes. However, firmware updates are not automatic. They must be initiated manually. A Squidstat won't connect to the API until its firmware is compatible. Once updated, the firmware stays on the device until another update is performed. No separate files or tools are needed, and updates are only required when the API version changes.

This page explains how to detect and handle outdated firmware.

Note

The latest Squidstat API and Squidstat User Interface should always have matching firmware versions for all devices. If switching between them requires a firmware update, please verify that you have both the latest SUI and API.

5.0.2 Checking for and Updating Outdated Firmware

If AisDeviceTracker::connectToDeviceOnComPort returns the AisErrorCode::FirmwareNotSupported error code, a firmware update is required to use the device with the API. Here is an example of how to update the firmware in this scenario.

12 Updating Firmware

```
});
      // Attempt to connect to the device
      auto error = tracker->connectToDeviceOnComPort(COMPORT);
      if (error == AisErrorCode::FirmwareNotSupported) {
           error = tracker->updateFirmwareOnComPort(COMPORT);
           \ensuremath{//} Some other error occured
           if (error != AisErrorCode::Success) {
   qDebug() « "Error: " « error.message();
      } else if (error != AisErrorCode::Success) {
   qDebug() « "Error: " « error.message();
           qDebug() « "Device firmware is up to date";
      return a.exec();

    Python

  from PySide6.QtWidgets import QApplication
  from SquidstatPyLibrary import AisDeviceTracker from SquidstatPyLibrary import AisInstrumentHandler
  from SquidstatPyLibrary import AisErrorCode
  # Define relavant device information, for easy access
  COMPORT = "COM16"
  app = OApplication()
  tracker = AisDeviceTracker.Instance()
  def onProgressMessage(message):
      print(message)
      if message. contains ("firmware is updated"):
           app.quit()
  tracker.firmwareUpdateNotification.connect(onProgressMessage)
  # Attempt to connect to the device
  error = tracker.connectToDeviceOnComPort(COMPORT)
  if error.value() == AisErrorCode.FirmwareNotSupported:
      error = tracker.updateFirmwareOnComPort(COMPORT)
      # Some other error occured
      if error.value() != AisErrorCode.Success:
          print(f"Error: {error.message()}")
           sys.exit()
  elif error.value() != AisErrorCode.Success:
      print(f"Error: {error.message()}")
      sys.exit()
  else:
      print("Device is already up to date.")
  sys.exit(app.exec())
```

The AisDeviceTracker::firmwareUpdateNotification signal will display the progress of the firmware update as a percentage while the instrument is updating. Once the update is complete, this notification will send the message "Firmware is updated", signaling that the instrument can now be connected to the API.

 Python tracker.firmwareUpdateNotification.connect(onProgressMessage)

See the full example here

Note that we check the A AisDeviceTracker::updateF	sisErrorCode first befor irmwareOnAllAvailableI	re performing the f Devices all connect	irmware update. Al ed devices that requi	ternatively, you re an update.	can	use

14 Updating Firmware

Previous	Next
Guided Examples	Running an Experiment

Guided Examples

In this section, we will walk through examples to demonstrate how to connect to instruments, build experiments, and handle data effectively. Each walkthrough references examples for Python and C++. Full versions of these can be found in the Raw Example Code tab of the Reference section. You can use these examples to test out code and explore the functionality of the API.

- · Updating Firmware
- · Running an Experiment
- · Handling Signals
- · Advanced Experiment Logic
- Manual Experiments

See the Reference section for more examples and documentation

6.1 About Qt

The SquidstatLibrary relies heavily on Qt's signals and slots mechanism for asynchronous communication between the API and Squidstats. This feature allows the API to operate in a non-blocking manner, enabling Squidstat control without blocking the main thread of your application. This ensures that the API can manage Squidstat operations seamlessly, without hindering the execution of other parts of your program or any additional applications running alongside it.

The SquidstatLibrary relies heavily on Qt's signal and slot mechanism for asynchronous communication between the API and Squidstats. This feature allows the API to operate in a non-blocking manner, enabling Squidstat control without interrupting the main thread of your application. This is essential for applications that need to control Squidstats via the API while keeping the main application responsive.

Here are some useful links to Qt documentation:

- Signals and Slots
- Qt's Event Loop

16 Guided Examples

Previous	Next
Main	Updating Firmware

Manual Experiments

Here, we will cover how to operate a Squidstat in Manual Mode. The Squistat API provides a set of manual mode commands as functions, which allow the user to control the squidstat while an experiment is running. Which can be useful for for scenarios where the user needs to change parameters on the fly (i.e. in a GUI application). See the manual experiments tab in the Squidstat User Interface software for an example use case of this feature.

7.1 Manual Mode Commands

```
The following commands are provided by the AisInstrumentHandler to control the Squidstat.

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

AisErrorCode AisInstrumentHandler::setManualModeCCP(uint8_t channel) const;

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

AisErrorCode AisInstrumentHandler::setManualModeCurrentRange(uint8_t channel, int currentRangeIndex) const;

AisErrorCode AisInstrumentHandler::setManualModeCurrentAutorange(uint8_t channel) const;

AisErrorCode AisInstrumentHandler::setManualModeVoltageRange(uint8_t channel, int voltageRangeIndex) const;

AisErrorCode AisInstrumentHandler::setManualModeVoltageAutorange(uint8_t channel) const;

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

7.1.1 Start the Experiment

Before calling any of the above commands, the manual experiment must be started.

Now our experiment will run indefinitely, until AisInstrumentHandler::stopExperiment is called.

18 Manual Experiments

7.1.2 Manual Modes

The three main operational modes are OCP (Open Circuit Potential), Potentiostatic, and Galvanostatic, defined below:

- · OCP: Measures the open circuit potential of the system.
- · Potentiostatic: Maintains a constant potential while measuring current.
- Galvanostatic: Maintains a constant current while measuring potential.

Here is an example of how to set each at timed intervals after an experiment has started.

```
· C++
```

```
// In this section we connect singleshot QTimers to lambda functions that call our mode
changing functions.
          // These lambdas are called asynchronously when the QTimers expire.
         \ensuremath{//} 5 seconds after starting experiment, change to Constant Current at .1A
         QTimer::singleShot(5000, [=, &handler]() {
    qDebug() « "Switching to constant current at .1A";
              AisErrorCode error = handler.setManualModeConstantCurrent(CHANNEL, .1);
              if (error != AisErrorCode::Success) {
                   qDebug() « error.message();
         });
          // 15 seconds after starting experiment, change to Constant Voltage at 1V
         QTimer::singleShot(15000, [=, &handler]() {
    qDebug() « "Switching to constant voltage at 1V";
              AisErrorCode error = handler.setManualModeConstantVoltage(CHANNEL, 1);
              if (error != AisErrorCode::Success) {
                   gDebug() « error.message();
              }
          // 25 seconds after starting experiment, change back into Open Circuit Potential mode.
         QTimer::singleShot(25000, [=, &handler]() {
    qDebug() « "Switching to open circuit potential";
    AisErrorCode error = handler.setManualModeOCP(CHANNEL);
              if (error != AisErrorCode::Success) {
                   qDebug() « error.message();
         });
```

Python

```
# In this section we create wrapper functions for the manual mode changing functions.
# These wrappers are called asynchronously when singleshot QTimers expire.
# This function changes the instrument to Constant Current at .1A
def setConstantCurrent(channel):
    print("Switching to constant current at .1A")
    error = handler.setManualModeConstantCurrent(channel, .1)
    if error.value() != AisErrorCode.Success:
        print(error.message())
\# It is called 5 seconds after the experiment starts
QTimer.singleShot(5000, lambda:setConstantCurrent(CHANNEL))
# This function changes the instrument to Constant Voltage at 1V
def setConstantVoltage(channel):
    print("Switching to constant voltage at 1V")
error = handler.setManualModeConstantVoltage(channel, 1)
    if error.value() != AisErrorCode.Success:
       print(error.message())
# It is called 15 seconds after the experiment starts
QTimer.singleShot(15000, lambda:setConstantVoltage(CHANNEL))
# This function changes the instrument to Open Circuit Potential
def setOpenCircuit(channel):
    print ("Switching to open circuit potential")
    error = handler.setManualModeOCP(channel)
    if error.value() != AisErrorCode.Success:
        print(error.message())
# It is called 25 seconds after the experiment starts
QTimer.singleShot(25000, lambda:setOpenCircuit(CHANNEL))
```

Note: As a reminder, it is always a good idea to check the error code returned by each command to ensure successful execution. See AisErrorCode for more information.

7.1.3 Stop the Experiment

Stopping the manual experiment uses the same command as stopping a normal experiment (AisInstrumentHandler::stopExperiment). In this case, we use a SingleShot timer (C++| Python) to stop the experiment after 30 seconds.

```
'C++

// Stop the experiment after 30 seconds
   QTimer::singleShot(30000, [=, &handler]() {
        qDebug() « "Stopping experiment";
        if (handler.stopExperiment(CHANNEL) != AisErrorCode::Success) {
            qDebug() « error.message();
        }
    });

Python

# Stop experiment after 30 seconds
def stopExperiment(channel):
    print("Stopping experiment.")
    error = handler.stopExperiment(channel)
    if error.value() != AisErrorCode.Success:
        print(error.message())
```

QTimer.singleShot(30000, lambda:stopExperiment(CHANNEL))

See the full example here

Previous	Next
Advanced Experiment Logic	Extra Examples

20 Manual Experiments

Running an Experiment

The following example demonstrates how to set up and run a basic experiment using the API. This example simply outlines the steps needed to connect to a Squidstat, build an experiment, and start it. Notably, it does not cover handling and reporting data from the Squidstat, which will be addressed in the Example on handling signals.

8.0.1 Setup

To begin, we need to include all neccessary headers or libraries, and setup the Qt event loop. At this stage, you can also retrieve the AisDeviceTracker Instance.

```
· C++
  #include "AisDeviceTracker.h"
  #include "AisExperiment.h'
  #include "AisInstrumentHandler.h"
  #include "experiments/builder_elements/AisConstantPotElement.h"
  #include <QCoreApplication>
  #include <QDebug>
  // Define relevant device information, for easy access
  #define COMPORT "COM1"
  #define CHANNEL 0
  int main()
      char** test = nullptr;
      int args;
      QCoreApplication a(args, test);
      auto tracker = AisDeviceTracker::Instance();

    Python

  import sys
  from PySide6.QtWidgets import QApplication
  from SquidstatPyLibrary import AisDeviceTracker from SquidstatPyLibrary import AisExperiment, AisErrorCode from SquidstatPyLibrary import AisInstrumentHandler
  from SquidstatPyLibrary import AisConstantPotElement
  \mbox{\#} Define relavant device information, for easy access COMPORT = "COM16"
  CHANNEL = 0
  app = QApplication()
  tracker = AisDeviceTracker.Instance()
```

Reference: https://doc.qt.io/qt-5/qcoreapplication.html

8.0.2 Building the experiment

Next, create an element (e.g. AisConstantPotElement) and pass in the required parameters. Then, we will create the experiment object AisExperiment and add the element to it using AisExperiment::appendElement.

```
• C++
              Voltage = 1V, Sampling Interval = 1s, Duration = 30s
      AisConstantPotElement cvElement(1, 1, 30);
      auto customExperiment = std::make_shared<AisExperiment>();
      // Append the constant potential element, and tell it to execute that element 1 time
      customExperiment->appendElement(cvElement, 1);

    Python

  cvElement = AisConstantPotElement(1, 1, 30)
  # After this point, the experiment is empty, so we need to add some elements to it
  experiment = AisExperiment()
  # Append the constant potential element, and tell the experiment to execute that element 1 time
  success = experiment.appendElement(cvElement, 1)
  # Check if the element was added successfully
  if not success:
     print("Error adding element to experiment")
      app.quit()
```

Reference: https://en.cppreference.com/w/cpp/memory/shared_ptr

8.0.3 Connecting to the Device

Connect to the instrument by calling AisDeviceTracker::connectToDeviceOnComPort with the desired COM port to link the AisDeviceTracker to the device Alternatively, use AisDeviceTracker::connectAllPluggedInDevices to connect to all available devices.

```
    C++
    AisErrorCode error = tracker->connectToDeviceOnComPort(COMPORT);
    if (error != error.Success) {
        qDebug() « error.message();
        return 0;
    }

    Python
    error = tracker.connectToDeviceOnComPort(COMPORT)
    # Check if connection was successful
    if error.value() != AisErrorCode.Success:
        print(error.message())
        sys.exit()
```

Note: Any command to the instrument should always be checked for errors via the returned AisErrorCode

8.0.4 Starting Experiment

Finally, to start the experiment:

- 1. Grab the instrument handler from the device tracker after establishing a connection to the instrument.
 - See Next Example for more details on establishing connections
- 2. Upload the experiment to the desired channel and check for errors
- 3. Start the experiment, again checking for any errors

4. Run the QApplication event loop to allow all relevant signals to be processed (see the next example "Handling Signals")

```
· C++
```

```
auto& handler = tracker->getInstrumentHandler(deviceName);
          connectSignals(handler);
          AisErrorCode error = handler.uploadExperimentToChannel(CHANNEL, customExperiment);
          if (error)
              qDebug() « error.message();
              return;
          // Start the previously uploaded experiment on the same channel
          error = handler.startUploadedExperiment(CHANNEL);
          // Exit the application if there is any error starting the experiment
          if (error) {
             gDebug() « error.message();

    Python

      handler = tracker.getInstrumentHandler(deviceName)
      connectSignals (handler)
      error = handler.uploadExperimentToChannel(CHANNEL, experiment)
      # Exit the application if there is any error uploading experiment
      if error.value() != AisErrorCode.Success:
          print(error.message())
          app.quit()
     error = handler.startUploadedExperiment(CHANNEL)
      # Exit the application if there is any error starting experiment
      if error.value() != AisErrorCode.Success:
          print(error.message())
```

See the full example here

app.quit()

Now that we have covered the basics of connecting to a Squidstat, building an experiment, and starting it, we can proceed to the next example. It demonstrates how to combine these steps and handle the signals emitted by the Squidstat.

Previous	Next
Updating Firmware	Handling Signals

Installing with C++

You can download the C++ version of the Squidstat API from the Admiral Instruments GitHub page or clone the repository using Git on your machine.

9.1 Cloning the Repository

- 1. To clone the repository, you will need to install Git. Depending on your platform, you can download the Git tool from this link.
 - To verify if Git is properly installed, you can follow these steps:
 - 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. ex: git version 2.41.0.windows.1
- 2. Open a command prompt and navigate to the directory where you want the API to be. **Note:** A new folder will be created automatically after the next step.
- 3. Clone the API from the Git repository by typing the following command in the command prompt and pressing Enter.

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

The result in the command prompt should 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.
```

The AdmiralSquidstatAPI folder will now be created, and it contains all the files required to get started with the API using a compatible compiler. We will refer to this folder as the "Root Directory" of the API.

26 Installing with C++

9.1.1 Mac ARM64 (Apple Silicon)

If you are using a newer Mac with an ARM64 processor (M1, M2, etc.), you must perform additional steps to set up the Squidstat API.

- 1. Open the cloned repository in a Finder window and navigate down a level to ./SquidstatLibrary.
 - In this folder you should see Linux, mac, windows, and mac.tar.gz.
- 2. Delete the mac directory. It contains the library for Intel-based Mac processors.
- 3. Extract the tar file by double clicking mac.tar.gz in the Finder window.
 - You should see a new mac/ directory get automatically created.
- 4. At this point, your build environment is set up, and you can proceed normally.

9.1.2 Linux

If you are on a device running Linux, before using the API, run sudo ./SquidstatLibrary/ \leftarrow Linux/install_dependencies.sh from the root directory of the SquidstatAPI. This will install all dependencies and configuration files needed to run the API.

Note: This will run apt-get update && sudo apt-get upgrade -y which will update all packages currently installed on your device.

9.1.3 CMake Installation

The API uses CMake natively to build the project. Provided is a CMakeLists.txt file which you can include in your project to build the API.

- 1. To utilize the CMakeLists.txt file, you need to install CMake. You can download CMake from here.
 Note: When installing, be sure to add the CMake executable to your environment variables.
- 2. To verify if CMake is installed correctly, type cmake in the command prompt and press Enter.

9.1.3.1 Building the project

- 1. To use the API on Windows you must install MSVC64 through Visual Studio. For Mac and Linux install Clang.
- 2. Open a command prompt or terminal in the root directory of the API.
- 3. Run the following command to generate the build files for the compiler installed in your environment. cmake -B ./SquidstatLibrary/

Note: If you want to use a different build generator, use the $-\mathsf{G}$ option.

- 4. Build the project using the following command. This will compile all examples present in the Squidstat← Library directory.

 cmake --build ./build
- 5. You can now run any of the examples through their compiled executable. ex: ./build/examples/← FirmwareUpdateDemo/Debug/FirmwareUpdateDemo.exe

Previous	Next
Main	Finding Comports

28 Installing with C++

Installing the Python Wrapper

The Squidstat API provides a Python wrapper, which can be downloaded from the Admiral Instruments GitHub page. The Python wrapper gives access to all the functions of the Squidstat API in Python.

10.0.1 Mac ARM64 (Apple Silicon)

If you are using a newer Mac with an ARM64 processor (M1, M2, etc.), you must perform additional steps to set up the Squidstat API.

- 1. Open the cloned repository in a Finder window and navigate down a level to ./SquidstatLibrary.
 - In this folder you should see Linux, mac, windows, and mac.tar.gz.
- 2. Delete the mac directory. It contains the library for Intel-based Mac processors.
- 3. Extract the tar file by double clicking mac.tar.gz in the Finder window.
 - You should see a new mac/ directory get automatically created.
- 4. At this point, your build environment is set up, and you can proceed normally.

10.1 Linux

If you are on a device running Linux, before using the API, run sudo ./SquidstatLibrary/ \leftarrow Linux/install_dependencies.sh from the root directory of the SquidstatAPI. This will install all dependencies and configuration files needed to run the API.

Note: This will run apt-get update && sudo apt-get upgrade -y which will update all packages currently installed on your device.

10.2 How To Use The SquidstatLibrary with Python

- 1. To use the SquidstatPyLibrary, you need to install a Python version \geq 3.9 and \leq 3.12.
 - Visit the official Python website at https://www.python.org/downloads/.
 - Download the installer for the desired Python version (\geq 3.9 and \leq 3.12).
 - · Run the installer and follow the installation wizard's instructions.
 - Make sure to select the option to add Python to the system environment (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 the 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 Python virtual environment, you can create one by running the command: python -m venv VIRTUAL_ENVIRONMENT_NAME
 - Open the command prompt and activate the virtual environment by typing: ./VIRTUAL_← ENVIRONMENT_NAME/Scripts/activate.bat
- 4. Now you can proceed to install the SquidstatPyLibrary. 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 it by using the command python Experiment.py.

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

Check out the Guided Examples for help getting started.

Handling Signals

This example reviews the basic signals emitted by the AisDeviceTracker and AisInstrumentHandler, which notify users of instrument events, errors, experiment data, and progress. For simplicity, all signals in this example are connected to lambda functions. However, they can be connected to any slot function compatible with the signal's signature.

11.0.1 Tracker Signals

The AisDeviceTracker emits signals regarding device connection, disconnection, and firmware update status. The following example demonstrates how these signals are connected. For details on firmware update signals, refer to the Firmware Update Example.

11.0.2 Experiment Data Signals

The AisInstrumentHandler emits signals containing experiment data and progress information during active experiments. The signals AisInstrumentHandler::activeDCDataReady and AisInstrumentHandler::activeACDataReady provide relevant DC and AC, respectively, and should be handled for each experiment.

32 Handling Signals

11.0.3 Other Signals

The AisInstrumentHandler also emits other signals, such as notifications for the start of new elements, the end of an experiment, instrument errors, and more. Below are some examples:

```
· C++
         // Whenever a new node in the element starts, note: some Ais Elements contain multiple logical
 nodes
          // i.e AisCyclicVoltammatryElement contains 4 nodes for each linear segment of each cycle plus
 a quiet time node if enabled
          // So this lambda would be executed atleast 4 times for each cycle
 QObject::connect(&handler, &AisInstrumentHandler::experimentNewElementStarting, [=](uint8_t channel, const AisExperimentNode& info) {
             qDebug() « "New element starting: " « info.stepName;
         QObject::connect(&handler, &AisInstrumentHandler::experimentStopped, [=] (uint8_t channel,
 const QString& reason)
             qDebug() « "Experiment Stopped Signal " « channel « "Reason : " « reason;
         QObject::connect(&handler, &AisInstrumentHandler::deviceError, [=] (uint8_t channel, const
 QString& error) {
             qDebug() « "Device Error: " « error;

    Python

     # Whenever a new node in the element starts, note: some Ais Elements contain multiple logical
 nodes
     # i.e AisCyclicVoltammatryElement contains 4 nodes for each linear segment of each cycle plus a
  quiet time node if enabled
     # So this lambda would be executed atleast 4 times for each cycle
     handler.experimentNewElementStarting.connect(lambda channel, info: print(f"New element starting:
  {info.stepName}"))
     # Whenever an experiment completes or is manually stopped, this will execute
     handler.experimentStopped.connect(lambda channel, reason: (print(f"Experiment Stopped Signal
  {channel}, {reason}"), app.quit()))
     handler.deviceError.connect(lambda channel, error: print(f"Device Error: {error}"))
```

For othere signals emitted: AisInstrumentHandler signals

11.0.4 Connecting Signals

· C++

To connect data and handler signals to a valid AisInstrumentHandler object, ensure the connection to the instrument is fully established. This can be done by connecting signals when the AisDeviceTracker::newDeviceConnected signal is emitted.

Here, a lambda function is created to connect relevant signals to the handler when the device is connected:

// Whenever an experiment completes or is manually stopped, this will execute

```
QObject::connect(&handler, &AisInstrumentHandler::experimentStopped, [=] (uint8_t channel,
 const OString& reason)
              qDebug() « "Experiment Stopped Signal " « channel « "Reason : " « reason;
         QObject::connect(&handler, &AisInstrumentHandler::deviceError, [=] (uint8_t channel, const
 OString& error) {
             qDebug() « "Device Error: " « error;
     };

    Python

 def connectSignals(handler):
     handler.activeDCDataReady.connect(lambda channel, data: print(f"Timestamp: {data.timestamp}
 Current: {data.current} Voltage: {data.workingElectrodeVoltage} CE Voltage:
  {data.counterElectrodeVoltage}"))
     handler.activeACDataReady.connect(lambda channel, data: print(f"Timestamp: {data.timestamp}
 Frequency: {data.frequency} Absolute Impedance: {data.absoluteImpedance}"))
      # Whenever a new node in the element starts, note: some Ais Elements contain multiple logical
 nodes
      # i.e AisCyclicVoltammatryElement contains 4 nodes for each linear segment of each cycle plus a
 quiet time node if enabled
      # So this lambda would be executed atleast 4 times for each cycle
     handler.experimentNewElementStarting.connect(lambda channel, info: print(f"New element starting:
  {info.stepName}"))
      # Whenever an experiment completes or is manually stopped, this will execute
     handler.experimentStopped.connect(lambda channel, reason: (print(f"Experiment Stopped Signal
  {channel}, {reason}"), app.quit()))
     handler.deviceError.connect(lambda channel, error: print(f"Device Error: {error}"))
```

Then, call this function after connecting to the device but before starting the experiment:

```
• C++
```

```
auto& handler = tracker->getInstrumentHandler(deviceName);
          connectSignals(handler);
          AisErrorCode error = handler.uploadExperimentToChannel(CHANNEL, customExperiment);
          if (error) {
              qDebug() « error.message();
              return;
          \ensuremath{//} Start the previously uploaded experiment on the same channel
          error = handler.startUploadedExperiment(CHANNEL);
           // Exit the application if there is any error starting the experiment
           if (error) {
              qDebug() « error.message();
               return:

    Python

      handler = tracker.getInstrumentHandler(deviceName)
      connectSignals (handler)
      error = handler.uploadExperimentToChannel(CHANNEL, experiment)
      # Exit the application if there is any error uploading experiment
      if error.value() != AisErrorCode.Success:
          print(error.message())
          app.quit()
      error = handler.startUploadedExperiment(CHANNEL)
      # Exit the application if there is any error starting experiment
if error.value() != AisErrorCode.Success:
          print(error.message())
          app.quit()
```

See the full example here

11.0.5 References

For information on syntax and related concepts used in this example, refer to the following documentation links:

34 Handling Signals

• Signals and Slots: C++ | Python

• Lambda Functions C++ | Python

With all the relevant code to run an experiment in place, we can now explore how the API supports building more complex experiment logic.

Previous	Next
Running an Experiment	Advanced Experiment Logic

Deprecated List

Member AisDiffPulseVoltammetryElement::isAutoRange () const

This function is deprecated and no longer supports auto range for this element. Specify the current using setApproxMaxCurrent(). The device will determine the appropriate range based on the current value.

Member AisDiffPulseVoltammetryElement::setAutoRange ()

This function is deprecated. Use setApproxMaxCurrent() to specify the current range instead.

Member AisNormalPulseVoltammetryElement::isAutoRange () const

This function is deprecated and no longer supports auto range for this element. Specify the current using setApproxMaxCurrent(). The device will determine the appropriate range based on the current value.

Member AisNormalPulseVoltammetryElement::setAutoRange ()

This function is deprecated. Use setApproxMaxCurrent() to specify the current range instead.

Member AisSquareWaveVoltammetryElement::AisSquareWaveVoltammetryElement (double startVoltage, double endVoltage, double voltageStep, double pulseAmp, double pulseFrequency)

Use the constructor with the approxMaxCurrent parameter instead.

Member AisSquareWaveVoltammetryElement::isAutoRange () const

This function is deprecated and no longer supports auto range for this element. Specify the current using setApproxMaxCurrent(). The device will determine the appropriate range based on the current value.

Member AisSquareWaveVoltammetryElement::setAutoRange ()

This function is deprecated. Use setApproxMaxCurrent() to specify the current range instead.

36 Deprecated List

Topic Index

13.1 API Objects by Category

Objects in the SquidstatAPI are grouped into categories to make it easier to find related items. Click any of the categories below to see the list of all included objects.

Experiment Elements	41
Instrument Control	42
Helpers	42

38 Topic Index

Class Index

14.1 All Classes

All classes and structures in the SquidstatAPI in alphabetical order.

AisACData
A structure containing AC data collected from the instrument
AisCompRange
This class has advanced options controlling the device stability including the bandwidth index
and the stability factor
AisConstantCurrentElement
Experiment that simulates a constant current flow with more advance options for stopping the
experiment.
48
AisConstantPotElement
Experiment that simulates a constant applied voltage.
55
AisConstantPowerElement
This experiment simulates a constant power, charge or discharge".
64
AisConstantResistanceElement This element/everyiment simulates a constant resistance lead
This element/experiment simulates a constant resistance load.
AisCyclicVoltammetryElement
This experiment sweeps the potential of the working electrode back and forth between the first
voltage-limit and the second voltage-limit at a constant scan rate (dE/dt) for a specified num-
ber of cycles
AisDataManipulator
This class offers advanced control over pulse data collection and manipulation. It provides meth-
ods to manipulate AIS primary data for all three pulse voltammetry experiments types, namely
Differential Pulse Voltammetry (DPV), Square Wave Voltammetry (SWV), and Normal Pulse
Voltammetry (NPV)
AisDCCurrentSweepElement
This experiment performs a DC current sweep from the starting current to the ending current
which progresses linearly according to the scan rate
AisDCData
A structure containing DC data collected from the instrument
AisDCPotentialSweepElement
This experiment performs a DC potential sweep from the starting current to the ending current
which progresses linearly according to the scan rate

40 Class Index

AisDeviceTracker	
This class is used track device connections to the computer. It also provides instrument handlers specific to each connected device which provide control of the relevant device	107
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	112
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	
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	126
AisErrorCode	120
This class contains the possible error codes returned to the user when working with the API. Error codes can help diagnose issues such as invalid parameters, communication failures, or device malfunctions. By handling errors properly, you can ensure reliable operation of your experiments	131
AisExperiment	
This class is used to create custom experiments. A custom experiment contains one or more elements. Once you create elements and set their parameters, you can add them to the containe	r 134
AisExperimentNode	
A structure containing some information regarding the running element	137
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	138
AisMottSchottkyElement	
This class performs Mott-Schottky analysis on the working electrode for a specified range of potentials	160
AisNormalPulseVoltammetryElement This experiment holds the working electrode at a baseline potential during the quiet time, then applies a train of pulses, which increase in amplitude until the final potential is reached	168
AisOpenCircuitElement	
This experiment observes the open circuit potential of the working electrode for a specific period of time.	
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	400
potential step magnitude	182
AisStaircasePotentialVoltammetryElement	
AisStaircasePotentialVoltammetryElement class represents an element for staircase potential	100
voltammetry experiments. It inherits from AisAbstractElement	190
A class representing an experiment to apply the stepped current.	
199	
AisSteppedVoltageElement	
A class representing an experiment to apply the stepped volatge	204

Topic Documentation

15.1 Experiment Elements

Represents all elements used to create and configure experiments in the API.

Classes

class AisConstantCurrentElement

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

· class AisConstantPotElement

an experiment that simulates a constant applied voltage.

· class AisConstantPowerElement

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

class AisConstantResistanceElement

This element/experiment simulates a constant resistance load.

class AisCyclicVoltammetryElement

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

class AisDCCurrentSweepElement

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

· class AisDCPotentialSweepElement

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

· class 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.

· class 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.

· class AisEISPotentiostaticElement

42 Topic Documentation

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

· class AisNormalPulseVoltammetryElement

This experiment holds the working electrode at a **baseline potential** during the **quiet time**, then applies a train of pulses, which increase in amplitude until the **final potential** is reached.

class AisOpenCircuitElement

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

class 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.

· class AisStaircasePotentialVoltammetryElement

AisStaircasePotentialVoltammetryElement class represents an element for staircase potential voltammetry experiments. It inherits from AisAbstractElement.

• class AisSteppedCurrentElement

A class representing an experiment to apply the stepped current.

class AisMottSchottkyElement

This class performs Mott-Schottky analysis on the working electrode for a specified range of potentials.

class AisSteppedVoltageElement

A class representing an experiment to apply the stepped volatge.

15.2 Instrument Control

Classes used to communicate with and control connected devices.

Classes

class AisDeviceTracker

This class is used track device connections to the computer. It also provides instrument handlers specific to each connected device which provide control of the relevant device.

class AisErrorCode

This class contains the possible error codes returned to the user when working with the API. Error codes can help diagnose issues such as invalid parameters, communication failures, or device malfunctions. By handling errors properly, you can ensure reliable operation of your experiments.

· class AisExperiment

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

class 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.

15.3 Helpers

Miscellaneous objects used to represent device configurations as well as manipulate and hold data.

15.3 Helpers 43

Classes

class AisCompRange

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

struct AisDCData

A structure containing DC data collected from the instrument.

struct AisACData

A structure containing AC data collected from the instrument.

struct AisExperimentNode

A structure containing some information regarding the running element.

· class AisDataManipulator

This class offers advanced control over pulse data collection and manipulation. It provides methods to manipulate AIS primary data for all three pulse voltammetry experiments types, namely Differential Pulse Voltammetry (DPV), Square Wave Voltammetry (SWV), and Normal Pulse Voltammetry (NPV).

Enumerations

• enum AisPulseType { DifferentialPulse = 0 , NormalPulse = 1 , SquarewavePulse = 2 }

This enum represents different pulse types, it is intended to tell the AisDataManipulator class how to calculate various pulse experiment parameters.

15.3.1 Enumeration Type Documentation

15.3.1.1 AisPulseType

enum AisPulseType

Enumerator

NormalPulse	Corresponds to AisDiffPulseVoltammetryElement.
SquarewavePulse	Corresponds to AisNormPulseVoltammetryElement.

44 Topic Documentation

Class Documentation

16.1 AisACData Struct Reference

A structure containing AC data collected from the instrument.

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

46 Class Documentation

16.1.1 Member Data Documentation

16.1.1.1 numberOfCycles

```
double AisACData::numberOfCycles
```

Note

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

16.2 AisCompRange Class Reference

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

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

16.2.1 Constructor & Destructor Documentation

16.2.1.1 AisCompRange()

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

16.2.2 Member Function Documentation

16.2.2.1 getBandwidthIndex()

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

Returns

the set value for the bandwidth index.

See also

setBandwidthIndex

16.2.2.2 getCompRangeName()

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

Returns

the name set for the compensation range.

16.2.2.3 getStabilityFactor()

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

Returns

the value set for the stability factor.

16.2.2.4 setBandwidthIndex()

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.

48 Class Documentation

Parameters

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

16.2.2.5 setCompRangeName()

Parameters

16.2.2.6 setStabilityFactor()

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

Parameters

factor	the stability-factor value (0-10)
--------	-----------------------------------

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

· AisCompRange.h

16.3 AisConstantCurrentElement Class Reference

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

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

Public Member Functions

AisConstantCurrentElement (double current, double samplingInterval, double duration)

the constant current element constructor.

AisConstantCurrentElement (const AisConstantCurrentElement &)

copy constructor for the AisConstantCurrentElement object.

• AisConstantCurrentElement & operator= (const AisConstantCurrentElement &)

overload equal to operator for the AisConstantCurrentElement object.

QString getName () const override

get the name of the element.

QStringList getCategory () const override

get a list of applicable categories of the element.

double getCurrent () const

get the value set for the current.

void setCurrent (double current)

set the value for the current.

double getSamplingInterval () const

get how frequently we are sampling the data.

void setSamplingInterval (double samplingInterval)

set how frequently we are sampling the data.

double getMinSamplingVoltageDifference () const

get the minimum sampling voltage difference for reporting the data.

void setMinSamplingVoltageDifference (double minVoltageDifference)

set a minimum sampling voltage difference for reporting the voltage.

• double getMaxVoltage () const

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

void setMaxVoltage (double maxVoltage)

set a maximum voltage to stop the experiment.

double getMinVoltage () const

get the value set minimum for the voltage in volts.

void setMinVoltage (double minVoltage)

set a minimum voltage to stop the experiment.

double getMaxDuration () const

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

• void setMaxDuration (double maxDuration)

set the maximum duration for the experiment.

• double getMaxCapacity () const

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

void setMaxCapacity (double maxCapacity)

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

• bool isAutoRange () const

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

void setAutoRange ()

set to auto-select the current range.

double getApproxMaxCurrent () const

get the value set for the expected maximum current.

void setApproxMaxCurrent (double approxMaxCurrent)

set maximum current expected, for manual current range selection.

• bool isAutoVoltageRange () const

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

50 Class Documentation

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.

16.3.1 Constructor & Destructor Documentation

16.3.1.1 AisConstantCurrentElement()

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.

16.3.2 Member Function Documentation

16.3.2.1 getApproxMaxCurrent()

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

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.

16.3.2.2 getApproxMaxVoltage()

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

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.

16.3.2.3 getCategory()

QStringList AisConstantCurrentElement::getCategory () const [override]

Returns

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

16.3.2.4 getCurrent()

double AisConstantCurrentElement::getCurrent () const

Returns

the value for the current in Amps.

16.3.2.5 getMaxCapacity()

double AisConstantCurrentElement::getMaxCapacity () const

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.

16.3.2.6 getMaxDuration()

double AisConstantCurrentElement::getMaxDuration () const

Returns

the maximum duration for the experiment in seconds.

16.3.2.7 getMaxVoltage()

double AisConstantCurrentElement::getMaxVoltage () const

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

52 Class Documentation

16.3.2.8 getMinSamplingVoltageDifference()

double AisConstantCurrentElement::getMinSamplingVoltageDifference () const

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.

16.3.2.9 getMinVoltage()

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

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

16.3.2.10 getName()

QString AisConstantCurrentElement::getName () const [override]

Returns

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

16.3.2.11 getSamplingInterval()

double AisConstantCurrentElement::getSamplingInterval () const

Returns

the data sampling interval value in seconds.

16.3.2.12 isAutoRange()

bool AisConstantCurrentElement::isAutoRange () const

Returns

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

16.3.2.13 isAutoVoltageRange()

bool AisConstantCurrentElement::isAutoVoltageRange () const

Returns

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

16.3.2.14 setApproxMaxCurrent()

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

Parameters

16.3.2.15 setApproxMaxVoltage()

This 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.
------------------	--

16.3.2.16 setAutoRange()

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

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

16.3.2.17 setAutoVoltageRange()

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

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

16.3.2.18 setCurrent()

Parameters

mps.

16.3.2.19 setMaxCapacity()

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

Parameters

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

16.3.2.20 setMaxDuration()

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

Parameters

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

16.3.2.21 setMaxVoltage()

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

Parameters

m	naxVoltage	the maximum voltage value in volts at which the experiment will stop.
---	------------	---

16.3.2.22 setMinSamplingVoltageDifference()

```
\label{lement::setMinSamplingVoltageDifference} \mbox{ \begin{tabular}{ll} double ${\it minVoltageDifference}$) \end{tabular}} \label{lement::setMinSamplingVoltageDifference} \mbox{ \end{tabular}}
```

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

Note

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

Parameters

minVoltageDifference the minimum sampling voltage difference value in	volts.
---	--------

16.3.2.23 setMinVoltage()

This 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

16.3.2.24 setSamplingInterval()

```
\label{lement::setSamplingInterval} \mbox{ \begin{tabular}{ll} \label{lement::setSamplingInterval (} \mbox{ \end{tabular} double $\it samplingInterval) \end{tabular}}
```

Parameters

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

· AisConstantCurrentElement.h

16.4 AisConstantPotElement Class Reference

an experiment that simulates a constant applied voltage.

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

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 getMaxAbsoluteCurrent () const

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

void setMaxAbsoluteCurrent (double maxCurrent)

set the maximum value for the absolute current in Amps.

• 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 getMinAbsoluteCurrent () 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 setMinAbsoluteCurrent (double minCurrent)

set the minimum value for the absolute current in Amps.

double getMinCurrent () const

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

void setMinCurrent (double minCurrent)

set the minimum value for the absolute current in Amps.

double getMaxCapacity () const

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

void setMaxCapacity (double maxCapacity)

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

• double getMindIdt () const

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

void setMindIdt (double mindIdt)

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

• bool isAutoRange () const

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

void setAutoRange ()

set to auto-select the current range.

• double getApproxMaxCurrent () const

get the value set for the expected maximum current.

void setApproxMaxCurrent (double approxMaxCurrent)

set maximum current expected, for manual current range selection.

• int getVoltageRange () const

get the value set for the voltage range.

void setVoltageRange (int idx)

manually set the voltage control range.

16.4.1 Constructor & Destructor Documentation

16.4.1.1 AisConstantPotElement()

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.

16.4.2 Member Function Documentation

16.4.2.1 getApproxMaxCurrent()

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

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.

16.4.2.2 getCategory()

QStringList AisConstantPotElement::getCategory () const [override]

Returns

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

16.4.2.3 getMaxAbsoluteCurrent()

double AisConstantPotElement::getMaxAbsoluteCurrent () const

Returns

the maximum absolute current value in Amps.

Note

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

16.4.2.4 getMaxCapacity()

double AisConstantPotElement::getMaxCapacity () const

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.

16.4.2.5 getMaxCurrent()

double AisConstantPotElement::getMaxCurrent () const

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.

Attention

Deprecation Warning: This function may be modified or changed in a future version. Use getMaxAbsoluteCurrent instead.

16.4.2.6 getMaxDuration()

double AisConstantPotElement::getMaxDuration () const

Returns

the maximum duration for the experiment in seconds.

16.4.2.7 getMinAbsoluteCurrent()

double AisConstantPotElement::getMinAbsoluteCurrent () const

Returns

the minimum absolute current value in Amps.

Note

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

16.4.2.8 getMinCurrent()

double AisConstantPotElement::getMinCurrent () const

Returns

the minimum absolute current value in Amps.

Note

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

Attention

Deprecation Warning: This function may be modified or changed in a future version. Use getMinAbsolute ← Current instead.

16.4.2.9 getMindldt()

double AisConstantPotElement::getMindIdt () const

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.

16.4.2.10 getName()

QString AisConstantPotElement::getName () const [override]

Returns

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

16.4.2.11 getPotential()

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

Returns

the value set for the potential in volts.

16.4.2.12 getSamplingInterval()

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

Returns

the data sampling interval value in seconds.

16.4.2.13 getVoltageRange()

```
int AisConstantPotElement::getVoltageRange () const
```

Returns

the index set for the voltage range. (see AisInstrumentHandler::getManualModeVoltageRangeList())

16.4.2.14 isAutoRange()

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

Returns

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

16.4.2.15 isVoltageVsOCP()

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

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

16.4.2.16 setApproxMaxCurrent()

This 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.
------------------	---

16.4.2.17 setAutoRange()

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

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

16.4.2.18 setMaxAbsoluteCurrent()

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

Parameters

maxCurrent	the maximum absolute current value in Amps.
------------	---

16.4.2.19 setMaxCapacity()

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

Parameters

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

16.4.2.20 setMaxCurrent()

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

Parameters

maxCurrent	the maximum current value in Amps.
------------	------------------------------------

Attention

Deprecation Warning: This function may be modified or changed in a future version. Use setMaxAbsoluteCurrent instead

16.4.2.21 setMaxDuration()

This 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.
-------------	---

16.4.2.22 setMinAbsoluteCurrent()

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

Parameters

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

16.4.2.23 setMinCurrent()

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

Parameters

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

Attention

Deprecation Warning: This function may be modified or changed in a future version. Use setMinAbsolute ← Current instead.

16.4.2.24 setMindIdt()

This 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

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

16.4.2.25 setPotential()

Parameters

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

16.4.2.26 setSamplingInterval()

Parameters

16.4.2.27 setVoltageRange()

```
void AisConstantPotElement::setVoltageRange ( int idx)
```

This is an **optional** parameter. If this function is not called, the device will auto-select the voltage range by default.

Parameters

idx the corresponding voltage range index (see AisInstrumentHandler::getManualModeVoltageRangeList())

16.4.2.28 setVoltageVsOCP()

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

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

16.5 AisConstantPowerElement Class Reference

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

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

Public Member Functions

• AisConstantPowerElement (double power, double duration, double samplingInterval)

the constant power element constructor

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

the constant power element constructor that supports the isCharge parameter

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.

bool isMaximumVoltageVsOCP () const

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

void setMaximumVoltageVsOCP (bool vsOCP)

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

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.

bool isMinimumVoltageVsOCP () const

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

void setMinimumVoltageVsOCP (bool vsOCP)

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

• double getMaxCurrent () const

get the value set maximum for the current in amps.

void setMaxCurrent (double maxCurrent)

set a maximum current to stop the experiment.

double getMinCurrent () const

get the value set minimum for the current in amps.

void setMinCurrent (double maxCurrent)

set a minimum current 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.

16.5.1 Constructor & Destructor Documentation

16.5.1.1 AisConstantPowerElement() [1/2]

Parameters

power	the value set for the power in watts.
duration	the maximum duration for the experiment in seconds.
samplingInterval	the data sampling interval value in seconds.

16.5.1.2 AisConstantPowerElement() [2/2]

Parameters

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

Attention

Deprecation Warning: the isCharge parameter will be deprecated in a future version. Using the alternative constructor is highly recommended to avoid compilation errors in a future version. If this constructor is used, the sign of the power will be ignored and isCharge state will be used to determine it instead.

16.5.2 Member Function Documentation

16.5.2.1 getCategory()

QStringList AisConstantPowerElement::getCategory () const [override]

Returns

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

16.5.2.2 getMaxCapacity()

double AisConstantPowerElement::getMaxCapacity () const

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.

16.5.2.3 getMaxCurrent()

double AisConstantPowerElement::getMaxCurrent () const

Returns

the value set for the maximum current in amps.

Note

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

16.5.2.4 getMaxDuration()

double AisConstantPowerElement::getMaxDuration () const

Returns

the maximum duration for the experiment in seconds.

16.5.2.5 getMaxVoltage()

double AisConstantPowerElement::getMaxVoltage () const

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

16.5.2.6 getMinCurrent()

 $\verb|double AisConstantPowerElement::getMinCurrent () const$

Returns

the value set for the minimum current in amps.

Note

this is an optional parameter. If no value has been set, the default value is 0.

16.5.2.7 getMinVoltage()

double AisConstantPowerElement::getMinVoltage () const

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

16.5.2.8 getName()

QString AisConstantPowerElement::getName () const [override]

Returns

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

16.5.2.9 getPower()

double AisConstantPowerElement::getPower () const

Returns

the value set for the power in watts.

16.5.2.10 getSamplingInterval()

double AisConstantPowerElement::getSamplingInterval () const

Returns

the data sampling interval value in seconds.

16.5.2.11 isCharge()

bool AisConstantPowerElement::isCharge () const

Returns

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

16.5.2.12 isMaximumVoltageVsOCP()

bool AisConstantPowerElement::isMaximumVoltageVsOCP () const

Returns

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

See also

setVsOcp

16.5.2.13 isMinimumVoltageVsOCP()

```
bool AisConstantPowerElement::isMinimumVoltageVsOCP () const
```

Returns

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

See also

setVsOcp

16.5.2.14 setCharge()

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

Parameters

isCharge	if the given argument is true, the experiment will simulate charge and discharge if given false.
icomargo	I in the given digenient is tree, the experiment will enhance charge and disentage it given idios.

Attention

Deprecation Warning: setCharge will be deprecated in a future version. Avoid using this function, and instead set the power to a positive or negative value. If AisConstantPowerElement(bool, double, double, double) is used, you must use this function to set the charge/discharge state.

16.5.2.15 setMaxCapacity()

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

Parameters

maxCapacity the value to set for the cell maximum capacit

16.5.2.16 setMaxCurrent()

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

maxCurrent	the maximum current value in amps at which the experiment will stop.
	the maximum can can raise in ampe at miner the experiment in etcp.

16.5.2.17 setMaxDuration()

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 second

16.5.2.18 setMaximumVoltageVsOCP()

```
void AisConstantPowerElement::setMaximumVoltageVsOCP ( bool vsOCP)
```

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 maximum voltage to reference the open-circuit voltage and false to set
	against the reference terminal.

16.5.2.19 setMaxVoltage()

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

Parameters

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

16.5.2.20 setMinCurrent()

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

Parameters

16.5.2.21 setMinimumVoltageVsOCP()

```
\label{local_volume} \begin{tabular}{ll} void AisConstantPowerElement::setMinimumVoltageVsOCP ( \\ bool \ensuremath{\textit{vsOCP}}) \end{tabular}
```

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 minimum voltage to reference the open-circuit voltage and false to set	
	against the reference terminal.	

16.5.2.22 setMinVoltage()

Parameters

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

Note

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

16.5.2.23 setPower()

```
void AisConstantPowerElement::setPower ( \mbox{double } power) \label{eq:power}
```

Parameters

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

16.5.2.24 setSamplingInterval()

Parameters

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

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

· AisConstantPowerElement.h

16.6 AisConstantResistanceElement Class Reference

This element/experiment simulates a constant resistance load.

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

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.

· bool isMinimumVoltageVsOCP () const

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

void setMinimumVoltageVsOCP (bool vsOCP)

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

• double getMaxVoltage () const

get the value set maximum for the voltage in volts.

• void setMaxVoltage (double maxVoltage)

set a maximum voltage to stop the experiment.

• bool isMaximumVoltageVsOCP () const

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

void setMaximumVoltageVsOCP (bool vsOCP)

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

• double getMaxCurrent () const

get the value set maximum for the current in amps.

void setMaxCurrent (double maxCurrent)

set a maximum current to stop the experiment.

• double getMinCurrent () const

get the value set minimum for the current in amps.

void setMinCurrent (double maxCurrent)

set a minimum current 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.

16.6.1 Constructor & Destructor Documentation

16.6.1.1 AisConstantResistanceElement()

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.

16.6.2 Member Function Documentation

16.6.2.1 getCategory()

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

Returns

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

16.6.2.2 getMaxCapacity()

double AisConstantResistanceElement::getMaxCapacity () const

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.

16.6.2.3 getMaxCurrent()

double AisConstantResistanceElement::getMaxCurrent () const

Returns

the value set for the maximum current in amps.

Note

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

16.6.2.4 getMaxDuration()

double AisConstantResistanceElement::getMaxDuration () const

Returns

the maximum duration for the experiment in seconds.

16.6.2.5 getMaxVoltage()

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

Returns

the value set for the maximum voltage in volts.

Note

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

16.6.2.6 getMinCurrent()

double AisConstantResistanceElement::getMinCurrent () const

Returns

the value set for the minimum current in amps.

Note

this is an optional parameter. If no value has been set, the default value is 0.

16.6.2.7 getMinVoltage()

double AisConstantResistanceElement::getMinVoltage () const

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

16.6.2.8 getName()

QString AisConstantResistanceElement::getName () const [override]

Returns

The name of the element: "Constant Resistance".

16.6.2.9 getResistance()

double AisConstantResistanceElement::getResistance () const

Returns

the value in ohm of the load resistance.

16.6.2.10 getSamplingInterval()

Returns

the data sampling interval value in seconds.

16.6.2.11 isMaximumVoltageVsOCP()

bool AisConstantResistanceElement::isMaximumVoltageVsOCP () const

Returns

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

See also

setVsOcp

16.6.2.12 isMinimumVoltageVsOCP()

bool AisConstantResistanceElement::isMinimumVoltageVsOCP () const

Returns

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

See also

setVsOcp

16.6.2.13 setMaxCapacity()

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

Parameters

maxCapacity	the value to set for the cell maximum capacity.

16.6.2.14 setMaxCurrent()

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

16.6.2.15 setMaxDuration()

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.
-------------	---

16.6.2.16 setMaximumVoltageVsOCP()

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

16.6.2.17 setMaxVoltage()

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

Parameters

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

16.6.2.18 setMinCurrent()

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

Parameters

16.6.2.19 setMinimumVoltageVsOCP()

```
void AisConstantResistanceElement::setMinimumVoltageVsOCP ( bool\ vsOCP)
```

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 minimum voltage to reference the open-circuit voltage and false to set	
	against the reference terminal.	

16.6.2.20 setMinVoltage()

This 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 volt	s at which the experiment will stop.
--	--------------------------------------

16.6.2.21 setResistance()

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

Parameters

resistance	the value in ohm of the load resistance.
------------	--

16.6.2.22 setSamplingInterval()

Parameters

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

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

· AisConstantResistanceElement.h

16.7 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>
#include <AisCyclicVoltammetryElement.h>
```

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 getQuietTime () const

Gets the quiet time duration.

void setQuietTime (double quietTime)

Sets the quiet time duration.

double getQuietTimeSamplingInterval () const

gets the potential sampling interval.

void setQuietTimeSamplingInterval (double quietTimeSamplingInterval)

Sets the quiet time sampling interval.

• 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.

unsigned int getNumberOfCycles ()

get the value set for the number of cycles

void setNumberOfCycles (unsigned 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.

• double getAlphaFactor () const

Get the value set for the alpha factor.

• void setAlphaFactor (double alphaFactor)

alpha factor controls the percentage of data sampled during a given interval. Data will be averaged over the last n% of the sampling interval.

16.7.1 Constructor & Destructor Documentation

16.7.1.1 AisCyclicVoltammetryElement()

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.

16.7.2 Member Function Documentation

16.7.2.1 getAlphaFactor()

```
double AisCyclicVoltammetryElement::getAlphaFactor () const
```

Returns

The value for the alpha factor is represented as a percent between 0 and 100.

Note

If nothing is set, this function will return a default value of 75.

16.7.2.2 getApproxMaxCurrent()

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

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.

16.7.2.3 getCategory()

QStringList AisCyclicVoltammetryElement::getCategory () const [override]

Returns

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

16.7.2.4 getdEdt()

double AisCyclicVoltammetryElement::getdEdt () const

Returns

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

16.7.2.5 getEndVoltage()

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

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.

16.7.2.6 getFirstVoltageLimit()

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

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.

16.7.2.7 getName()

 ${\tt QString\ AisCyclicVoltammetryElement::getName\ ()\ const\ [override]}$

Returns

The name of the element: "Cyclic Voltammetry".

16.7.2.8 getNumberOfCycles()

 $unsigned \ int \ AisCyclicVoltammetry {\tt Element::} get {\tt NumberOfCycles} \ \ ()$

Returns

the number of cycles set.

16.7.2.9 getQuietTime()

double AisCyclicVoltammetryElement::getQuietTime () const

Returns

The quiet time duration in seconds.

16.7.2.10 getQuietTimeSamplingInterval()

double AisCyclicVoltammetryElement::getQuietTimeSamplingInterval () const

Returns

samplingInterval The quiet time sampling interval to set in seconds.

16.7.2.11 getSamplingInterval()

double AisCyclicVoltammetryElement::getSamplingInterval () const

Returns

the data sampling interval value in seconds.

16.7.2.12 getSecondVoltageLimit()

double AisCyclicVoltammetryElement::getSecondVoltageLimit () const

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.

16.7.2.13 getStartVoltage()

double AisCyclicVoltammetryElement::getStartVoltage () const

Returns

the value of the start voltage in volts

16.7.2.14 isAutoRange()

bool AisCyclicVoltammetryElement::isAutoRange () const

Returns

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

16.7.2.15 isEndVoltageVsOCP()

bool AisCyclicVoltammetryElement::isEndVoltageVsOCP () const

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

16.7.2.16 isFirstVoltageLimitVsOCP()

 $\verb|bool AisCyclicVoltammetryElement:: is First Voltage \verb|LimitVsOCP| () const|\\$

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.

16.7.2.17 isSecondVoltageLimitVsOCP()

bool AisCyclicVoltammetryElement::isSecondVoltageLimitVsOCP () const

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.

16.7.2.18 isStartVoltageVsOCP()

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

Returns

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

16.7.2.19 setAlphaFactor()

This is an **optional** parameter. If nothing is set, the device will use the default value of 75.

Parameters

alphaFactor	the value for the alphaFactor ranges from 0 to 100.

16.7.2.20 setApproxMaxCurrent()

This 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.
------------------	---

16.7.2.21 setAutoRange()

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

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

16.7.2.22 setdEdt()

Parameters

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

16.7.2.23 setEndVoltage()

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

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

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

16.7.2.24 setEndVoltageVsOCP()

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.

16.7.2.25 setFirstVoltageLimit()

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

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
```

16.7.2.26 setFirstVoltageLimitVsOCP()

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

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	1
	false otherwise.	

16.7.2.27 setNumberOfCycles()

Parameters

cycles	the number of cycles to set
--------	-----------------------------

16.7.2.28 setQuietTime()

Parameters

quietTime	The quiet time duration to set in seconds.
-----------	--

16.7.2.29 setQuietTimeSamplingInterval()

```
\label{lem:condition} \begin{tabular}{ll} void AisCyclicVoltammetryElement::setQuietTimeSamplingInterval ( \\ double & quietTimeSamplingInterval) \end{tabular}
```

Parameters

quietTimeSamplingInterval The quiet time	e sampling interval to set in seconds.
--	--

16.7.2.30 setSamplingInterval()

```
\label{lem:condition} \begin{tabular}{ll} void AisCyclicVoltammetryElement::setSamplingInterval ( \\ double $sampInterval) \end{tabular}
```

Parameters

16.7.2.31 setSecondVoltageLimit()

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

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

16.7.2.32 setSecondVoltageLimitVsOCP()

```
\label{lem:void} \begin{tabular}{ll} void AisCyclicVoltammetryElement::setSecondVoltageLimitVsOCP ( \\ bool secondVoltageLimitVsOCP) \end{tabular}
```

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.

16.7.2.33 setStartVoltage()

```
\begin{tabular}{ll} void AisCyclicVoltammetryElement::setStartVoltage ( \\ & double \ startVoltage) \end{tabular}
```

Parameters

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

16.7.2.34 setStartVoltageVsOCP()

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

16.8 AisDataManipulator Class Reference

This class offers advanced control over pulse data collection and manipulation. It provides methods to manipulate AIS primary data for all three pulse voltammetry experiments types, namely Differential Pulse Voltammetry (DPV), Square Wave Voltammetry (SWV), and Normal Pulse Voltammetry (NPV).

```
#include <AisDataManipulator.h>
#include <AisDataManipulator.h>
```

Public Member Functions

AisDataManipulator ()

Default constructor for AisDataManipulator class.

AisErrorCode setPulseType (AisPulseType type, double pulseWidth, double pulsePeriod)

Set pulse type with pulse width and pulse period.

AisErrorCode setPulseType (AisPulseType type, double frequency)

Set pulse type with frequency.

• double getPulseWidth () const

Get the pulse width.

• double getPulsePeriod () const

Get the pulse period.

• double getFrequency () const

Get the pulse frequency.

• bool isPulseCompleted () const

Check if the pulse is completed.

• double getBaseCurrent () const

Get the base current.

• double getPulseCurrent () const

Get the pulse current.

• double getBaseVoltage () const

Get the base voltage.

• double getPulseVoltage () const

Get the pulse voltage.

void loadPrimaryData (const AisDCData &data)

Load primary data from AisDCData object.

16.8.1 Member Function Documentation

16.8.1.1 getBaseCurrent()

```
double AisDataManipulator::getBaseCurrent () const
```

Returns

The base current.

16.8.1.2 getBaseVoltage()

double AisDataManipulator::getBaseVoltage () const

Returns

The base voltage.

16.8.1.3 getFrequency()

double AisDataManipulator::getFrequency () const

Returns

The pulse frequency.

16.8.1.4 getPulseCurrent()

double AisDataManipulator::getPulseCurrent () const

Returns

The pulse current.

16.8.1.5 getPulsePeriod()

double AisDataManipulator::getPulsePeriod () const

Returns

The pulse period.

16.8.1.6 getPulseVoltage()

double AisDataManipulator::getPulseVoltage () const

Returns

The pulse voltage.

16.8.1.7 getPulseWidth()

double AisDataManipulator::getPulseWidth () const

Returns

The pulse width.

16.8.1.8 isPulseCompleted()

```
\verb|bool AisDataManipulator:: isPulseCompleted () const|\\
```

Returns

True if the pulse is completed, false otherwise.

16.8.1.9 loadPrimaryData()

Parameters

data	The AisDCData object containing primary data.
------	---

16.8.1.10 setPulseType() [1/2]

Note

This function is usefull only for SquarewavePulse.

Parameters

type	The type of pulse.
frequency	The frequency of the pulse.

Returns

AisErrorCode::Success if pulse setting was successful. If not successful, possible returned errors are:

· AisErrorCode::FailedRequest

16.8.1.11 setPulseType() [2/2]

Note

This function is usefull only for DifferentialPulse and NormalPulse.

Parameters

type	The type of pulse.
pulseWidth	The width of the pulse.
pulsePeriod	The period of the pulse.

Returns

AisErrorCode::Success if pulse setting was successful. If not successful, possible returned errors are:

· AisErrorCode::FailedRequest

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

· AisDataManipulator.h

16.9 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>
#include <AisDCCurrentSweepElement.h>
```

Public Member Functions

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

the DC current sweep element.

AisDCCurrentSweepElement (const AisDCCurrentSweepElement &)

 $copy\ constructor\ for\ the\ {\it AisDCCurrentSweepElement}\ object.$

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 getQuietTime () const

Gets the quiet time duration.

void setQuietTime (double quietTime)

Sets the quiet time duration.

• double getQuietTimeSamplingInterval () const

gets the quiet time sampling interval.

void setQuietTimeSamplingInterval (double quietTimeSamplingInterval)

Sets the quiet time sampling interval.

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.

· double getAlphaFactor () const

Get the value set for the alpha factor.

void setAlphaFactor (double alphaFactor)

alpha factor controls the percentage of data sampled during a given interval. Data will be averaged over the last n% of the sampling interval.

16.9.1 Constructor & Destructor Documentation

16.9.1.1 AisDCCurrentSweepElement()

Parameters

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

16.9.2 Member Function Documentation

16.9.2.1 getAlphaFactor()

double AisDCCurrentSweepElement::getAlphaFactor () const

Returns

The value for the alpha factor is represented as a percent between 0 and 100.

Note

If nothing is set, this function will return a default value of 75.

16.9.2.2 getCategory()

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

Returns

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

16.9.2.3 getEndingCurrent()

 $\verb|double AisDCCurrentSweepElement::getEndingCurrent () const$

Returns

the value for the ending current in Amps.

16.9.2.4 getMaxVoltage()

double AisDCCurrentSweepElement::getMaxVoltage () const

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

16.9.2.5 getMinVoltage()

double AisDCCurrentSweepElement::getMinVoltage () const

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

16.9.2.6 getName()

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

Returns

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

16.9.2.7 getQuietTime()

 $\verb|double AisDCCurrentSweepElement::getQuietTime () const|\\$

Returns

The quiet time duration in seconds.

16.9.2.8 getQuietTimeSamplingInterval()

```
double AisDCCurrentSweepElement::getQuietTimeSamplingInterval () const
```

Returns

samplingInterval The quiet time sampling interval to set in seconds.

16.9.2.9 getSamplingInterval()

 $\verb|double AisDCCurrentSweepElement::getSamplingInterval () | const|\\$

Returns

the data sampling interval value in seconds.

16.9.2.10 getScanRate()

double AisDCCurrentSweepElement::getScanRate () const

Returns

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

See also

setScanRate

16.9.2.11 getStartingCurrent()

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

Returns

the value set for the constant current in Amps.

16.9.2.12 setAlphaFactor()

This is an **optional** parameter. If nothing is set, the device will use the default value of 75.

Parameters

	alphaFactor	the value for the alphaFactor ranges from 0 to 100.
--	-------------	---

16.9.2.13 setEndingCurrent()

Parameters

endingCurrent	the value for the ending current in Amps
---------------	--

16.9.2.14 setMaxVoltage()

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

Parameters

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

16.9.2.15 setMinVoltage()

This 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.
------------	---

16.9.2.16 setQuietTime()

```
\begin{tabular}{ll} \begin{tabular}{ll} void AisDCCurrentSweepElement::setQuietTime ( \\ \begin{tabular}{ll} double & quietTime) \end{tabular}
```

Parameters

quietTime	The quiet time duration to set in seconds.
-----------	--

16.9.2.17 setQuietTimeSamplingInterval()

```
\label{local_problem} \begin{tabular}{ll} void $AisDCCurrentSweepElement::setQuietTimeSamplingInterval ( \\  & double $quietTimeSamplingInterval) \end{tabular}
```

Parameters

quietTimeSamplingInterval	The quiet time sampling interval to set in seconds.	ı

16.9.2.18 setSamplingInterval()

```
\label{local_continuity} \mbox{void AisDCCurrentSweepElement::setSamplingInterval (} \\ \mbox{double } samplingInterval)
```

Parameters

samplingInterval the	data sampling interva	I value in seconds.
----------------------	-----------------------	---------------------

16.9.2.19 setScanRate()

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

Parameters

scanRate	the value to set for the scan rate.

16.9.2.20 setStartingCurrent()

```
\begin{tabular}{ll} \begin{tabular}{ll} void AisDCCurrentSweepElement::setStartingCurrent ( \\ & double \begin{tabular}{ll} doub
```

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

16.10 AisDCData Struct Reference

A structure containing DC data collected from the instrument.

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

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

· AisDataPoints.h

16.11 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>
#include <AisDCPotentialSweepElement.h>
```

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 getQuietTime () const

Gets the quiet time duration.

void setQuietTime (double quietTime)

Sets the quiet time duration.

double getQuietTimeSamplingInterval () const

gets the quiet time sampling interval.

void setQuietTimeSamplingInterval (double quietTimeSamplingInterval)

Sets the quiet time sampling interval.

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.

double getMaxAbsoluteCurrent () const

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

void setMaxAbsoluteCurrent (double maxCurrent)

set a maximum Current to stop the experiment.

double getMinAbsoluteCurrent () const

get the value set minimum for the Current in amps.

void setMinAbsoluteCurrent (double minCurrent)

set a minimum Current to stop the experiment.

· double getAlphaFactor () const

Get the value set for the alpha factor.

void setAlphaFactor (double alphaFactor)

alpha factor controls the percentage of data sampled during a given interval. Data will be averaged over the last n% of the sampling interval.

16.11.1 Constructor & Destructor Documentation

16.11.1.1 AisDCPotentialSweepElement()

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.

16.11.2 Member Function Documentation

16.11.2.1 getAlphaFactor()

double AisDCPotentialSweepElement::getAlphaFactor () const

Returns

The value for the alpha factor is represented as a percent between 0 and 100.

Note

If nothing is set, this function will return a default value of 75.

16.11.2.2 getApproxMaxCurrent()

double AisDCPotentialSweepElement::getApproxMaxCurrent () const

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.

16.11.2.3 getCategory()

QStringList AisDCPotentialSweepElement::getCategory () const [override]

Returns

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

16.11.2.4 getEndingPot()

 $\verb|double AisDCPotentialSweepElement::getEndingPot () const$

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

Returns

the value set for the ending voltage in volts.

16.11.2.5 getMaxAbsoluteCurrent()

double AisDCPotentialSweepElement::getMaxAbsoluteCurrent () const

Returns

the value set for the maximum Current.

Note

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

16.11.2.6 getMinAbsoluteCurrent()

double AisDCPotentialSweepElement::getMinAbsoluteCurrent () const

Returns

the value set for the minimum Current in amps.

Note

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

16.11.2.7 getName()

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

Returns

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

16.11.2.8 getQuietTime()

```
double AisDCPotentialSweepElement::getQuietTime () const
```

Returns

The quiet time duration in seconds.

16.11.2.9 getQuietTimeSamplingInterval()

```
\verb|double AisDCP| otential SweepElement:: getQuietTimeSamplingInterval () constant of the con
```

Returns

samplingInterval The quiet time sampling interval to set in seconds.

16.11.2.10 getSamplingInterval()

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

Returns

the data sampling interval value in seconds.

16.11.2.11 getScanRate()

double AisDCPotentialSweepElement::getScanRate () const

Returns

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

See also

setScanRate

16.11.2.12 getStartingPot()

double AisDCPotentialSweepElement::getStartingPot () const

Returns

the value of the starting potential in volts.

16.11.2.13 isAutoRange()

bool AisDCPotentialSweepElement::isAutoRange () const

Returns

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

16.11.2.14 isEndVoltageVsOCP()

 $\verb|bool AisDCP| otentialSweepElement:: is \verb|EndVoltageVsOCP| () const| \\$

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

16.11.2.15 isStartVoltageVsOCP()

 $\verb|bool AisDCP| otentialSweepElement:: is StartVoltageVsOCP () const$

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

16.11.2.16 setAlphaFactor()

This is an **optional** parameter. If nothing is set, the device will use the default value of 75.

Parameters

alphaFactor	the value for the alphaFactor ranges from 0 to 100.
-------------	---

16.11.2.17 setApproxMaxCurrent()

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

Parameters

16.11.2.18 setAutoRange()

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

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

16.11.2.19 setEndingPot()

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.
-----------------	---

16.11.2.20 setEndVoltageVsOCP()

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.

16.11.2.21 setMaxAbsoluteCurrent()

This 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 with the harware current limitation.

Parameters

maxCurrent	the maximum Current value in volts at which the experiment will stop.
------------	---

16.11.2.22 setMinAbsoluteCurrent()

This 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 voltage is above that value.

Parameters

inCurrent the minimum Current value in volts at wh	nich the experiment will stop.
--	--------------------------------

16.11.2.23 setQuietTime()

```
void AisDCPotentialSweepElement::setQuietTime ( \label{eq:constraint} \mbox{double } quietTime)
```

Parameters

16.11.2.24 setQuietTimeSamplingInterval()

```
\label{lem:condition} \begin{tabular}{ll} void AisDCP otential Sweep Element:: set Quiet Time Sampling Interval ( \\ double & quiet Time Sampling Interval) \end{tabular}
```

Parameters

16.11.2.25 setSamplingInterval()

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

Parameters

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

16.11.2.26 setScanRate()

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.	
--	--

16.11.2.27 setStartingPot()

```
\begin{tabular}{ll} \beg
```

Parameters

16.11.2.28 setStartVoltageVsOCP()

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

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	1
	against the reference terminal.	

Note

by default, this is set to false.

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

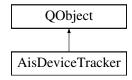
• AisDCPotentialSweepElement.h

16.12 AisDeviceTracker Class Reference

This class is used track device connections to the computer. It also provides instrument handlers specific to each connected device which provide control of the relevant device.

```
#include <AisDeviceTracker.h>
#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)
 - a signal which is emitted regularaly during a firmware update, providing information about the progress of the update.

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 (const QString &comport) const
 - update firmware on connected device at USB port.
- int updateFirmwareOnAllAvailableDevices ()
 - request firmware update for all available devices.
- void saveLogToFile (bool save)
 - Allow to collect device error message in file for debugging purpose.
- void setLogFilePath (const QString &path)
 - This will help to change the log file directory.

Static Public Member Functions

• static AisDeviceTracker * Instance ()

get the instance of the device tracker.

16.12.1 Member Function Documentation

16.12.1.1 connectAllPluggedInDevices()

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

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.

16.12.1.2 connectToDeviceOnComPort()

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".

16.12.1.3 deviceDisconnected

Parameters

deviceName	the name of the newly disconnected device.
------------	--

Examples

advancedControlFlow.cpp, basicExperiment.cpp, and pulseData.cpp.

16.12.1.4 firmwareUpdateNotification

Parameters

message	a string containing the progress percentage message.
	l a an ma a a man ma a ma b a a a a a b a a a a ma a a ma a a a

Examples

firmwareUpdate.cpp.

16.12.1.5 getConnectedDevices()

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

Returns

a list of all the connected devices.

16.12.1.6 getInstrumentHandler()

Parameters

deviceName the name of the connected device to get the instrument hand
--

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.

See also

AisInstrumentHandler

AisdeviceTracker::connectToDeviceOnComPort()
AisdeviceTracker::getConnectedDevices()

16.12.1.7 Instance()

```
static AisDeviceTracker * AisDeviceTracker::Instance () [static]
```

Returns

the static instance of the AisDeviceTracker

Examples

advancedControlFlow.cpp, advancedExperiment.cpp, basicExperiment.cpp, dataOutput.cpp, firmwareUpdate.cpp, linkedChannels.cpp, manualExperiment.cpp, nonblockingExperiment.cpp, and pulseData.cpp.

16.12.1.8 newDeviceConnected

Parameters

deviceName	the name of the newly connected device.
------------	---

Note

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

Examples

advancedControlFlow.cpp, advancedExperiment.cpp, basicExperiment.cpp, dataOutput.cpp, linkedChannels.cpp, manualExperiment.cpp, nonblockingExperiment.cpp, and pulseData.cpp.

16.12.1.9 saveLogToFile()

Note

by default it will be true.

Parameters

save	When set to 'false,' it will not write logs to the file. When set to 'true,' it will begin writing device error logs
	to the file.

See also

setLogFilePath

16.12.1.10 setLogFilePath()

Note

by default it will be Document/Admiral Instrument/API

Parameters

path Set the path value at which you want to save the log file.

Note

If you set 'false' for 'saveLogToFile,' it will not generate the log file. It is recommended to set it to 'true' or leave the permission as the default setting.

See also

saveLogToFile

16.12.1.11 updateFirmwareOnAllAvailableDevices()

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

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

16.12.1.12 updateFirmwareOnComPort()

Parameters

comport the communication port to connect through.

Returns

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

- · AisErrorCode::FirmwareUptodate
- · AisErrorCode::ConnectionFailed

Note

emits firmwareUpdateNotification() signal to provide firmware update progress.

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

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

· AisDeviceTracker.h

16.13 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>
#include <AisDiffPulseVoltammetryElement.h>
```

Public Member Functions

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

the differential pulse element constructor.

 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 getQuietTime () const

Gets the quiet time duration.

void setQuietTime (double quietTime)

Sets the quiet time duration.

• double getQuietTimeSamplingInterval () const

gets the quiet time sampling interval.

void setQuietTimeSamplingInterval (double quietTimeSamplingInterval)

Sets the quiet time sampling interval.

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.

• double getAlphaFactor () const

Get the value set for the alpha factor.

void setAlphaFactor (double alphaFactor)

alpha factor controls the percentage of data sampled during a given interval. Data will be averaged over the last n% of the sampling interval.

16.13.1 Constructor & Destructor Documentation

16.13.1.1 AisDiffPulseVoltammetryElement() [1/2]

Parameters

startVoltage	the value of the starting potential in volts
endVoltage	the value of the ending potential in volts
voltageStep	the value set for the voltage step in volts.
pulseHeight	the value for the pulse height in volts.
pulseWidth	the value for the pulse width in seconds.
pulsePeriod	the value for the pulse period in seconds.
approxMaxCurrent	the value for the approximate maximum current in amperes.

16.13.1.2 AisDiffPulseVoltammetryElement() [2/2]

Parameters

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

16.13.2 Member Function Documentation

16.13.2.1 getAlphaFactor()

```
double AisDiffPulseVoltammetryElement::getAlphaFactor () const
```

Returns

The value for the alpha factor is represented as a percent between 0 and 100.

Note

If nothing is set, this function will return a default value of 75.

16.13.2.2 getApproxMaxCurrent()

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

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.

16.13.2.3 getCategory()

QStringList AisDiffPulseVoltammetryElement::getCategory () const [override]

Returns

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

16.13.2.4 getEndVoltage()

double AisDiffPulseVoltammetryElement::getEndVoltage () const

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

Returns

the value set for the ending voltage in volts.

16.13.2.5 getName()

QString AisDiffPulseVoltammetryElement::getName () const [override]

Returns

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

16.13.2.6 getPulseHeight()

double AisDiffPulseVoltammetryElement::getPulseHeight () const

Returns

the value set for the pulse height in volts.

See also

setPulseHeight

16.13.2.7 getPulsePeriod()

double AisDiffPulseVoltammetryElement::getPulsePeriod () const

Returns

the value set for the pulse period in seconds.

See also

setPulsePeriod

Examples

pulseData.cpp.

16.13.2.8 getPulseWidth()

double AisDiffPulseVoltammetryElement::getPulseWidth () const

Returns

the value set for the pulse width in seconds.

See also

setPulseWidth

Examples

pulseData.cpp.

16.13.2.9 getQuietTime()

double AisDiffPulseVoltammetryElement::getQuietTime () const

Returns

The quiet time duration in seconds.

16.13.2.10 getQuietTimeSamplingInterval()

 $\verb|double AisDiffPulseVoltammetryElement::getQuietTimeSamplingInterval () constitution of the property of the$

Returns

samplingInterval The quiet time sampling interval to set in seconds.

16.13.2.11 getStartVoltage()

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

Returns

the value of the start voltage in volts.

16.13.2.12 getVStep()

double AisDiffPulseVoltammetryElement::getVStep () const

Returns

the value set for the potential step in volts.

See also

setVStep

16.13.2.13 isAutoRange()

bool AisDiffPulseVoltammetryElement::isAutoRange () const

Returns

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

Deprecated This function is deprecated and no longer supports auto range for this element. Specify the current using setApproxMaxCurrent(). The device will determine the appropriate range based on the current value.

16.13.2.14 isEndVoltageVsOCP()

bool AisDiffPulseVoltammetryElement::isEndVoltageVsOCP () const

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

16.13.2.15 isStartVoltageVsOCP()

 $\verb|bool AisDiffPulseVoltammetryElement:: is StartVoltageVsOCP () constant to the property of the property of$

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

16.13.2.16 setAlphaFactor()

This is an **optional** parameter. If nothing is set, the device will use the default value of 75.

Parameters

alphaFactor the value for the alphaFactor ranges from 0 to 100.

16.13.2.17 setApproxMaxCurrent()

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

This 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.

Examples

pulseData.cpp.

16.13.2.18 setAutoRange()

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

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

Deprecated This function is deprecated. Use setApproxMaxCurrent() to specify the current range instead.

16.13.2.19 setEndVoltage()

```
\label{lem:condition} \begin{tabular}{ll} void AisDiffPulseVoltammetryElement::setEndVoltage ( \\ double & endVoltage) \end{tabular}
```

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

Parameters

ge the value to set for the ending voltage in volts.
--

16.13.2.20 setEndVoltageVsOCP()

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.

Examples

pulseData.cpp.

16.13.2.21 setPulseHeight()

```
\begin{tabular}{ll} \begin{tabular}{ll} void AisDiffPulseVoltammetryElement::setPulseHeight (\\ double $pulseHeight)$ \end{tabular}
```

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.
-------------	---

16.13.2.22 setPulsePeriod()

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

Parameters

nulcoPoriod	the value to set for the pulse period in seconds.
puiserenou	the value to set for the pulse period in seconds.

16.13.2.23 setPulseWidth()

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

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

16.13.2.24 setQuietTime()

```
\begin{tabular}{ll} \begin{tabular}{ll} void AisDiffPulseVoltammetryElement::setQuietTime ( \\ \begin{tabular}{ll} double & quietTime) \end{tabular}
```

Parameters

quietTime	The quiet time duration to set in seconds.	1
-----------	--	---

16.13.2.25 setQuietTimeSamplingInterval()

Parameters

quietTimeSamplingInterval	The quiet time sampling interval to set in seconds.
---------------------------	---

16.13.2.26 setStartVoltage()

Parameters

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

16.13.2.27 setStartVoltageVsOCP()

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

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.

Examples

pulseData.cpp.

16.13.2.28 setVStep()

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

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.

Note

Regardless of vStep's sign, the device will determine the step direction based on the start and end voltage.

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

· AisDiffPulseVoltammetryElement.h

16.14 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>
#include <AisEISGalvanostaticElement.h>
```

Public Member Functions

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

the EIS galvanostatic element constructor.

AisEISGalvanostaticElement (const AisEISGalvanostaticElement &)

copy constructor for the AisEISGalvanostaticElement object.

• AisEISGalvanostaticElement & operator= (const AisEISGalvanostaticElement &)

overload equal to operator for the AisEISGalvanostaticElement object.

· QString getName () const override

get the name of the element.

• QStringList getCategory () const override

get a list of applicable categories of the element.

• double getQuietTime () const

Gets the quiet time duration.

void setQuietTime (double quietTime)

Sets the quiet time duration.

double getQuietTimeSamplingInterval () const

gets the quiet time sampling interval.

void setQuietTimeSamplingInterval (double quietTimeSamplingInterval)

Sets the quiet time sampling interval.

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.

• unsigned int getMinimumCycles () const

get the minimum number of periods of applied sinusoidal current to sample at each frequency.

void setMinimumCycles (unsigned int numberOfCycle)

set the minimum number of periods of applied sinusoidal current to sample at each frequency.

16.14.1 Constructor & Destructor Documentation

16.14.1.1 AisEISGalvanostaticElement()

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).
currentAmplitude	the AC current amplitude.

16.14.2 Member Function Documentation

16.14.2.1 getAmplitude()

```
double AisEISGalvanostaticElement::getAmplitude () const
```

Returns

the value set for the AC current amplitude in Amps.

16.14.2.2 getBiasCurrent()

double AisEISGalvanostaticElement::getBiasCurrent () const

Returns

the value set for the DC bias in Amps.

16.14.2.3 getCategory()

QStringList AisEISGalvanostaticElement::getCategory () const [override]

Returns

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

16.14.2.4 getEndFreq()

double AisEISGalvanostaticElement::getEndFreq () const

Returns

the value set for the current end frequency in Hz

16.14.2.5 getMinimumCycles()

unsigned int AisEISGalvanostaticElement::getMinimumCycles () const

Returns

get number of cycles to sample at each frequency.

16.14.2.6 getName()

QString AisEISGalvanostaticElement::getName () const [override]

Returns

The name of the element: "Galvanostatic EIS".

16.14.2.7 getQuietTime()

 $\verb|double AisEISGalvanostaticElement::getQuietTime () const|\\$

Returns

The quiet time duration in seconds.

16.14.2.8 getQuietTimeSamplingInterval()

double AisEISGalvanostaticElement::getQuietTimeSamplingInterval () const

Returns

samplingInterval The quiet time sampling interval to set in seconds.

16.14.2.9 getStartFreq()

```
double AisEISGalvanostaticElement::getStartFreq () const
```

Returns

the value set for the current start frequency in Hz?

16.14.2.10 getStepsPerDecade()

```
double AisEISGalvanostaticElement::getStepsPerDecade () const
```

Returns

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

16.14.2.11 setAmplitude()

Parameters

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

16.14.2.12 setBiasCurrent()

Parameters

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

16.14.2.13 setEndFreq()

```
void AisEISGalvanostaticElement::setEndFreq ( \label{eq:condFreq} \mbox{double } endFreq)
```

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

16.14.2.14 setMinimumCycles()

Parameters

16.14.2.15 setQuietTime()

```
void AisEISGalvanostaticElement::setQuietTime ( \label{eq:quietTime} \mbox{double } quietTime)
```

Parameters

quietTime The quiet time duration to set in seconds.
--

16.14.2.16 setQuietTimeSamplingInterval()

```
\label{local_problem} \begin{tabular}{ll} void AisEISGalvanostaticElement::setQuietTimeSamplingInterval ( \\ double & quietTimeSamplingInterval) \end{tabular}
```

Parameters

16.14.2.17 setStartFreq()

```
\begin{tabular}{ll} \begin{tabular}{ll} void AisEISGalvanostaticElement::setStartFreq ( \\ \begin{tabular}{ll} double & startFreq) \end{tabular}
```

Parameters

16.14.2.18 setStepsPerDecade()

```
\begin{tabular}{ll} \begin{tabular}{ll} void AisEISGalvanostaticElement::setStepsPerDecade ( \\ \begin{tabular}{ll} double & stepsPerDecade) \end{tabular}
```

Parameters

stepsPerDecade	the value to set for the number of steps per decade.
010po. 0.200aa0	the raide to certical the manner of etope per decades.

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

· AisEISGalvanostaticElement.h

16.15 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>
#include <AisEISPotentiostaticElement.h>
```

Public Member Functions

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

the EIS potentiostatic element

• AisEISPotentiostaticElement (const AisEISPotentiostaticElement &)

 $copy\ constructor\ for\ the\ \textit{AisEISPotentiostaticElement}\ object.$

• 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 getQuietTime () const

Gets the quiet time duration.

void setQuietTime (double quietTime)

Sets the quiet time duration.

double getQuietTimeSamplingInterval () const

gets the quiet time sampling interval.

void setQuietTimeSamplingInterval (double quietTimeSamplingInterval)

Sets the quiet time sampling interval.

• 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.

• unsigned int getMinimumCycles () const

get the minimum number of periods of applied sinusoidal voltage to sample at each frequency.

void setMinimumCycles (unsigned int numberOfCycle)

set the minimum number of periods of applied sinusoidal voltage to sample at each frequency.

16.15.1 Constructor & Destructor Documentation

16.15.1.1 AisEISPotentiostaticElement()

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).
voltageAmplitude	the AC voltage amplitude.

16.15.2 Member Function Documentation

16.15.2.1 getAmplitude()

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

Returns

the value set for the AC voltage amplitude in volts.

16.15.2.2 getBiasVoltage()

double AisEISPotentiostaticElement::getBiasVoltage () const

Returns

the value set for the DC bias in volts.

16.15.2.3 getCategory()

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

Returns

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

16.15.2.4 getEndFreq()

double AisEISPotentiostaticElement::getEndFreq () const

Returns

the value set for the voltage end frequency in Hz

16.15.2.5 getMinimumCycles()

unsigned int AisEISPotentiostaticElement::getMinimumCycles () const

Returns

get number of cycles to sample at each frequency.

16.15.2.6 getName()

QString AisEISPotentiostaticElement::getName () const [override]

Returns

The name of the element: "Potentiostatic EIS".

16.15.2.7 getQuietTime()

 $\verb|double AisEISPotentiostaticElement::getQuietTime () const$

Returns

The quiet time duration in seconds.

16.15.2.8 getQuietTimeSamplingInterval()

 $\verb|double AisEISPotentiostaticElement::getQuietTimeSamplingInterval () const|\\$

Returns

samplingInterval The quiet time sampling interval to set in seconds.

16.15.2.9 getStartFreq()

double AisEISPotentiostaticElement::getStartFreq () const

Returns

the value set for the start frequency in Hz

16.15.2.10 getStepsPerDecade()

double AisEISPotentiostaticElement::getStepsPerDecade () const

Returns

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

16.15.2.11 isBiasVoltageVsOCP()

```
\verb|bool AisEISPotentiostaticElement:: is \verb|BiasVoltageVsOCP| () const|
```

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.

16.15.2.12 setAmplitude()

Parameters

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

16.15.2.13 setBiasVoltage()

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

Parameters

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

16.15.2.14 setBiasVoltageVsOCP()

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

Parameters

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

16.15.2.15 setEndFreq()

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

Parameters

16.15.2.16 setMinimumCycles()

```
void AisEISPotentiostaticElement::setMinimumCycles ( unsigned\ int\ numberOfCycle)
```

Parameters

numberOfCvcle	number of cycles to sample at each frequency.

16.15.2.17 setQuietTime()

Parameters

16.15.2.18 setQuietTimeSamplingInterval()

```
\label{lem:condition} \begin{tabular}{ll} void AisEISP otentios tatic Element:: set Quiet Time Sampling Interval ( \\ double & quiet Time Sampling Interval) \end{tabular}
```

16.15.2.19 setStartFreq()

Parameters

startFreq	the value to set the starting frequency Hz
-----------	--

16.15.2.20 setStepsPerDecade()

Parameters

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

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

· AisEISPotentiostaticElement.h

16.16 AisErrorCode Class Reference

This class contains the possible error codes returned to the user when working with the API. Error codes can help diagnose issues such as invalid parameters, communication failures, or device malfunctions. By handling errors properly, you can ensure reliable operation of your experiments.

```
#include <AisErrorCode.h>
#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, FeatureNotSupported = 14, ManualExperimentNotRunning = 51, ExperimentNotUploaded
 ExperimentIsEmpty = 53, InvalidParameters = 54, ChannelNotBusy = 55, ExperimentUploaded = 56,
 DeviceCommunicationFailed = 100, FailedToSetManualModeCurrentRange = 101, FailedToSetManualModeConstantVoltage
 = 102, FailedToPauseExperiment = 103,
 FailedToResumeExperiment = 104, FailedToStopExperiment = 105, FailedToUploadExperiment = 106,
 ExperimentAlreadyPaused = 107,
 ExperimentAlreadyRun = 108, FailedToSetManualModeVoltageRange = 109, FailedToSetManualModeConstantCurrent
 = 110, FailedToSetManualModeInOCP = 111,
 FailedToSetManualModeSamplingInterval = 112, FailedToSetIRComp = 113, FailedToSetCompRange =
 114, FailedToSetChannelMaximumVoltage = 115,
 FailedToSetChannelMinimumVoltage = 116, FailedToSetChannelMaximumCurrent = 117, FailedToSetChannelMinimumCurrent
 = 118, FailedToSetChannelMinimumTemperature = 119,
 FailedRequest = 254 }
     The possible error codes that can be returned to the user.
```

Public Member Functions

- QString message () const
 - a function to get a message explaining the error.
- int value () const

a function to get the error code.

16.16.1 Member Enumeration Documentation

16.16.1.1 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 the failure was due to the channel being busy.
DeviceNotFound	indicates that no device was detected to be connected.
FeatureNotSupported	indicates that the feature is not available on the device.
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.

Enumerator

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.
ExperimentUploaded	indicates that the given command could not be completed because an experiment is already uploaded to the channel.
DeviceCommunicationFailed	indicates that there was failure in communication with the device.
FailedToSetManualModeCurrentRange	indicates failure to set manual mode current range due to a possible communication failure with the device.
FailedToSetManualModeConstantVoltage	indicates failure to set manual mode constant voltage due to a possible communication failure with the device
FailedToPauseExperiment	indicates that pausing the experiment failed because either there is no active experiment or due to a possible communication failure with the device.
FailedToResumeExperiment	indicates that resuming the experiment failed because either there is no paused experiment or due to a possible communication failure with the device.
FailedToStopExperiment	indicates that stopping the experiment failed because either there is no experiment running, the experiment is paused, or due to a 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 the experiment is already paused.
ExperimentAlreadyRun	indicates that resuming the experiment failed because an experiment is already running.
FailedToSetManualModeConstantCurrent	indicates failure to set manual mode constant current due to a possible communication failure with the device.
FailedToSetManualModeInOCP	indicates failure of setting manual mode in open circuit mode for possible communication failure with the device.
FailedToSetManualModeSamplingInterval	indicates failure of setting manual mode sampling interval. possible communication failure with the device.
FailedToSetIRComp	indicates failure of setting IR Compensation. Possible communication failure with the device.
FailedToSetCompRange	indicates failure of setting Compensation Range. Possible communication failure with the device.
FailedToSetChannelMaximumVoltage	indicates failure of setting Channel Maximum Voltage. Possible communication failure with the device.
FailedToSetChannelMinimumVoltage	indicates failure of setting Channel Minimum Voltage. Possible communication failure with the device.
FailedToSetChannelMaximumCurrent	indicates failure of setting Channel Maximum Current. Possible communication failure with the device.
FailedToSetChannelMinimumCurrent	indicates failure of setting Channel Minimum Current. Possible communication failure with the device.
FailedToSetChannelMinimumTemperature	indicates failure of setting Channel Maximum Temperature. Possible communication failure with the device.

16.16.2 Member Function Documentation

16.16.2.1 message()

```
QString AisErrorCode::message () const
```

Returns

a message that explains the error.

16.16.2.2 value()

```
int AisErrorCode::value () const
```

Returns

the error code

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

· AisErrorCode.h

16.17 AisExperiment Class Reference

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

```
#include <AisExperiment.h>
#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, unsigned int repeat=1)

Append an element to this experiment.

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

Append a sub experiment to this experiment.

Friends

· class AisInstrumentHandler

16.17.1 Constructor & Destructor Documentation

16.17.1.1 AisExperiment()

Parameters

exp the custom experiment to copy from.

16.17.2 Member Function Documentation

16.17.2.1 appendElement()

Parameters

element	The experiment element to be appended to this experiment.
repeat	The number of times this element will be repeated. This is an optional parameter with a default value
	of 1.
	The minimum value is 1. If smaller the function will not append the node to the experiment.
	The maximum value is 65535. The function will clamp any value greater than this to 65535, and it
	will append the node to the experiment.

Returns

true if the element was appended to the experiment 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.

16.17.2.2 appendSubExperiment()

Parameters

subExp	A sub experiment to be appended to this experiment.
repeat	The number of times this sub experiment will be repeated. This is an optional parameter with a
	default value of 1.
	The minimum value is 1. If smaller, the function will not append the sub experiment to the experiment
	The maximum value is 65535. The function will clamp any value greater than this to 65535, and it will
	append the sub experiment to the experiment.

Returns

true if the sub experiment was appended to the experiment and false otherwise.

16.17.2.3 getCategory()

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

Returns

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

16.17.2.4 getDescription()

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

Returns

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

16.17.2.5 getExperimentName()

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

Returns

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

16.17.2.6 operator=()

exp the custom experiment to copy from.

16.17.2.7 setDescription()

Parameters

16.17.2.8 setExperimentName()

Parameters

name the name to be set for the custom experiment.

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

· AisExperiment.h

16.18 AisExperimentNode Struct Reference

A structure containing some information regarding the running element.

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

• int cycle

this number is cycle within the element.

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

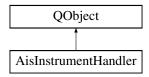
· AisDataPoints.h

16.19 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>
#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, const QString &reason)
 - 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.
- void deviceError (uint8_t channel, const QString &error)
 - a signal that is emitted whenever device send any critical error.

Public Member Functions

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

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 startIdleSampling (uint8_t channel) const

start idle sampling when an experiment is neither uploaded nor running on the specified 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)

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.

AisErrorCode setFanSpeedMaximum () const

Set the fan speed to maximum when an experiment is active. Only available on some devices.

AisErrorCode setFanSpeedVariable () const

Set the fan speed to always adjust automatically based on the internal temperature of the instrument. Only available on some devices.

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 setManualModeCurrentRange (uint8 t channel, int currentRangeIndex) const

set the current range for the manual experiment. Once a range is set, autoranging capability is turned off. That means that during potentiostatic control, the current range may range up if necessary, but it will not drop below the user-set range. During galvanostatic control, the lowest current range that contains the designated setpoint will be chosen, provided it is not lower than the user-set range.

AisErrorCode setManualModeCurrentAutorange (uint8_t channel) const

enable current autoranging for the manual experiment.

AisErrorCode setManualModeVoltageRange (uint8 t channel, int voltageRangeIndex) const

set the voltage range for the manual experiment. Once a range is set, autoranging capability is turned off. That means that during galvanostatic control, the voltage range may range up if necessary, but it will not drop below the user-set range. During potentiostatic control, the lowest voltage range that contains the designated setpoint will be chosen, provided it is not lower than the user-set range.

AisErrorCode setManualModeVoltageAutorange (uint8 t channel) const

enable voltage autoranging for the manual experiment.

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.
- $\bullet \ \ \mathsf{std} :: \mathsf{vector} < \mathsf{std} :: \mathsf{pair} < \mathsf{double}, \ \mathsf{double} > > \mathsf{getManualModeVoltageRangeList} \ (\mathsf{uint8_t} \ \mathsf{channel}) \ \mathsf{constante} = \mathsf{vector} < \mathsf{vecto$

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

• AisErrorCode setChannelMaximumVoltage (uint8_t channel, double Vmax) const

Sets the maximum allowable voltage for a channel. The experiment will stop if the measured voltage exceeds this limit

AisErrorCode setChannelMinimumVoltage (uint8_t channel, double Vmin) const

Sets the minimum allowable voltage for a channel. The experiment will stop if the measured voltage falls below this limit.

• AisErrorCode setChannelMaximumCurrent (uint8 t channel, double lmax) const

Sets the maximum allowable current for a channel. The experiment will stop if the measured current exceeds this limit

• AisErrorCode setChannelMinimumCurrent (uint8_t channel, double lmin) const

Sets the minimum allowable current for a channel. The experiment will stop if the measured current falls below this

AisErrorCode setChannelMaximumTemperature (uint8 t channel, double MaxTemperature) const

Sets the maximum allowable temperature for a channel. The experiment will stop if the measured temperature exceeds this limit.

AisErrorCode resetChannelLimits (uint8_t channel) const

Resets all limits for the specified channel.

16.19.1 Member Function Documentation

16.19.1.1 activeACDataReady

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

Examples

basicExperiment.cpp, dataOutput.cpp, nonblockingExperiment.cpp, and pulseData.cpp.

16.19.1.2 activeDCDataReady

Parameters

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

Examples

advancedExperiment.cpp, basicExperiment.cpp, dataOutput.cpp, linkedChannels.cpp, manualExperiment.cpp, nonblockingExperiment.cpp, and pulseData.cpp.

16.19.1.3 deviceError

Parameters

channel	the channel number at which error rise.
error	information about error message.

Note

stop experiment command will automatilcally send on channel.

Examples

 $advanced {\sf Experiment.cpp, basic Experiment.cpp, linked Channels.cpp, {\it and } manual {\sf Experiment.cpp.} \\$

16.19.1.4 eraseRecoverData()

AisErrorCode AisInstrumentHandler::eraseRecoverData () const

Returns

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

- AisErrorCode::DeviceNotFound
- · AisErrorCode::DeviceCommunicationFailed

16.19.1.5 experimentNewElementStarting

Parameters

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

See also

AisExperimentNode

Examples

advancedExperiment.cpp, basicExperiment.cpp, dataOutput.cpp, nonblockingExperiment.cpp, and pulseData.cpp.

16.19.1.6 experimentPaused

Parameters

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

Examples

pulseData.cpp.

16.19.1.7 experimentResumed

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

Examples

pulseData.cpp.

16.19.1.8 experimentStopped

Parameters

channel	the channel on which the experiment has stopped.
reason	the reason why the experiment has stopped.

Examples

advancedControlFlow.cpp, advancedExperiment.cpp, basicExperiment.cpp, dataOutput.cpp, manualExperiment.cpp, nonblockingExperiment.cpp, and pulseData.cpp.

16.19.1.9 getExperimentUTCStartTime()

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

Parameters

Returns

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

16.19.1.10 getFreeChannels()

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

Returns

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

16.19.1.11 getLinkedChannels()

Parameters

channel	a valid channel number to find which other channels are linked to it.
---------	---

Returns

a list of channels linked to the channel parameter.

16.19.1.12 getManualModeCurrentRangeList()

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 or setManualModeConstantCurrent.

Parameters

channel	a valid channel number for which to check the current range.
---------	--

Returns

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

16.19.1.13 getManualModeVoltageRangeList()

```
\label{eq:std:std:std:std:std:std:std:std} std::pair < double, double > \\ > AisInstrumentHandler::getManualModeVoltageRange \leftarrow List ( \\ uint8\_t \ channel) \ const
```

The list is indexed, with each index containing a range with minimum and maximum voltage for the range. You can pass the index of the desired current range to setManualModeConstantVoltage or setManualModeConstantCurrent.

Parameters

channel	a valid channel number for which to check the current range.
---------	--

Returns

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

16.19.1.14 getNumberOfChannels()

```
\verb|int AisInstrumentHandler::getNumberOfChannels () const|\\
```

Returns

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

16.19.1.15 groundFloatStateChanged

arounded	true if there is a connection to ground and false if the ground has disconnected.
9	The second secon

16.19.1.16 hasBipolarMode()

Parameters

Returns

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

16.19.1.17 idleDCDataReady

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.

16.19.1.18 isChannelBusy()

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.

16.19.1.19 isChannelPaused()

Parameters

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

Returns

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

16.19.1.20 pauseExperiment()

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.

16.19.1.21 recoverDataErased

Parameters

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

16.19.1.22 recoveryACDataReady

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

16.19.1.23 recoveryDCDataReady

Parameters

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

16.19.1.24 resetChannelLimits()

This function removes all configured voltage, current, and temperature limits for a specific channel, restoring it to its default state.

Parameters

The channel number for which the limits are to be reset	t.
---	----

Returns

AisErrorCode indicating success or failure of the operation.

16.19.1.25 resumeExperiment()

Parameters

channel the channel number to resume the experimen	t 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.

16.19.1.26 setBipolarLinkedChannels()

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

Parameters

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

Returns

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

Note

this functionality is only applicable to the cycler model.

16.19.1.27 setChannelMaximumCurrent()

Sets the maximum allowable current for a channel. The experiment will stop if the measured current exceeds this limit.

Parameters

channel	The channel number for which the current limit is to be set.
Imax	The maximum allowable current in amperes.

Returns

AisErrorCode indicating success or failure of the operation.

Note

- For Squidstat Cyclers, this limit will not apply to AC elements.
- · For all other devices, it is not recommended to use this for AC elements.

16.19.1.28 setChannelMaximumTemperature()

```
AisErrorCode AisInstrumentHandler::setChannelMaximumTemperature ( uint8_t channel, double MaxTemperature) const
```

Sets the maximum allowable temperature for a channel. The experiment will stop if the measured temperature exceeds this limit.

channel	The channel number for which the temperature limit is to be set.
MaxTemperature	The maximum allowable temperature in degrees Celsius.

Returns

AisErrorCode indicating success or failure of the operation.

Note

· For Squidstat Cyclers, this limit will not apply to AC elements.

16.19.1.29 setChannelMaximumVoltage()

Sets the maximum allowable voltage for a channel. The experiment will stop if the measured voltage exceeds this limit.

Parameters

channel	The channel number for which the voltage limit is to be set.
Vmax	The maximum allowable voltage in volts.

Returns

AisErrorCode indicating success or failure of the operation.

Note

- · For Squidstat Cyclers, this limit will not apply to AC elements.
- For all other devices, it is not recommended to use this for AC elements.

16.19.1.30 setChannelMinimumCurrent()

Sets the minimum allowable current for a channel. The experiment will stop if the measured current falls below this limit.

Parameters

channel	The channel number for which the current limit is to be set.
Imin	The minimum allowable current in amperes.

Returns

AisErrorCode indicating success or failure of the operation.

Note

- For Squidstat Cyclers, this limit will not apply to AC elements.
- For all other devices, it is not recommended to use this for AC elements.

16.19.1.31 setChannelMinimumVoltage()

Sets the minimum allowable voltage for a channel. The experiment will stop if the measured voltage falls below this limit.

Parameters

channel	The channel number for which the voltage limit is to be set.
Vmin	The minimum allowable voltage in volts.

Returns

AisErrorCode indicating success or failure of the operation.

Note

- For Squidstat Cyclers, this limit will not apply to AC elements.
- For all other devices, it is not recommended to use this for AC elements.

16.19.1.32 setCompRange()

Parameters

channel	the channel for which to set the compensation range.
	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

16.19.1.33 setFanSpeedMaximum()

AisErrorCode AisInstrumentHandler::setFanSpeedMaximum () const

Note

Setting this mode may remove small variations in data caused by fan cycling in some electrochemical systems.

Set the fan speed to maximum when an experiment is active. Only available on some devices. Model-specific fan behavior:

- · Squidstat Cycler: Not available.
- Squidstat Ace, Prime, Solo, and Squidstat Plus (serial numbers below 1700): Always operate in this mode. Fans will turn off when no experiment is running.
- Squidstat <u>Plus</u> (serial numbers <u>1700</u> or higher), Squidstat <u>Penta</u>, <u>Decka</u>, and <u>Venta</u>: Default operating mode. Fans will adjust based on the internal temperature of the instrument when no experiment is running.

Return values

AisErrorCode::Success	
AisErrorCode::DeviceNotFound	
AisErrorCode::FeatureNotSupported	
AisErrorCode::DeviceCommunicationFailed	

See also

setFanSpeedVariable

16.19.1.34 setFanSpeedVariable()

AisErrorCode AisInstrumentHandler::setFanSpeedVariable () const

Set the fan speed to always adjust automatically based on the internal temperature of the instrument. Only available on some devices.

Model-specific fan behavior:

- Squidstat Cycler: Always operates in this mode; no data variances occur from fan cycling.
- Squidstat Ace, Prime, Solo, and Squidstat Plus (serial numbers below 1700): Not available.
- Squidstat <u>Plus</u> (serial numbers <u>1700</u> or higher), Squidstat <u>Penta</u>, <u>Decka</u>, and <u>Venta</u>: Available on these models.

Return values

AisErrorCode::Success	
AisErrorCode::DeviceNotFound	
AisErrorCode::FeatureNotSupported	
AisErrorCode::DeviceCommunicationFailed	

See also

setFanSpeedMaximum

16.19.1.35 setIRComp()

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::DeviceNotFoundAisErrorCode::InvalidChannel

· AisErrorCode::InvalidParameters

16.19.1.36 setLinkedChannels()

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.

16.19.1.37 setManualModeConstantCurrent()

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
- AisErrorCode::FailedToSetManualModeConstantCurrent

16.19.1.38 setManualModeConstantVoltage() [1/2]

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

16.19.1.39 setManualModeConstantVoltage() [2/2]

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

16.19.1.40 setManualModeCurrentAutorange()

Parameters

	channel	a valid channel number to enable current autoranging for.
--	---------	---

Returns

AisErrorCode::Success if enabling current autoranging successful. If not successful, possible returned errors are:

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

16.19.1.41 setManualModeCurrentRange()

Parameters

channel	a valid channel number to set the current range for.
currentRangeIndex	the index of the desired current range.

Returns

AisErrorCode::Success if setting the current range 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::FailedToSetManualModeCurrentRange
- · AisErrorCode::ManualExperimentNotRunning
- AisErrorCode::DeviceNotFound
- · AisErrorCode::InvalidChannel

16.19.1.42 setManualModeOCP()

To apply the set potential or current, leave the open circuit potential mode off. This operation is reversed automatically when calling either setManualModeConstantVoltage() or setManualModeConstantCurrent()

Parameters

channel	a valid channel number to set open circuit mode on.
---------	---

Returns

AisErrorCode::Success if turning on the open circuit mode was successful. If not successful, possible returned errors are:

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

16.19.1.43 setManualModeSamplingInterval()

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

16.19.1.44 setManualModeVoltageAutorange()

Parameters

ch	nannel	a valid channel number to enable voltage autoranging for.
----	--------	---

Returns

AisErrorCode::Success if enabling voltage autoranging successful. If not successful, possible returned errors are:

- AisErrorCode::FailedToSetManualModeVoltageRange
- AisErrorCode::ManualExperimentNotRunning
- AisErrorCode::DeviceNotFound
- AisErrorCode::InvalidChannel

16.19.1.45 setManualModeVoltageRange()

Parameters

channel	a valid channel number to set the voltage range for.
voltageRangeIndex	the index of the desired voltage range.

Returns

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

- AisErrorCode::FailedToSetManualModeVoltageRange
- AisErrorCode::ManualExperimentNotRunning
- AisErrorCode::DeviceNotFound
- AisErrorCode::InvalidChannel

16.19.1.46 skipExperimentStep()

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

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

16.19.1.47 startIdleSampling()

Returns

AisErrorCode::Success if the request to start idle data was sent. If not successful, possible returned errors are:

- AisErrorCode::ExperimentUploaded
- · AisErrorCode::DeviceNotFound
- AisErrorCode::InvalidChannel
- AisErrorCode::BusyChannel

See also

isChannelBusy

16.19.1.48 startManualExperiment()

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:

- $\bullet \ \ Ais Error Code :: Failed Manual Mode Start Experiment$
- AisErrorCode::DeviceNotFound
- AisErrorCode::InvalidChannel
- AisErrorCode::BusyChannel

16.19.1.49 startUploadedExperiment()

Parameters

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

Returns

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

- AisErrorCode::DeviceCommunicationFailed
- · AisErrorCode::ExperimentNotUploaded
- AisErrorCode::DeviceNotFound
- AisErrorCode::InvalidChannel
- AisErrorCode::BusyChannel

See also

```
uploadExperimentToChannel isChannelBusy
```

16.19.1.50 stopExperiment()

Parameters

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.

16.19.1.51 uploadExperimentToChannel() [1/2]

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.

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

16.19.1.52 uploadExperimentToChannel() [2/2]

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

16.20 AisMottSchottkyElement Class Reference

This class performs Mott-Schottky analysis on the working electrode for a specified range of potentials.

```
#include <AisMottSchottkyElement.h>
#include <AisMottSchottkyElement.h>
```

Public Member Functions

• AisMottSchottkyElement (double startingPotential, double endingPotential, double voltageStep, double startFrequency, double endFrequency, double stepsPerDecade, double amplitude, unsigned int minCycles)

Constructor for the Mott-Schottky experiment element.

AisMottSchottkyElement (const AisMottSchottkyElement &other)

Copy constructor for the AisMottSchottkyElement object.

AisMottSchottkyElement & operator= (const AisMottSchottkyElement & other)

Assignment operator for the AisMottSchottkyElement object.

∼AisMottSchottkyElement () override

Destructor for the AisMottSchottkyElement object.

QString getName () const override

Get the name of the experiment element.

QStringList getCategory () const override

Get a list of applicable categories of the experiment element.

double getStartingPotential () const

Get the starting potential for the experiment.

void setStartingPotential (double startingPotential)

Set the starting potential for the experiment.

• double getEndingPotential () const

Get the ending potential for the experiment.

void setEndingPotential (double endingPotential)

Set the ending potential for the experiment.

• double getVoltageStep () const

Get the voltage step size between each potential.

void setVoltageStep (double voltageStep)

Set the voltage step size between each potential.

• double getStartFrequency () const

Get the starting frequency for the EIS measurement.

void setStartFrequency (double startFrequency)

Set the starting frequency for the EIS measurement.

• double getEndFrequency () const

Get the ending frequency for the EIS measurement.

void setEndFrequency (double endFrequency)

Set the ending frequency for the EIS measurement.

double getStepsPerDecade () const

Get the number of frequency steps per decade.

void setStepsPerDecade (double stepsPerDecade)

Set the number of frequency steps per decade.

· double getAmplitude () const

Get the amplitude of the AC signal used in the EIS measurement.

• void setAmplitude (double amplitude)

Set the amplitude of the AC signal used in the EIS measurement.

· unsigned int getMinCycles () const

Get the minimum number of cycles per frequency step.

· void setMinCycles (unsigned int minCycles)

Set the minimum number of cycles per frequency step.

• double getQuietTime () const

Get the quiet time before starting the EIS measurement.

void setQuietTime (double quietTime)

Set the quiet time before starting the EIS measurement.

double getQuietTimeSampInterval () const

Get the sampling interval during the guiet time.

void setQuietTimeSampInterval (double quietTimeSampInterval)

Set the sampling interval during the quiet time.

• double getStepQuietTime () const

Get the quiet time after each potential step before starting the EIS measurement.

void setStepQuietTime (double stepQuietTime)

Set the quiet time after each potential step before starting the EIS measurement.

double getStepQuietSampInterval () const

Get the sampling interval during the quiet time after each potential step.

void setStepQuietSampInterval (double stepQuietTimeSampInterval)

Set the sampling interval during the quiet time after each potential step.

bool isStartVoltageVsOCP () const

Check if the starting voltage is measured versus the open circuit potential (OCP).

void setStartVoltageVsOCP (bool startVsOCP)

Set whether the starting voltage is measured versus the open circuit potential (OCP).

• bool isEndVoltageVsOCP () const

Check if the ending voltage is measured versus the open circuit potential (OCP).

void setEndVoltageVsOCP (bool endVsOCP)

Set whether the ending voltage is measured versus the open circuit potential (OCP).

16.20.1 Constructor & Destructor Documentation

16.20.1.1 AisMottSchottkyElement() [1/2]

Parameters

startingPotential	The starting potential (voltage) for the experiment.
endingPotential	The ending potential (voltage) for the experiment.
voltageStep	The voltage step size between each potential during the experiment.
startFrequency	The starting frequency for the EIS measurement.
endFrequency	The ending frequency for the EIS measurement.

stepsPerDecade	The number of frequency steps per decade.
amplitude	The amplitude of the AC signal used in the EIS measurement.
minCycles	The minimum number of cycles per frequency step during the EIS measurement.

16.20.1.2 AisMottSchottkyElement() [2/2]

Parameters

16.20.2 Member Function Documentation

16.20.2.1 getAmplitude()

double AisMottSchottkyElement::getAmplitude () const

Returns

The AC amplitude in volts.

16.20.2.2 getCategory()

QStringList AisMottSchottkyElement::getCategory () const [override]

Returns

A list of categories where the experiment is applicable, such as "Advanced Experiments".

16.20.2.3 getEndFrequency()

double AisMottSchottkyElement::getEndFrequency () const

Returns

The ending frequency in Hz.

16.20.2.4 getEndingPotential()

double AisMottSchottkyElement::getEndingPotential () const

Returns

The ending potential in volts.

16.20.2.5 getMinCycles()

 $unsigned\ int\ AisMottSchottkyElement::getMinCycles\ ()\ const$

Returns

The minimum number of cycles.

16.20.2.6 getName()

QString AisMottSchottkyElement::getName () const [override]

Returns

The name of the element, "Mott-Schottky".

16.20.2.7 getQuietTime()

double AisMottSchottkyElement::getQuietTime () const

Returns

The quiet time in seconds.

16.20.2.8 getQuietTimeSampInterval()

double AisMottSchottkyElement::getQuietTimeSampInterval () const

Returns

The sampling interval in seconds.

16.20.2.9 getStartFrequency()

 $\verb|double AisMottSchottkyElement::getStartFrequency () const$

Returns

The starting frequency in Hz.

16.20.2.10 getStartingPotential()

double AisMottSchottkyElement::getStartingPotential () const

Returns

The starting potential in volts.

16.20.2.11 getStepQuietSampInterval()

 $\verb|double AisMottSchottkyElement::getStepQuietSampInterval () const|\\$

Returns

The sampling interval in seconds.

16.20.2.12 getStepQuietTime()

double AisMottSchottkyElement::getStepQuietTime () const

Returns

The quiet time after each potential step in seconds.

16.20.2.13 getStepsPerDecade()

double AisMottSchottkyElement::getStepsPerDecade () const

Returns

The number of steps per decade.

16.20.2.14 getVoltageStep()

double AisMottSchottkyElement::getVoltageStep () const

Returns

The voltage step size in volts.

16.20.2.15 isEndVoltageVsOCP()

 $\verb|bool AisMottSchottkyElement:: is \verb|EndVoltageVsOCP| () const|\\$

Returns

True if the ending voltage is measured versus OCP, false otherwise.

16.20.2.16 isStartVoltageVsOCP()

 $\verb|bool AisMottSchottkyElement:: is StartVoltageVsOCP () const$

Returns

True if the starting voltage is measured versus OCP, false otherwise.

16.20.2.17 operator=()

Parameters

other	The object to assign from.
-------	----------------------------

Returns

A reference to the assigned object.

16.20.2.18 setAmplitude()

Parameters

amplitude	The AC amplitude in volts.
-----------	----------------------------

16.20.2.19 setEndFrequency()

Parameters

endFrequency	The ending frequency in Hz.
--------------	-----------------------------

16.20.2.20 setEndingPotential()

Parameters

endingPotential	The ending potential in volts.
-----------------	--------------------------------

16.20.2.21 setEndVoltageVsOCP()

```
\label{eq:condition} \begin{tabular}{ll} void AisMottSchottkyElement::setEndVoltageVsOCP ( \\ bool $endVsOCP)$ \end{tabular}
```

Parameters

endVsO($CP \mid$	True if the ending voltage is measured versus OCP, false otherwise.
---------	-----------	---

16.20.2.22 setMinCycles()

```
void AisMottSchottkyElement::setMinCycles (
          unsigned int minCycles)
```

Parameters

cycles.	number of	The minimum	minCycles
---------	-----------	-------------	-----------

16.20.2.23 setQuietTime()

Parameters

quietTime	The quiet time in seconds.
-----------	----------------------------

16.20.2.24 setQuietTimeSampInterval()

```
\label{lement:setQuietTimeSampInterval} \mbox{ \begin{tabular}{ll} \label{lement:setQuietTimeSampInterval \end{tabular}} \mbox{ \begin{tabular}{ll} \label{lement:setQuietTimeSampInt
```

Parameters

quietTimeSampInterval The sampling interval in seconds
--

16.20.2.25 setStartFrequency()

```
\label{lement:setStartFrequency} \mbox{ void AisMottSchottkyElement::setStartFrequency (} \\ \mbox{ double } startFrequency)
```

Parameters

startFrequency The starting frequency in Hz.
--

16.20.2.26 setStartingPotential()

Parameters

startingPotential	The starting potential in volts.
-------------------	----------------------------------

16.20.2.27 setStartVoltageVsOCP()

```
\label{eq:condition} \begin{tabular}{ll} void AisMottSchottkyElement::setStartVoltageVsOCP ( \\ bool startVsOCP) \end{tabular}
```

Parameters

16.20.2.28 setStepQuietSampInterval()

```
\label{lem:condition} \begin{tabular}{ll} void AisMottSchottkyElement::setStepQuietSampInterval ( \\ double & stepQuietTimeSampInterval) \end{tabular}
```

Parameters

stepQuietTimeSampInterval	The sampling interval in seconds.

16.20.2.29 setStepQuietTime()

```
void AisMottSchottkyElement::setStepQuietTime ( \label{eq:condition} \mbox{double } stepQuietTime)
```

Parameters

16.20.2.30 setStepsPerDecade()

```
\label{lement:setStepsPerDecade} \begin{tabular}{ll} void AisMottSchottkyElement::setStepsPerDecade ( \\ double & stepsPerDecade) \end{tabular}
```

Parameters

stepsPerDecade The number of steps per decade.
--

16.20.2.31 setVoltageStep()

```
\begin{tabular}{ll} void AisMottSchottkyElement::setVoltageStep ( \\ double \begin{tabular}{ll} doubl
```

Parameters

voltageStep	The voltage step size in volts.
-------------	---------------------------------

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

· AisMottSchottkyElement.h

16.21 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>
#include <AisNormalPulseVoltammetryElement.h>
```

Public Member Functions

• AisNormalPulseVoltammetryElement (double startVoltage, double endVoltage, double voltageStep, double pulseWidth, double pulsePeriod)

the normal-pulse-voltammetry element constructor

• AisNormalPulseVoltammetryElement (double startVoltage, double endVoltage, double voltageStep, double pulseWidth, double pulsePeriod, double approxMaxCurrent)

the normal-pulse-voltammetry element constructor

AisNormalPulseVoltammetryElement (const AisNormalPulseVoltammetryElement &)

 $copy\ constructor\ for\ the\ {\it AisNormalPulseVoltammetryElement}\ object.$

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 getQuietTime () const

Gets the quiet time duration.

void setQuietTime (double quietTime)

Sets the quiet time duration.

double getQuietTimeSamplingInterval () const

gets the quiet time sampling interval.

void setQuietTimeSamplingInterval (double quietTimeSamplingInterval)

Sets the quiet time sampling interval.

· 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 set for the pulse width

• void setPulseWidth (double pulseWidth)

set the value in seconds for 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.

• double getAlphaFactor () const

Get the value set for the alpha factor.

• void setAlphaFactor (double alphaFactor)

alpha factor controls the percentage of data sampled during a given interval. Data will be averaged over the last n% of the sampling interval.

16.21.1 Constructor & Destructor Documentation

16.21.1.1 AisNormalPulseVoltammetryElement() [1/2]

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 seconds.
pulsePeriod	the value for the pulse period in seconds.

16.21.1.2 AisNormalPulseVoltammetryElement() [2/2]

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 seconds.
pulsePeriod	the value for the pulse period in seconds.
approxMaxCurrent	the value for the approximate maximum current in amperes.

16.21.2 Member Function Documentation

16.21.2.1 getAlphaFactor()

```
\verb|double AisNormalPulseVoltammetryElement::getAlphaFactor () const|\\
```

Returns

The value for the alpha factor is represented as a percent between 0 and 100.

Note

If nothing is set, this function will return a default value of 75.

16.21.2.2 getApproxMaxCurrent()

double AisNormalPulseVoltammetryElement::getApproxMaxCurrent () const

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.

16.21.2.3 getCategory()

QStringList AisNormalPulseVoltammetryElement::getCategory () const [override]

Returns

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

16.21.2.4 getEndVoltage()

double AisNormalPulseVoltammetryElement::getEndVoltage () const

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

Returns

the value set for the ending voltage in volts.

16.21.2.5 getName()

QString AisNormalPulseVoltammetryElement::getName () const [override]

Returns

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

16.21.2.6 getPulsePeriod()

 $\verb|double AisNormalPulseVoltammetryElement::getPulsePeriod () const$

Returns

the value for the pulse period in seconds.

See also

setPulsePeriod

16.21.2.7 getPulseWidth()

double AisNormalPulseVoltammetryElement::getPulseWidth () const

Returns

the value of the pulse width in seconds.

See also

setPulseWidth

16.21.2.8 getQuietTime()

double AisNormalPulseVoltammetryElement::getQuietTime () const

Returns

The quiet time duration in seconds.

16.21.2.9 getQuietTimeSamplingInterval()

 ${\tt double\ AisNormalPulseVoltammetryElement::getQuietTimeSamplingInterval\ ()\ const.}$

Returns

quiet time The quiet time sampling interval to set in seconds.

16.21.2.10 getStartVoltage()

double AisNormalPulseVoltammetryElement::getStartVoltage () const

Returns

the value of the start voltage in volts.

16.21.2.11 getVStep()

double AisNormalPulseVoltammetryElement::getVStep () const

Returns

the value set for the voltage step in volts.

See also

setVStep

16.21.2.12 isAutoRange()

bool AisNormalPulseVoltammetryElement::isAutoRange () const

Returns

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

Deprecated This function is deprecated and no longer supports auto range for this element. Specify the current using setApproxMaxCurrent(). The device will determine the appropriate range based on the current value.

16.21.2.13 isEndVoltageVsOCP()

bool AisNormalPulseVoltammetryElement::isEndVoltageVsOCP () const

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.

16.21.2.14 isStartVoltageVsOCP()

bool AisNormalPulseVoltammetryElement::isStartVoltageVsOCP () const

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

16.21.2.15 setAlphaFactor()

This is an **optional** parameter. If nothing is set, the device will use the default value of 75.

Parameters

alphaFactor	the value for the alphaFactor ranges from 0 to 100.
-------------	---

16.21.2.16 setApproxMaxCurrent()

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

Parameters

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

16.21.2.17 setAutoRange()

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

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

Deprecated This function is deprecated. Use setApproxMaxCurrent() to specify the current range instead.

16.21.2.18 setEndVoltage()

```
void AisNormalPulseVoltammetryElement::setEndVoltage ( \label{eq:double_endVoltage} double \ endVoltage)
```

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

Parameters

16.21.2.19 setEndVoltageVsOCP()

```
\label{local_voltageVsOCP} void \ \texttt{AisNormalPulseVoltammetryElement::} setEndVoltageVsOCP \ ( \\ bool \ endVoltageVsOcp)
```

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.

16.21.2.20 setPulsePeriod()

```
void AisNormalPulseVoltammetryElement::setPulsePeriod ( double pulsePeriod)
```

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
--

16.21.2.21 setPulseWidth()

```
void AisNormalPulseVoltammetryElement::setPulseWidth ( \label{eq:condition} \mbox{double } pulseWidth)
```

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.
----------------	--

16.21.2.22 setQuietTime()

Parameters

quietTime	The quiet time duration to set in seconds.
-----------	--

16.21.2.23 setQuietTimeSamplingInterval()

```
\label{lem:condition} \begin{tabular}{ll} void AisNormalPulseVoltammetryElement::setQuietTimeSamplingInterval ( \\ double & quietTimeSamplingInterval) \end{tabular}
```

Parameters

·	[-	
quiet LimeSamplingInterval	The quiet time sampling interval to set in seconds.	ı
quiet i i i e cui i gi i tei i ai	into quiet into eartipling into rai to eet in eeconaei	

16.21.2.24 setStartVoltage()

```
void AisNormalPulseVoltammetryElement::setStartVoltage ( \label{eq:double_startVoltage} double \ startVoltage)
```

Parameters

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

16.21.2.25 setStartVoltageVsOCP()

```
\label{local_voltageVsOCP} \mbox{ void AisNormalPulseVoltammetryElement::setStartVoltageVsOCP (} \\ \mbox{ bool } startVoltageVsOCP) \mbox{ }
```

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.

16.21.2.26 setVStep()

```
void AisNormalPulseVoltammetryElement::setVStep ( \label{eq:condition} \mbox{double } vStep)
```

The voltage step is the voltage difference between the heights of two consecutive pulses.

Parameters

vStep	the value for the voltage step in volts.
-------	--

Note

Regardless of vStep's sign, the device will determine the step direction based on the start and end voltage.

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

· AisNormalPulseVoltammetryElement.h

16.22 AisOpenCircuitElement Class Reference

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

```
#include <AisOpenCircuitElement.h>
#include <AisOpenCircuitElement.h>
```

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.

16.22.1 Constructor & Destructor Documentation

16.22.1.1 AisOpenCircuitElement()

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

Parameters

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

16.22.2 Member Function Documentation

16.22.2.1 getApproxMaxVoltage()

double AisOpenCircuitElement::getApproxMaxVoltage () const

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.

16.22.2.2 getCategory()

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

Returns

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

16.22.2.3 getMaxDuration()

double AisOpenCircuitElement::getMaxDuration () const

Returns

the value set for the duration of the experiment in seconds.

16.22.2.4 getMaxVoltage()

double AisOpenCircuitElement::getMaxVoltage () const

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.

16.22.2.5 getMindVdt()

double AisOpenCircuitElement::getMindVdt () const

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.

16.22.2.6 getMinVoltage()

double AisOpenCircuitElement::getMinVoltage () const

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.

16.22.2.7 getName()

QString AisOpenCircuitElement::getName () const [override]

Returns

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

16.22.2.8 getSamplingInterval()

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

Returns

the data sampling interval value in seconds.

16.22.2.9 isAutoVoltageRange()

```
bool AisOpenCircuitElement::isAutoVoltageRange () const
```

Returns

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

16.22.2.10 setApproxMaxVoltage()

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

Parameters

approxMaxVoltage the value for the maximum voltage expected in V.

16.22.2.11 setAutoVoltageRange()

```
void AisOpenCircuitElement::setAutoVoltageRange ()
```

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

16.22.2.12 setMaxDuration()

Parameters

maxDuration the value to set for the duration of the experiment in seconds.

16.22.2.13 setMaxVoltage()

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

Parameters

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

16.22.2.14 setMindVdt()

This 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).

16.22.2.15 setMinVoltage()

This 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.
minivollage	

16.22.2.16 setSamplingInterval()

```
\label{local_problem} \begin{tabular}{ll} void AisOpenCircuitElement::setSamplingInterval ( \\ double $\it samplingInterval) \end{tabular}
```

Parameters

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

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

· AisOpenCircuitElement.h

16.23 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>
#include <AisSquareWaveVoltammetryElement.h>
```

Public Member Functions

AisSquareWaveVoltammetryElement (double startVoltage, double endVoltage, double voltageStep, double pulseAmp, double pulseFrequency)

the square wave element constructor

 AisSquareWaveVoltammetryElement (double startVoltage, double endVoltage, double voltageStep, double pulseAmp, double pulseFrequency, double approxMaxCurrent)

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 getQuietTime () const

Gets the quiet time duration.

void setQuietTime (double quietTime)

Sets the quiet time duration.

double getQuietTimeSamplingInterval () const

gets the quiet time sampling interval.

void setQuietTimeSamplingInterval (double quietTimeSamplingInterval)

Sets the quiet time sampling interval.

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. The voltage step is added to the value of the starting potential of the previous pulse to start the new pulse.

• 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.

· double getAlphaFactor () const

Get the value set for the alpha factor.

void setAlphaFactor (double alphaFactor)

alpha factor controls the percentage of data sampled during a given interval. Data will be averaged over the last n% of the sampling interval.

16.23.1 Constructor & Destructor Documentation

16.23.1.1 AisSquareWaveVoltammetryElement() [1/2]

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.

Deprecated Use the constructor with the approxMaxCurrent parameter instead.

16.23.1.2 AisSquareWaveVoltammetryElement() [2/2]

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.
approxMaxCurrent	the value for the approximate maximum current in amperes.

16.23.2 Member Function Documentation

16.23.2.1 getAlphaFactor()

```
double AisSquareWaveVoltammetryElement::getAlphaFactor () const
```

Returns

The value for the alpha factor is represented as a percent between 0 and 100.

Note

If nothing is set, this function will return a default value of 75.

16.23.2.2 getApproxMaxCurrent()

```
\verb|double AisSquareWaveVoltammetryElement::getApproxMaxCurrent () const
```

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.

16.23.2.3 getCategory()

QStringList AisSquareWaveVoltammetryElement::getCategory () const [override]

Returns

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

16.23.2.4 getEndVoltage()

double AisSquareWaveVoltammetryElement::getEndVoltage () const

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

Returns

the value set for the ending voltage in volts.

16.23.2.5 getName()

QString AisSquareWaveVoltammetryElement::getName () const [override]

Returns

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

16.23.2.6 getPulseAmp()

 $\verb|double AisSquareWaveVoltammetryElement::getPulseAmp () const$

Returns

the value set for the pulse amplitude in volts.

See also

setPulseAmp

16.23.2.7 getPulseFreq()

 $\verb|double AisSquareWaveVoltammetryElement::getPulseFreq () const|\\$

Returns

the value set for the frequency in Hz.

16.23.2.8 getQuietTime()

double AisSquareWaveVoltammetryElement::getQuietTime () const

Returns

The quiet time duration in seconds.

16.23.2.9 getQuietTimeSamplingInterval()

 $\verb|double AisSquareWaveVoltammetryElement::getQuietTimeSamplingInterval () constitution of the constituti$

Returns

samplingInterval The quiet time sampling interval to set in seconds.

16.23.2.10 getStartVoltage()

double AisSquareWaveVoltammetryElement::getStartVoltage () const

Returns

the value of the start voltage in volts.

16.23.2.11 getVStep()

double AisSquareWaveVoltammetryElement::getVStep () const

Returns

the value set for the voltage step in volts.

See also

setVStep

16.23.2.12 isAutoRange()

 $\verb|bool AisSquareWaveVoltammetryElement:: is \verb|AutoRange| () | const|$

Returns

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

Deprecated This function is deprecated and no longer supports auto range for this element. Specify the current using setApproxMaxCurrent(). The device will determine the appropriate range based on the current value.

16.23.2.13 isEndVoltageVsOCP()

bool AisSquareWaveVoltammetryElement::isEndVoltageVsOCP () const

Returns

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

Note

if nothing is set, the default is false.

16.23.2.14 isStartVoltageVsOCP()

bool AisSquareWaveVoltammetryElement::isStartVoltageVsOCP () const

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

16.23.2.15 setAlphaFactor()

```
void AisSquareWaveVoltammetryElement::setAlphaFactor ( \mbox{double } alphaFactor)
```

This is an **optional** parameter. If nothing is set, the device will use the default value of 75.

Parameters

alphaFactor the value for the alphaFactor ranges from 0 to 100.

16.23.2.16 setApproxMaxCurrent()

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

This 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.
------------------	---

16.23.2.17 setAutoRange()

```
void AisSquareWaveVoltammetryElement::setAutoRange ()
```

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

Deprecated This function is deprecated. Use setApproxMaxCurrent() to specify the current range instead.

16.23.2.18 setEndVoltage()

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.
------------	---

16.23.2.19 setEndVoltageVsOCP()

```
void AisSquareWaveVoltammetryElement::setEndVoltageVsOCP ( bool\ endVoltageVsOcp)
```

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.

16.23.2.20 setPulseAmp()

```
void AisSquareWaveVoltammetryElement::setPulseAmp ( \label{eq:condition} \texttt{double} \ pulseAmp)
```

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	e in volts.
---	-------------

16.23.2.21 setPulseFreq()

```
void AisSquareWaveVoltammetryElement::setPulseFreq ( \label{eq:condition} \mbox{double } pulseFreq)
```

Parameters

16.23.2.22 setQuietTime()

Parameters

quietTime	The quiet time duration to set in seconds.
-----------	--

16.23.2.23 setQuietTimeSamplingInterval()

Parameters

16.23.2.24 setStartVoltage()

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

Parameters

startVoltage	the value of the start voltage in volts
olarivorlago	the raide of the clart restage in reste

16.23.2.25 setStartVoltageVsOCP()

```
\label{thm:cond} \mbox{void AisSquareWaveVoltammetryElement::setStartVoltageVsOCP (} \\ \mbox{bool } startVoltageVsOcp)
```

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.

16.23.2.26 setVStep()

```
void AisSquareWaveVoltammetryElement::setVStep ( \mbox{double } vStep) \label{eq:voltam}
```

Parameters

Note

Regardless of vStep's sign, the device will determine the step direction based on the start and end voltage.

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

· AisSquareWaveVoltammetryElement.h

16.24 AisStaircasePotentialVoltammetryElement Class Reference

AisStaircasePotentialVoltammetryElement class represents an element for staircase potential voltammetry experiments. It inherits from AisAbstractElement.

```
#include <AisStaircasePotentialVoltammetryElement.h>
#include <AisStaircasePotentialVoltammetryElement.h>
```

Public Member Functions

&other)

AisStaircasePotentialVoltammetryElement (double startVoltage, double firstVoltageLimit, double second
 — VoltageLimit, double endVoltage, double stepSize, double stepDuration, double samplingInterval)

Constructs an AisStaircasePotentialVoltammetryElement with specified parameters.

- AisStaircasePotentialVoltammetryElement (const AisStaircasePotentialVoltammetryElement &other)
- Copy constructor for AisStaircasePotentialVoltammetryElement.

 AisStaircasePotentialVoltammetryElement & operator= (const AisStaircasePotentialVoltammetryElement

Assignment operator for AisStaircasePotentialVoltammetryElement.

• ~AisStaircasePotentialVoltammetryElement () override

Destructor for AisStaircasePotentialVoltammetryElement.

· QString getName () const override

Gets the name of the element.

QStringList getCategory () const override

Gets the category of the element.

double getQuietTime () const

Gets the quiet time duration.

void setQuietTime (double quietTime)

Sets the quiet time duration.

double getQuietTimeSamplingInterval () const

gets the quiet time sampling interval.

void setQuietTimeSamplingInterval (double quietTimeSamplingInterval)

Sets the quiet time sampling interval.

double getStartVoltage () const

Gets the starting voltage.

void setStartVoltage (double startVoltage)

Sets the starting voltage.

• bool isStartVoltageVsOCP () const

Checks if the starting voltage is with respect to the open circuit mode.

void setStartVoltageVsOCP (bool startVsOCP)

Sets whether the starting voltage is with respect to the open circuit mode.

• double getEndVoltage () const

Gets the ending voltage.

void setEndVoltage (double endVoltage)

Sets the ending voltage.

bool isEndVoltageVsOCP () const

Checks if the ending voltage is with respect to the open circuit mode.

void setEndVoltageVsOCP (bool endVsOCP)

Sets whether the ending voltage is with respect to the open circuit mode.

· double getFirstVoltageLimit () const

Gets the first voltage limit.

void setFirstVoltageLimit (double firstVoltageLimit)

Sets the first voltage limit.

bool isFirstVoltageLimitVsOCP () const

Checks if the first voltage limit is with respect to the open circuit mode.

void setFirstVoltageLimitVsOCP (bool firstVoltageLimitVsOCP)

Sets whether the first voltage limit is with respect to the open circuit mode.

double getSecondVoltageLimit () const

Gets the second voltage limit.

void setSecondVoltageLimit (double secondVoltageLimit)

Sets the second voltage limit.

bool isSecondVoltageLimitVsOCP () const

Checks if the second voltage limit is with respect to the open circuit mode.

void setSecondVoltageLimitVsOCP (bool secondVoltageLimitVsOCP)

Sets whether the second voltage limit is with respect to the open circuit mode.

• double getStepSize () const

Gets the potential step size.

void setStepSize (double stepSize)

Sets the potential step size.

• double getStepDuration () const

Gets the potential step duration.

void setStepDuration (double stepDuration)

Sets the potential step duration.

• double getSamplingInterval () const

Gets the potential sampling interval.

· void setSamplingInterval (double samplingInterval)

Sets the potential sampling interval.

bool isAutorange () const

Checks if the experiment should autorange the current.

void setAutorange ()

Enables autorange for the experiment.

double getApproxMaxCurrent () const

Gets the approximate maximum current.

void setApproxMaxCurrent (double approxMaxCurrent)

Sets the approximate maximum current.

unsigned int getNumberOfCycles ()

get the value set for the number of cycles

void setNumberOfCycles (unsigned int cycles)

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

16.24.1 Constructor & Destructor Documentation

16.24.1.1 AisStaircasePotentialVoltammetryElement() [1/2]

Parameters

startVoltage	The starting voltage in volts.
firstVoltageLimit	The first voltage limit in volts.
secondVoltageLimit	The second voltage limit in volts.
endVoltage	The ending voltage in volts.
stepSize	The potential step size in volts.
stepDuration	The potential step duration in seconds.
samplingInterval	The potential sampling interval in seconds.

16.24.1.2 AisStaircasePotentialVoltammetryElement() [2/2]

```
AisStaircasePotentialVoltammetryElement::AisStaircasePotentialVoltammetryElement (
const AisStaircasePotentialVoltammetryElement & other)
```

Parameters

other	The AisStaircasePotentialVoltammetryElement to copy.
-------	--

16.24.2 Member Function Documentation

16.24.2.1 getApproxMaxCurrent()

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

Returns

The approximate maximum current.

16.24.2.2 getCategory()

 $\verb"QStringList AisStaircasePotentialVoltammetryElement::getCategory () const [override]$

Returns

The category of the element.

16.24.2.3 getEndVoltage()

double AisStaircasePotentialVoltammetryElement::getEndVoltage () const

Returns

The ending voltage in volts.

16.24.2.4 getFirstVoltageLimit()

 $\verb|double AisStaircasePotentialVoltammetryElement::getFirstVoltageLimit () const$

Returns

The first voltage limit in volts.

16.24.2.5 getName()

QString AisStaircasePotentialVoltammetryElement::getName () const [override]

Returns

The name of the element.

16.24.2.6 getNumberOfCycles()

unsigned int AisStaircasePotentialVoltammetryElement::getNumberOfCycles ()

Returns

the number of cycles set.

16.24.2.7 getQuietTime()

double AisStaircasePotentialVoltammetryElement::getQuietTime () const

Returns

The quiet time duration in seconds.

16.24.2.8 getQuietTimeSamplingInterval()

 $\verb|double AisStaircasePotentialVoltammetryElement::getQuietTimeSamplingInterval () | constaircasePotentialVoltammetryElement::getQuietTimeSamplingInterval () | constaircasePotentialVoltammetryElementr$

Returns

samplingInterval The quiet time sampling interval to set in seconds.

16.24.2.9 getSamplingInterval()

 $\verb|double AisStaircasePotentialVoltammetryElement::getSamplingInterval () | constaircasePotentialVoltammetryElement::getSamplingInterval () | constaircasePotentialVoltametryElement::getSamplingInterval () | constairc$

Returns

The potential sampling interval in seconds.

16.24.2.10 getSecondVoltageLimit()

 $\verb|double AisStaircasePotentialVoltammetryElement::getSecondVoltageLimit () constitution of the constitut$

Returns

The second voltage limit in volts.

16.24.2.11 getStartVoltage()

double AisStaircasePotentialVoltammetryElement::getStartVoltage () const

Returns

The starting voltage in volts.

16.24.2.12 getStepDuration()

double AisStaircasePotentialVoltammetryElement::getStepDuration () const

Returns

The potential step duration in seconds.

16.24.2.13 getStepSize()

double AisStaircasePotentialVoltammetryElement::getStepSize () const

Returns

The potential step size in volts.

16.24.2.14 isAutorange()

bool AisStaircasePotentialVoltammetryElement::isAutorange () const

Returns

True if autorange is enabled, false otherwise.

16.24.2.15 isEndVoltageVsOCP()

 $\verb|bool AisStaircasePotentialVoltammetryElement:: is \verb|EndVoltageVsOCP| () const| \\$

Returns

True if the ending voltage is with respect to the open circuit mode, false otherwise.

16.24.2.16 isFirstVoltageLimitVsOCP()

 $\verb|bool AisStaircasePotentialVoltammetryElement:: is First Voltage Limit Vs OCP () constitution of the property of the proper$

Returns

True if the first voltage limit is with respect to the open circuit mode, false otherwise.

16.24.2.17 isSecondVoltageLimitVsOCP()

bool AisStaircasePotentialVoltammetryElement::isSecondVoltageLimitVsOCP () const

Returns

True if the second voltage limit is with respect to the open circuit mode, false otherwise.

16.24.2.18 isStartVoltageVsOCP()

 $\verb|bool AisStaircasePotentialVoltammetryElement:: is StartVoltageVsOCP () constant to the property of the pro$

Returns

True if the starting voltage is with respect to the open circuit mode, false otherwise.

16.24.2.19 operator=()

 $\label{lem:aisStaircasePotentialVoltammetryElement \& AisStaircasePotentialVoltammetryElement:: operator = (const AisStaircasePotentialVoltammetryElement \& other)$

Parameters

other The AisStaircasePotentialVoltammetryElement to assign.

Returns

Reference to this AisStaircasePotentialVoltammetryElement.

16.24.2.20 setApproxMaxCurrent()

 $\label{thm:cond} \mbox{void AisStaircasePotentialVoltammetryElement::setApproxMaxCurrent (} \\ \mbox{double } approxMaxCurrent)$

Parameters

approxMaxCurrent	The approximate maximum current to set.
------------------	---

16.24.2.21 setEndVoltage()

void AisStaircasePotentialVoltammetryElement::setEndVoltage (${\tt double} \ endVoltage)$

Parameters

16.24.2.22 setEndVoltageVsOCP()

```
\label{lem:condition} \mbox{void AisStaircasePotentialVoltammetryElement::} \mbox{setEndVoltageVsOCP} \ ( \mbox{bool } \mbox{endVsOCP})
```

Parameters

endVsOCP	True to set the ending voltage with respect to the open circuit mode, false otherwise.	
011410001	indo to bot the origing voltage with respect to the open endat mede, lake other wice.	П

16.24.2.23 setFirstVoltageLimit()

Parameters

16.24.2.24 setFirstVoltageLimitVsOCP()

```
\label{thm:composition} \begin{tabular}{ll} void AisStaircasePotentialVoltammetryElement::setFirstVoltageLimitVsOCP ( bool $firstVoltageLimitVsOCP)$ \end{tabular}
```

Parameters

16.24.2.25 setNumberOfCycles()

Parameters

cycles	the number of cycles to set
--------	-----------------------------

16.24.2.26 setQuietTime()

```
\begin{tabular}{ll} \begin{tabular}{ll} void AisStaircasePotentialVoltammetryElement::setQuietTime ( \\ \begin{tabular}{ll} double & quietTime) \end{tabular}
```

Parameters

auietTime	The quiet time duration to set in seconds.
9410111110	The quiet time duration to oot in occoride.

16.24.2.27 setQuietTimeSamplingInterval()

```
\label{thm:cond} \mbox{void AisStaircasePotentialVoltammetryElement::setQuietTimeSamplingInterval (} \\ \mbox{double } quietTimeSamplingInterval)
```

Parameters

quietTimeSamplingInterval The quiet time sampling interval to set in seconds.

16.24.2.28 setSamplingInterval()

Parameters

16.24.2.29 setSecondVoltageLimit()

 $\label{thm:cond} \mbox{void AisStaircasePotentialVoltammetryElement::setSecondVoltageLimit (} \\ \mbox{double } secondVoltageLimit)$

Parameters

secondVoltageLimit	The second voltage limit to set in volts.
--------------------	---

16.24.2.30 setSecondVoltageLimitVsOCP()

 $\label{thm:condvoltageLimitVsOCP} \mbox{ void AisStaircasePotentialVoltammetryElement::setSecondVoltageLimitVsOCP)} \mbox{ } \mbox{ bool } secondVoltageLimitVsOCP)$

Parameters

secondVoltageLimitVsOCP	True to set the second voltage limit with respect to the open circuit mode, false
	otherwise.

16.24.2.31 setStartVoltage()

```
\begin{tabular}{ll} \begin{tabular}{ll} void AisStaircasePotentialVoltammetryElement::setStartVoltage ( \\ \begin{tabular}{ll} double & startVoltage) \end{tabular}
```

Parameters

16.24.2.32 setStartVoltageVsOCP()

```
\label{thm:condition} \mbox{void AisStaircasePotentialVoltammetryElement::setStartVoltageVsOCP (} \\ \mbox{bool } startVsOCP) \mbox{}
```

Parameters

startVsOCP True to set the starting voltage with respect to the open circuit mode, false other	vise.
--	-------

16.24.2.33 setStepDuration()

```
\label{thm:condition} \mbox{void AisStaircasePotentialVoltammetryElement::setStepDuration (} \\ \mbox{double } stepDuration)
```

Parameters

16.24.2.34 setStepSize()

```
void AisStaircasePotentialVoltammetryElement::setStepSize ( {\tt double} \ stepSize)
```

Parameters

otential step size to set in volts.

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

· AisStaircasePotentialVoltammetryElement.h

16.25 AisSteppedCurrentElement Class Reference

A class representing an experiment to apply the stepped current.

```
#include <AisSteppedCurrentElement.h>
#include <AisSteppedCurrentElement.h>
```

Public Member Functions

 AisSteppedCurrentElement (double startCurrent, double endCurrent, double stepSize, double stepDuration, double samplingInterval)

Constructs a Stepped Current element.

AisSteppedCurrentElement (const AisSteppedCurrentElement &)

copy constructor for the AisSteppedCurrentElement object.

AisSteppedCurrentElement & operator= (const AisSteppedCurrentElement &)

overload equal to operator for the AisSteppedCurrentElement 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 getEndCurrent () const

Gets the ending current value for the stepped experiment.

• void setEndCurrent (double iEnd)

Sets the ending current value for the stepped experiment.

double getStepSize () const

Gets the size of each current step in the stepped experiment.

• void setStepSize (double iStep)

Sets the size of each current step in the stepped experiment.

double getStartCurrent () const

Gets the starting current value for the stepped experiment.

void setStartCurrent (double iStart)

Sets the starting current value for the stepped experiment.

• double getStepDuration () const

Gets the duration of each current step in the stepped experiment.

void setStepDuration (double tStep)

Sets the duration of each current step in the stepped experiment.

• 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.

double getApproxMinVoltage () const

get the value set for the expected minimum voltage.

• void setApproxMinVoltage (double approxMinVoltage)

set minimum voltage expected, for manual voltage range selection.

16.25.1 Constructor & Destructor Documentation

16.25.1.1 AisSteppedCurrentElement()

This constructor initializes the Stepped Current element with the specified parameters.

Parameters

startCurrent	The initial current value in amperes.
endCurrent	The final current value in amperes.
stepSize	The size of each current step in amperes.
stepDuration	The duration of each current step in seconds.
samplingInterval	The data sampling interval value in seconds.

16.25.2 Member Function Documentation

16.25.2.1 getApproxMaxVoltage()

double AisSteppedCurrentElement::getApproxMaxVoltage () const

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.

16.25.2.2 getApproxMinVoltage()

double AisSteppedCurrentElement::getApproxMinVoltage () const

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.

16.25.2.3 getCategory()

QStringList AisSteppedCurrentElement::getCategory () const [override]

Returns

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

16.25.2.4 getEndCurrent()

double AisSteppedCurrentElement::getEndCurrent () const

Returns

The ending current value in amperes.

16.25.2.5 getName()

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

Returns

The name of the element: "SteppedCurrent".

16.25.2.6 getSamplingInterval()

```
double AisSteppedCurrentElement::getSamplingInterval () const
```

Returns

the data sampling interval value in seconds.

16.25.2.7 getStartCurrent()

double AisSteppedCurrentElement::getStartCurrent () const

Returns

The starting current value in amperes.

16.25.2.8 getStepDuration()

```
double AisSteppedCurrentElement::getStepDuration () const
```

Returns

The duration of each current step in seconds.

16.25.2.9 getStepSize()

```
double AisSteppedCurrentElement::getStepSize () const
```

Returns

The size of each current step in amperes.

16.25.2.10 setApproxMaxVoltage()

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

Parameters

16.25.2.11 setApproxMinVoltage()

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

Parameters

16.25.2.12 setEndCurrent()

```
void AisSteppedCurrentElement::setEndCurrent ( \label{eq:condition} \mbox{double } i\mbox{\it End})
```

Parameters

iEnd	The ending current value in amperes.
------	--------------------------------------

16.25.2.13 setSamplingInterval()

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

Parameters

16.25.2.14 setStartCurrent()

Parameters

iStart The starting current value in amperes.

16.25.2.15 setStepDuration()

```
void AisSteppedCurrentElement::setStepDuration ( \label{eq:condition} \mbox{double } tStep)
```

Parameters

tStep The duration of each current step in seconds.

16.25.2.16 setStepSize()

Parameters

iStep The size	of each current step in amperes.
----------------	----------------------------------

Note

Regardless of iStep's sign, the device will determine the step direction based on the start and end current.

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

· AisSteppedCurrentElement.h

16.26 AisSteppedVoltageElement Class Reference

A class representing an experiment to apply the stepped volatge.

```
#include <AisSteppedVoltageElement.h>
#include <AisSteppedVoltageElement.h>
```

Public Member Functions

AisSteppedVoltageElement (double startVoltage, double endVoltage, double voltageStep, double voltage
 — StepDuration, double samplingInterval)

Constructor for the AisSteppedVoltageElement element.

AisSteppedVoltageElement (const AisSteppedVoltageElement &other)

Copy constructor for the AisSteppedVoltageElement object.

• AisSteppedVoltageElement & operator= (const AisSteppedVoltageElement &other)

Overloaded assignment operator for the AisSteppedVoltageElement object.

∼AisSteppedVoltageElement () override

Destructor for the AisSteppedVoltageElement 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 starting voltage for the experiment.

• void setStartVoltage (double vStart)

Set the starting voltage for the experiment.

• double getEndVoltage () const

Get the ending voltage for the experiment.

void setEndVoltage (double vEnd)

Set the ending voltage for the experiment.

• double getStepSize () const

Get the voltage step for each iteration.

void setStepSize (double vStep)

Set the voltage step for each iteration.

• double getStepDuration () const

Get the time step for each iteration.

• void setStepDuration (double duration)

Set the duration of each step.

• double getSamplingInterval () const

Get the data sampling interval.

void setSamplingInterval (double samplingInterval)

Set the data sampling interval.

double getApproxMaxCurrent () const

Get the approximate maximum current.

void setApproxMaxCurrent (double approxMaxCurrent)

Set the approximate maximum current.

bool isStartVoltageVsOCP () const

Check if the experiment starts with the open circuit potential.

void setStartVoltageVsOCP (bool startVsOCP)

Set whether the experiment starts with the open circuit potential.

• bool isEndVoltageVsOCP () const

Check if the experiment ends with the open circuit potential.

void setEndVoltageVsOCP (bool endVsOCP)

Set whether the experiment ends with the open circuit potential.

• bool isAutoRange () const

Check if current autoranging is enabled.

• void setCurrentAutorange ()

Enable current autoranging for the experiment.

16.26.1 Constructor & Destructor Documentation

16.26.1.1 AisSteppedVoltageElement() [1/2]

This constructor initializes the AisSteppedVoltageElement element with the specified parameters.

Parameters

startVoltage	The initial voltage value in volts.
endVoltage	The final voltage value in volts.
voltageStep	The size of each voltage step in volts.
voltageStepDuration	The duration of each voltage step in seconds.
samplingInterval	The data sampling interval value in seconds.

16.26.1.2 AisSteppedVoltageElement() [2/2]

Parameters

other The AisSteppedVoltageElement object to be copied.

16.26.2 Member Function Documentation

16.26.2.1 getApproxMaxCurrent()

double AisSteppedVoltageElement::getApproxMaxCurrent () const

Returns

The approximate maximum current in Amps.

16.26.2.2 getCategory()

QStringList AisSteppedVoltageElement::getCategory () const [override]

Returns

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

16.26.2.3 getEndVoltage()

double AisSteppedVoltageElement::getEndVoltage () const

Returns

The ending voltage in volts.

16.26.2.4 getName()

QString AisSteppedVoltageElement::getName () const [override]

Returns

The name of the element: "Stepped Voltage".

16.26.2.5 getSamplingInterval()

double AisSteppedVoltageElement::getSamplingInterval () const

Returns

The data sampling interval in seconds.

16.26.2.6 getStartVoltage()

double AisSteppedVoltageElement::getStartVoltage () const

Returns

The starting voltage in volts.

16.26.2.7 getStepDuration()

double AisSteppedVoltageElement::getStepDuration () const

Returns

The time step in seconds.

16.26.2.8 getStepSize()

double AisSteppedVoltageElement::getStepSize () const

Returns

The voltage step in volts.

16.26.2.9 isAutoRange()

bool AisSteppedVoltageElement::isAutoRange () const

Returns

True if current autoranging is enabled, false otherwise.

16.26.2.10 isEndVoltageVsOCP()

 $\verb|bool AisSteppedVoltageElement:: is \verb|EndVoltageVsOCP| () const|\\$

Returns

True if the experiment ends with open circuit potential, false otherwise.

16.26.2.11 isStartVoltageVsOCP()

bool AisSteppedVoltageElement::isStartVoltageVsOCP () const

Returns

True if the experiment starts with open circuit potential, false otherwise.

16.26.2.12 operator=()

Parameters

other The AisSteppedVoltageElement object to be assigned.

Returns

A reference to the assigned AisSteppedVoltageElement object.

16.26.2.13 setApproxMaxCurrent()

Parameters

16.26.2.14 setEndVoltage()

```
void AisSteppedVoltageElement::setEndVoltage ( \label{eq:cond} \mbox{double $vEnd$})
```

Parameters

vEnd	The ending voltage in volts.
------	------------------------------

16.26.2.15 setEndVoltageVsOCP()

```
\label{total_continuous} \mbox{void AisSteppedVoltageElement::setEndVoltageVsOCP (} \\ \mbox{bool } endVsOCP) \mbox{}
```

Parameters

endVsOCP	True to end with open circuit potential, false otherwise.
----------	---

16.26.2.16 setSamplingInterval()

Parameters

samplingInterval The data sampling interval in seconds.

16.26.2.17 setStartVoltage()

```
void AisSteppedVoltageElement::setStartVoltage ( \label{eq:condition} \mbox{double } vStart)
```

Parameters

16.26.2.18 setStartVoltageVsOCP()

```
void AisSteppedVoltageElement::setStartVoltageVsOCP ( bool\ startVsOCP)
```

Parameters

16.26.2.19 setStepDuration()

```
\label{lement::setStepDuration} % \begin{center} \begin{center}
```

Parameters

duration The step duration in seconds.	_
--	---

16.26.2.20 setStepSize()

```
void AisSteppedVoltageElement::setStepSize ( \label{eq:condition} \mbox{double } vStep)
```

Parameters

vStep	The voltage step in volts.
-------	----------------------------

Note

Regardless of vStep's sign, the device will determine the step direction based on the start and end voltage.

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

• AisSteppedVoltageElement.h

Chapter 17

Examples

17.1 advancedControlFlow.cpp

This example shows how we can setup an advanced control flow for a sequence of experiments. For simplicity we will be using an incremented integer as an external condition to control the workflow. This logic can be replaced with any sort of logic pertaining to external instruments, sensors, or any other conditions.

```
#include "AisExperiment.h"
#include "AisDeviceTracker.h"
#include "AisInstrumentHandler.h"
#include "experiments/builder_elements/AisConstantCurrentElement.h"
#include "experiments/builder_elements/AisConstantPotElement.h"
#include <QCoreApplication>
#include <QTimer>
#include <qdebug.h>
int main()
        // Environment Setup
        char** test = nullptr;
        QCoreApplication a(args, test); // this creates a non-GUI Qt application
         // constructing a constant potential element with required arguments
        AisConstantPotElement cvElement (
                 5, // voltage: 1v
1, // sampling interval: 1s
                 10 // duration: 10s
         // constructing a constant current element with required arguments
        AisConstantCurrentElement ccElement (
                 0.002, // current: 2mA
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
        auto experimentB = std::make_shared<AisExperiment>(); // create a second experiment
        \hbox{experimentB--} \hbox{appendElement (ccElement, 1); // append to experimentB, the created CC element and set it to the composition of the compositi
        auto experimentC = std::make_shared<AisExperiment>(); // create a third experiment
        experimentC->appendElement(cvElement, 2); // append to experimentC, the created CV element and set it to
             run 2 times
        \star 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.
        \star The following lambda function creates a logic and assigns it to the given handler.
        * We will call this function after a newDeviceConnected signal has been emitted and a handler has been
        * The logic controls the workflow as follows:
```

```
\star # Start the first timer
\star # once the timer times out, start Experiment A
\star # once Experiment A completes, start the second timer
\star # once the second timer times out, start Experiment B
* # start a third timer to stop Experiment B early
* # once the third timer out, stop Experiment B
* # start a fourth timer
* # once the fourth timer times out, start Experiment C
* # once Experiment C completes, start Experiment B
// Lambda function
auto createLogic = [&](const AisInstrumentHandler* handler) {
    QTimer* timer1 = new QTimer(); // the first timer is used in lieu of the first external condition
    timer1->setSingleShot(true);
    timer1->start(1000);
   OObject::connect(timer1, &OTimer::timeout, [=]() {
        qDebug() « "Initial condition met. Starting Experiment A ";
        handler->uploadExperimentToChannel(0, experimentA);
        handler->startUploadedExperiment(0);
        // once the first experiment is completed (Experiment A), start the next experiment (Experiment
  B).
        // this signal will be emitted for any experiment not just A so, we will track of the sequence
  with experimentStep
        // once an experiment has completed or has been stopped, continue to the next experimentStep
        QObject::connect(handler, &AisInstrumentHandler::experimentStopped, [&](uint8_t channel) {
            static int experimentStep = 0;
qDebug() « "Experiment Step " « experimentStep « " Completed";
            experimentStep++; //increment the experiment step
             if (experimentStep == 1) {
                 // Wait for external start condition
                 QTimer* timer = new QTimer(); // the timer is used in lieu of an external condition
                 timer->setSingleShot(true);
                timer->start(10000); // when this timer times out, the next experiment (Experiment B)
  will start
                 // Create an external condition that will stop the upcoming experiment early
                QTimer* StopEarlyTimer = new QTimer();
                 StopEarlyTimer->setSingleShot(true);
                QObject::connect(StopEarlyTimer, &QTimer::timeout, [&]() {
                    qDebug() « "External early stop condition met";
handler->stopExperiment(0); // Once the external condition is met, experiment B will
  stop, and the experimentCompleted signal will be emitted
                });
                QObject::connect(timer, &QTimer::timeout, [&, StopEarlyTimer]() {
                    gDebug() « "External condition met, starting experiment B";
                     handler->uploadExperimentToChannel(0, experimentB); // start Experiment B
                    handler->startUploadedExperiment(0);
                    StopEarlyTimer->start(2000);
                 });
            } else if (experimentStep == 2) {
                QTimer* timer = new QTimer(); // the timer is used in lieu of an external condition
                 timer->setSingleShot(true);
                timer->start(10000); // when this timer times out, the next experiment (Experiment C)
  will start
                QObject::connect(timer, &QTimer::timeout, [&]() {
                    qDebug() « "External condition met, starting Experiment C ";
                    handler->uploadExperimentToChannel(0, experimentC); // start Experiment C
                    handler->startUploadedExperiment(0);
                 });
            } else if (experimentStep == 3) {
                QTimer* timer = new QTimer(); // the timer is used in lieu of an external condition
                 timer->setSingleShot(true);
                timer->start(10000); // when this timer times out, the next experiment (Experiment B)
  will start
                QObject::connect(timer, &QTimer::timeout, [&]() {
                    qDebug() « "External condition met, starting Experiment B ";
handler->uploadExperimentToChannel(0, experimentB); // start Experiment B
                    handler->startUploadedExperiment(0);
                });
       });
   });
// this is a signal-slot connection where the slot assigns the logic to the device handler when
 {\tt newDeviceConnected\ signal\ is\ emitted.}
auto tracker = AisDeviceTracker::Instance(); // create a tracker that tracks connected devices
QObject::connect(tracker, &AisDeviceTracker::newDeviceConnected, &a, [&](const QString& deviceName) {
    auto& handler = tracker->getInstrumentHandler(deviceName);
    createLogic(&handler); // controlling experiments is to be done only after a device handler has been
```

17.2 advancedExperiment.cpp

```
#include "AisDeviceTracker.h"
#include "AisExperiment.h"
#include "AisInstrumentHandler.h"
#include "experiments/builder_elements/AisOpenCircuitElement.h"
#include "experiments/builder_elements/AisConstantPotElement.h"
#include "experiments/builder_elements/AisEISGalvanostaticElement.h"
#include "experiments/builder_elements/AisConstantCurrentElement.h
#include <QCoreApplication>
#include <QDebug>
// Define relevant device information, for easy access
#define COMPORT "COM5"
#define CHANNEL 0
#define INSTRUMENT_NAME "PLus1366"
int main()
    char** test = nullptr;
    int args;
    QCoreApplication a(args, test);
    auto tracker = AisDeviceTracker::Instance();
   bool success = true;
   auto customExperiment = std::make_shared<AisExperiment>();
    AisOpenCircuitElement ocpElement(1, 10);
    success &= customExperiment->appendElement(ocpElement);
    int voltage = 0;
    for (int i = 0; i < 4; i++) {
        AisConstantPotElement cvElement (voltage, 0.1, 5);
        success &= customExperiment->appendElement(cvElement, 1);
        voltage += 0.1; // Adding 100 mV
    AisExperiment eisSubExperiment;
    AisEISGalvanostaticElement galvEISElement(10, 10000, 10, 0.01, 0.1);
    AisOpenCircuitElement ocpElement2(1, 5);
    success &= eisSubExperiment.appendElement(galvEISElement, 1);
    success &= eisSubExperiment.appendElement(ocpElement2, 1);
    success &= customExperiment->appendSubExperiment(eisSubExperiment, 3);
    AisConstantCurrentElement ccElement(0.1, 1, 10);
    success &= customExperiment->appendElement(ccElement, 2);
    if (!success) {
        qDebug() « "Error building experiment";
    auto connectSignals = [=](const AisInstrumentHandler& handler) {
        QObject::connect(&handler, &AisInstrumentHandler::activeDCDataReady, [=] (uint8_t channel, const
      AisDCData& data) {
      qDebug() « "Timestamp: " « data.timestamp « " Current: " « data.current « " Voltage: " «
data.workingElectrodeVoltage « " CE Voltage : " « data.counterElectrodeVoltage;
        QObject::connect(&handler, &AisInstrumentHandler::experimentNewElementStarting, [=] (uint8_t channel,
      const AisExperimentNode& info) {
```

```
qDebug() « "New element starting: " « info.stepName;
    QObject::connect(&handler, &AisInstrumentHandler::experimentStopped, [=] (uint8_t channel, const
  QString& reason) {
       qDebug() « "Experiment Stopped Signal " « channel « "Reason : " « reason;
   Object::connect(&handler, &AisInstrumentHandler::deviceError, [=](uint8_t channel, const QString&
  error) {
       qDebug() « "Device Error: " « error;
   });
};
// When device is connected, setup connections, and upload/start the experiment
QObject::connect(tracker, &AisDeviceTracker::newDeviceConnected, [=](const QString& deviceName) {
    qDebug() « "New Device Connected: " « deviceName;
    auto& handler = tracker->getInstrumentHandler(INSTRUMENT_NAME);
   connectSignals(handler);
   AisErrorCode error = handler.uploadExperimentToChannel(CHANNEL, customExperiment);
        qDebug() « error.message();
        return 0;
    error = handler.startUploadedExperiment(CHANNEL);
        qDebug() « error.message();
        return 0;
});
AisErrorCode error = tracker->connectToDeviceOnComPort(COMPORT);
if (error != error.Success) {
    qDebug() « error.message();
    return 0;
return a.exec();
```

17.3 basicExperiment.cpp

```
#include "AisDeviceTracker.h"
#include "AisExperiment.h'
#include "AisInstrumentHandler.h"
#include "experiments/builder_elements/AisConstantPotElement.h"
#include <QCoreApplication>
#include <QDebug>
// Define relevant device information, for easy access
#define COMPORT "COM1"
#define CHANNEL 0
int main()
   char** test = nullptr;
   int args;
   OCoreApplication a(args, test);
   auto tracker = AisDeviceTracker::Instance();
           Voltage = 1V, Sampling Interval = 1s, Duration = 30s
   AisConstantPotElement cvElement(1, 1, 30);
   auto customExperiment = std::make_shared<AisExperiment>();
   // Append the constant potential element, and tell it to execute that element 1 time
   customExperiment->appendElement(cvElement, 1);
   auto connectSignals = [=](const AisInstrumentHandler& handler) {
       QObject::connect(&handler, &AisInstrumentHandler::activeDCDataReady, [=](uint8_t channel, const
     QObject::connect(&handler, &AisInstrumentHandler::activeACDataReady, [=](uint8_t channel, const
     AisACData& data) { qDebug() « "Timestamp: " « data.timestamp « " Frequency: " « data.frequency « "" «
     data.absoluteImpedance;
       });
```

17.4 dataOutput.cpp 215

```
// Whenever a new node in the element starts, note: some Ais Elements contain multiple logical nodes
    // i.e AisCyclicVoltammatryElement contains 4 nodes for each linear segment of each cycle plus a
 quiet time node if enabled
   // So this lambda would be executed atleast 4 times for each cycle
   QObject::connect(&handler, &AisInstrumentHandler::experimentNewElementStarting, [=] (uint8_t channel,
 const AisExperimentNode& info) {
       qDebug() « "New element starting: " « info.stepName;
    // Whenever an experiment completes or is manually stopped, this will execute
   QObject::connect(&handler, &AisInstrumentHandler::experimentStopped, [=](uint8_t channel, const
 QString& reason) {
       qDebug() « "Experiment Stopped Signal " « channel « "Reason : " « reason;
   QObject::connect(&handler, &AisInstrumentHandler::deviceError, [=](uint8_t channel, const QString&
  error) {
       qDebug() « "Device Error: " « error;
   });
QObject::connect(tracker, &AisDeviceTracker::deviceDisconnected, [=] (const QString& deviceName) {
   qDebug() « "New Device Connected: " « deviceName;
connectSignals(handler);
   AisErrorCode error = handler.uploadExperimentToChannel(CHANNEL, customExperiment);
   if (error) {
       gDebug() « error.message();
       return:
   // Start the previously uploaded experiment on the same channel
   error = handler.startUploadedExperiment(CHANNEL);
   // Exit the application if there is any error starting the experiment
   if (error) {
       qDebug() « error.message();
});
AisErrorCode error = tracker->connectToDeviceOnComPort(COMPORT);
if (error != error.Success) {
   qDebug() « error.message();
   return 0;
// Returning a.axec() executes the event loop, which continues running until the application is exited
return a.exec();
```

17.4 dataOutput.cpp

This example shows how to capture data from the device, and write it to a CSV file using Qt's QFile and QTextStream classes.

```
#include "AisExperiment.h"
#include "AisDeviceTracker.h"
#include "AisInstrumentHandler.h"
#include "experiments/builder_elements/AisConstantCurrentElement.h"
#include "experiments/builder_elements/AisConstantPotElement.h"
#include <QCoreApplication>
#include <QTimer>
#include <QDebug>
#include <OFile>
#include <QStandardPaths>
// Define relevant device information, for easy access
#define COMPORT "COM1"
#define CHANNEL O
int main()
    char** test = nullptr;
    QCoreApplication a(args, test);
    auto tracker = AisDeviceTracker::Instance();
    // Build the experiment
```

```
AisConstantPotElement cvElement(
          // voltage: 1v
// sampling interval: 1s
   1,
    1.
            // duration: 30s
    30
AisConstantCurrentElement ccElement(
    0.001, // current: 1mA
            // sampling interval: 1s
    60
            // duration: 60s
auto customExperiment = std::make shared<AisExperiment>();
customExperiment->appendElement(ccElement, 1);
customExperiment->appendElement(cvElement, 1);
// Static QString variable to store the current elements file path
static QString filePath;
// Create a lambda funnction to connect signals to the handler
auto connectSignals = [=](const AisInstrumentHandler& handler) {
    QObject::connect(&handler, &AisInstrumentHandler::experimentNewElementStarting, [=] (uint8_t channel,
  const AisExperimentNode& node) {
        // Create a unique file name for each element, with the format
  "stepNumber_stepName_expStartTime.csv"
 static int fileNum = 1;
auto name = "/" + QString::number(fileNum) + "_" + QString::number(node.stepNumber) + "_" +
node.stepName + ".csv";
        filePath = QString(QStandardPaths::writableLocation(QStandardPaths::DesktopLocation)) + name;
        QFile file(filePath);
        if (!file.open(QIODevice::WriteOnly | QIODevice::Text))
            return:
        // Writing headers to file
        QTextStream out(&file);
        « "Working Electrode Voltage,"
« "Current"
            « "\n";
        file.close();
        qDebug() « "New element beginning: " « node.stepName « "step: " « node.stepNumber;
    }):
    QObject::connect(&handler, &AisInstrumentHandler::activeDCDataReady, [=](uint8_t channel, const
  AisDCData& data) {
        qDebug() « "current :" « data.current « " voltage: " « data.workingElectrodeVoltage « "
  \texttt{counter} \ \texttt{electrode} : \ \texttt{"} \ \texttt{``data.counterElectrodeVoltage} \ \texttt{``} \ \texttt{'`timestamp} : \ \texttt{"} \ \texttt{``data.timestamp};
        // Save the DC data to the file created at element beginning
        QFile file(filePath);
        if (!file.open(QIODevice::Append | QIODevice::WriteOnly | QIODevice::Text))
             eturn;
        QTextStream out(&file);
out « data.timestamp « ","
            « data.counterElectrodeVoltage « ","
            « data.workingElectrodeVoltage « ","
            « data.current
            « "\n";
        file.close();
    }):
    // The experiment created only uses DC elements, so this will not be executed.
    // However, ac data can be handled and written to a csv like seen above with dc data
    QObject::connect(&handler, &AisInstrumentHandler::activeACDataReady, [=](uint8_t channel, const
  AisACData& data) {
       qDebug() « data.frequency « "
                                                                                         " « data.phaseAngle;
                                                " « data.absoluteImpedance « "
    QObject::connect(&handler, &AisInstrumentHandler::experimentStopped, [=] (uint8_t channel, const
  QString& reason) {
        qDebug() « "Experiment Completed Signal " « channel « "Reason : " « reason;
    });
};
// When device is connected, setup connections, and upload/start the experiment
QObject::connect(tracker, &AisDeviceTracker::newDeviceConnected, [=](const QString& deviceName) {
    auto& handler = tracker->getInstrumentHandler(deviceName);
    connectSignals(handler):
    AisErrorCode error = handler.uploadExperimentToChannel(0, customExperiment);
        qDebug() « error.message();
        return 0;
```

17.5 firmwareUpdate.cpp

```
#include "AisInstrumentHandler.h"
#include "AisDeviceTracker.h"
#include <OCoreApplication>
#include <QThread>
#include <QDebug>
#define COMPORT "COM1"
int main()
    int args;
    QCoreApplication a(args, nullptr);
    auto tracker = AisDeviceTracker::Instance();
    QObject::connect(tracker, &AisDeviceTracker::firmwareUpdateNotification, [=](const QString& message) {
        qDebug() « message;
        if (message.contains("firmware is updated.")) {
            // Now instrument is ready to go
    });
    // Attempt to connect to the device
    auto error = tracker->connectToDeviceOnComPort(COMPORT);
    if (error == AisErrorCode::FirmwareNotSupported)
        error = tracker->updateFirmwareOnComPort(COMPORT);
        // Some other error occured
        if (error != AisErrorCode::Success) {
            qDebug() « "Error: " « error.message();
    } else if (error != AisErrorCode::Success) {
   qDebug() « "Error: " « error.message();
        qDebug() « "Device firmware is up to date";
    return a.exec();
```

17.6 linkedChannels.cpp

This example will show you how linked channels can be used to combine the multiple channels on a device, in order to amplify the output for a single experiment. AisInstrumentHandler::setLinkedChannels MUST be called before each experiment that uses paralleled channels. Once linked, these channels must be controlled by ONLY the master channel, which is returned by the AlsInstrumentHandler::setLinkedChannels function.

Note

This feature is only available on Cycler models.

```
#include "AisDeviceTracker.h"
#include "AisExperiment.h"
#include "AisInstrumentHandler.h"
#include "experiments/builder_elements/AisConstantCurrentElement.h"
#include <OCoreApplication>
#include <QDebug>
// Define relevant device information, for easy access
#define COMPORT "COM1"
int main()
    char** test = nullptr;
    int args;
    QCoreApplication a(args, test);
    auto tracker = AisDeviceTracker::Instance();
    // Create an experiment
    AisConstantCurrentElement ccElement (10, 1, 30);
    auto customExperiment = std::make_shared<AisExperiment>();
    customExperiment->appendElement(ccElement, 1);
    auto connectSignals = [=](const AisInstrumentHandler& handler) {
        QObject::connect(&handler, &AisInstrumentHandler::activeDCDataReady, [=](uint8_t channel, const
      AisDCData& data) {
      qDebug() « "Timestamp: " « data.timestamp « " Current: " « data.current « " Voltage: " «
data.workingElectrodeVoltage « " CE Voltage: " « data.counterElectrodeVoltage;
        });
        QObject::connect(&handler, &AisInstrumentHandler::deviceError, [=] (uint8_t channel, const QString&
      error) {
           qDebug() « "Device Error: " « error;
        });
    } ;
    auto& handler = tracker->getInstrumentHandler(deviceName);
        // Here we want to link channels 0 and 1 together, so we pass in a vector of the channels to link
        // It will return which of the channels is the master channel, this should be used to control the
      experiment
        int8_t masterChannel = handler.setLinkedChannels({ 0, 1 });
        connectSignals(handler);
        AisErrorCode error = handler.uploadExperimentToChannel(masterChannel, customExperiment);
        if (error) {
            qDebug() « error.message();
        // Start the previously uploaded experiment on the master channel error = handler.startUploadedExperiment (masterChannel);
        if (error) {
            qDebug() « error.message();
            return 0;
    });
    AisErrorCode error = tracker->connectToDeviceOnComPort(COMPORT);
    if (error != error.Success) {
        qDebug() « error.message();
    return a.exec();
```

17.7 manualExperiment.cpp

```
#include "AisDeviceTracker.h"
#include "AisInstrumentHandler.h"
#include <QCoreApplication>
```

```
#include <QDebug>
#include <QTimer> // For QTimer::singleShot
// Define relevant device information, for easy access
#define COMPORT "COM1"
#define CHANNEL 0
int main()
    char** test = nullptr:
    int args;
    QCoreApplication a(args, test);
    auto tracker = AisDeviceTracker::Instance();
    // Create a lambda function to connect signals to the handler
    auto connectSignals = [=](const AisInstrumentHandler& handler) {
        QObject::connect(&handler, &AisInstrumentHandler::activeDCDataReady, [=] (uint8_t channel, const
      Object::connect(&handler, &AisInstrumentHandler::deviceError, [=] (uint8_t channel, const QString&
      error) {
           qDebug() « "Device Error: " « error;
        QObject::connect(&handler, &AisInstrumentHandler::experimentStopped, [=](uint8_t channel, const
      QString& reason) {
            <code>qDebug()</code> « reason; <code>qDebug()</code> « "Experiment has ended. Closing application.";
            QCoreApplication::quit();
        });
    };
    // Create a lambda function to run the experiment
    auto runExperiment = [=](const AisInstrumentHandler& handler) {
        //The default starting mode is always Open Circuit Potential
        qDebug() « "Starting manual mode at open circuit potential";
        AisErrorCode error = handler.startManualExperiment(CHANNEL);
        if (error != AisErrorCode::Success) {
            qDebug() « error.message();
            QCoreApplication::quit();
        // In this section we connect singleshot QTimers to lambda functions that call our mode changing
      functions.
        // These lambdas are called asynchronously when the QTimers expire.
        // 5 seconds after starting experiment, change to Constant Current at .1A
        QTimer::singleShot(5000, [=, &handler]() {
            qDebug() « "Switching to constant current at .1A";
            AisErrorCode error = handler.setManualModeConstantCurrent(CHANNEL, .1);
if (error != AisErrorCode::Success) {
                qDebug() « error.message();
        // 15 seconds after starting experiment, change to Constant Voltage at 1V
        QTimer::singleShot(15000, [=, &handler]() {
    qDebug() « "Switching to constant voltage at 1V";
            AisErrorCode error = handler.setManualModeConstantVoltage(CHANNEL, 1);
            if (error != AisErrorCode::Success) {
                qDebug() « error.message();
        });
        // 25 seconds after starting experiment, change back into Open Circuit Potential mode.
        QTimer::singleShot(25000, [=, &handler]() {
    qDebug() « "Switching to open circuit potential";
            AisErrorCode error = handler.setManualModeOCP(CHANNEL);
            if (error != AisErrorCode::Success) {
                qDebug() « error.message();
        });
        // Stop the experiment after 30 seconds
        QTimer::singleShot(30000, [=, &handler]() {
    qDebug() « "Stopping experiment";
            if (handler.stopExperiment(CHANNEL) != AisErrorCode::Success) {
                qDebug() « error.message();
        });
    };
    OObject::connect(tracker, &AisDeviceTracker::newDeviceConnected, [=](const OString& deviceName) {
```

```
qDebug() « "New Device Connected: " « deviceName;

// When instrument is connected, grab the handler and setup/start the experiment
auto& handler = tracker->getInstrumentHandler(deviceName);

connectSignals(handler);

runExperiment(handler);

});

AisErrorCode error = tracker->connectToDeviceOnComPort(COMPORT);
if (error != error.Success) {
    qDebug() « error.message();
    return 0;
}

return a.exec();
```

17.8 nonblockingExperiment.cpp

This non blocking example show how we can start and run an experiment, while still being able to process other events. Say there is other logic that needs to be processed while an experiment is running, this example shows how we can continue to process events while our experiment and relavant logic is running. The Admiral Instruments API gives more control of the device 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. For example, you may want to start an experiment, and stop after certain time, and exit the program.

Below is an example for start a manual experiment, stop after 25 s, and exit the program after.

```
#include "AisExperiment.h"
#include "AisDeviceTracker.h"
#include "AisInstrumentHandler.h"
#include "experiments/builder_elements/AisConstantCurrentElement.h"
#include "experiments/builder_elements/AisConstantPotElement.h"
#include <QCoreApplication>
#include <OTimer>
// Define relevant device information, for easy access
#define COMPORT "COM1"
#define CHANNEL 0
#define INSTRUMENT_NAME "Plus2001"
int main()
    char** test = nullptr;
    int args;
    QCoreApplication a(args, test);
    auto tracker = AisDeviceTracker::Instance();
    auto createLogic = [=] (const AisInstrumentHandler* handler) {
        QObject::connect(handler, &AisInstrumentHandler::activeDCDataReady,
                                                                                  [=] (uint8_t channel, const
      AisDCData& data) {
      qDebug() « "Timestamp : " « data.timestamp « " Current : " « data.current « " Voltage: " «
data.workingElectrodeVoltage « " CE Voltage: " « data.counterElectrodeVoltage ;
        Object::connect(handler, &AisInstrumentHandler::activeACDataReady, [=](uint8 t channel, const
      AisACData& data) {
            qDebug() « "Frequency: " « data.frequency « " Absolute Impedance: " « data.absoluteImpedance « "
      Phase Angle" « data.phaseAngle;
        });
        OObject::connect(handler, &AisInstrumentHandler::experimentNewElementStarting, [=] (uint8 t channel,
      const AisExperimentNode& nodeInfo) {
            qDebug() « "New Node beging ";
        QObject::connect(handler, &AisInstrumentHandler::experimentStopped, [=] (uint8_t channel, const
      OString& reason)
            qDebug() « "Experiment Compleletd Signal " « channel « "Reason: " « reason;
    };
    QObject::connect(tracker, &AisDeviceTracker::newDeviceConnected, [=, &a](const QString& deviceName) {
```

17.9 pulseData.cpp 221

```
qDebug() « "New Device Connected: " « deviceName;
    auto& handler = tracker->getInstrumentHandler(INSTRUMENT_NAME);
    createLogic(&handler);
    auto error = handler.startManualExperiment(CHANNEL);
        qDebug() « error.message();
        return;
    error = handler.setManualModeSamplingInterval(CHANNEL, 2);
    if (error) {
        qDebug() « error.message();
    error = handler.setManualModeConstantVoltage(CHANNEL, 2);
    if (error) {
        qDebug() « error.message();
    QTimer::singleShot(25000, [=, &handler]() {
        auto error = handler.stopExperiment(CHANNEL);
        if (error) {
            qDebug() « error.message();
             return;
    });
    // Allow the event loop to process other events while the current experiment is running // isChannelBusy will return true while the channel is running an experiment
    while (handler.isChannelBusy(CHANNEL)) {
        a.processEvents();
    // Process any remaining events
    a.processEvents();
});
auto connectdevices = tracker->connectAllPluggedInDevices();
if (connectdevices == 0)
{
    qDebug() « "No devices connected";
    return 0;
return 0;
```

17.9 pulseData.cpp

This example demonstrates how to use the AisDataManipulator class to calculate advanced parameters from raw DC data recieved from pulse experiments.

```
#include "AisDeviceTracker.h"
#include "AisInstrumentHandler.h"
#include "AisExperiment.h"
#include "AisDataManipulator.h"
#include "experiments/builder_elements/AisDiffPulseVoltammetryElement.h"
#include <QCoreApplication>
#include <QTimer>
#include <QDebug>
// Define relevant device information, for easy access
#define COMPORT "COM1"
#define CHANNEL 0
int main()
    char** test = nullptr;
    int args;
    QCoreApplication a(args, test);
    auto tracker = AisDeviceTracker::Instance();
    // Create the AisDiffPulseVoltammetryElement element and add it to the experiment
    std::shared_ptr<AisExperiment> experiment = nullptr;
AisDiffPulseVoltammetryElement dpv_element(-0.4, 0.5, 0.01, 0.1, 0.1, 0.45);
    dpv_element.setStartVoltageVsOCP(false);
```

```
dpv_element.setEndVoltageVsOCP(false);
dpv_element.setApproxMaxCurrent(0.001);
experiment = std::make_shared<AisExperiment>();
experiment->appendElement(dpv_element, 1);
 // Create an `AisDataManipulator' class for calculating advance parameters.
std::shared_ptr<AisDataManipulator> dataManipulator = std::make_shared<AisDataManipulator>();
dataManipulator->setPulseType(AisPulseType::DifferentialPulse, dpv_element.getPulseWidth(),
  dpv_element.getPulsePeriod());
auto connectSignals = [=](const AisInstrumentHandler* handler) {
    QObject::connect(handler, &AisInstrumentHandler::activeDCDataReady, [=] (uint8_t channel, const
  AisDCData& data) {
        auto utcTime = handler->getExperimentUTCStartTime(0);
        dataManipulator->loadPrimaryData(data);
        if (dataManipulator->isPulseCompleted()) {
            static bool writeheader = true;
            if (writeheader) {
                 aDebua() « "time"
                          « ", Pulse_current"
« ", base_current"
                           « ", diff_current"
                           « ", base_voltage"
                 ", pulse_volatge";
writeheader = false;
 qDebug() « data.timestamp « ", " « dataManipulator->getPulseCurrent() « ", " « dataManipulator->getBaseCurrent() - dataManipulator->getBaseCurrent() « ", " « dataManipulator->getBaseVoltage() « ", " «
  dataManipulator->getPulseVoltage();
    });
    QObject::connect(handler, &AisInstrumentHandler::activeACDataReady, [=] (uint8_t channel, const
  AisACData& data) {
   qDebug() « "channel: " « (int)channel « "frequency :" « data.frequency « " absoluteImpedance: « data.absoluteImpedance « " phaseAngle : " « data.phaseAngle « " timestamp : " « data.timestamp;
    });
    QObject::connect(handler, &AisInstrumentHandler::experimentNewElementStarting, [=] (uint8_t channel,
  const AisExperimentNode& data) {
        qDebug() « "New Node beginning " « data.stepName « " step number " « data.stepNumber « " step
  sub : " « data.substepNumber;
    });
    Object::connect(handler, &AisInstrumentHandler::experimentStopped, [=](uint8 t channel, const
  QString& reason) {
        qDebug() « "Experiment Completed Signal " « channel « "Reason : " « reason;
    QObject::connect(handler, &AisInstrumentHandler::experimentPaused, [=] (uint8_t channel) {
    qDebug() « "Experiment Paused " « channel;
    QObject::connect(handler, &AisInstrumentHandler::experimentResumed, [=](uint8_t channel) {
        qDebug() « "Experiment Resume " « channel;
};
QObject::connect(tracker, &AisDeviceTracker::newDeviceConnected, &a, [=](const QString& deviceName) {
    auto& handler = tracker->getInstrumentHandler(deviceName);
    connectSignals(&handler);
        std::shared_ptr<AisExperiment> deleteExp = std::make_shared<AisExperiment>(*experiment);
        auto error = handler.uploadExperimentToChannel(CHANNEL, deleteExp);
        if (error) {
            gDebug() « "Error: " « error.message();
    auto error = handler.startUploadedExperiment(0);
    if (error) {
        qDebug() « "Error: " « error.message();
});
}):
auto error = tracker->connectToDeviceOnComPort(COMPORT);
    qDebug() « "Error: " « error.message();
a.exec();
```

}

17.10 advancedExperiment.py

```
00001 """! @example advancedExperiment.py """
00002 #! [Setup]
00003 import sys
00004 from PySide6.QtWidgets import QApplication
00005 from SquidstatPyLibrary import AisDeviceTracker, AisExperiment, AisInstrumentHandler, AisErrorCode,
          AisOpenCircuitElement, AisConstantPotElement, AisConstantCurrentElement, AisEISGalvanostaticElement
00006
00007 #Define relayant device information, for easy access
00008 COMPORT = "COM1"
00009 CHANNEL = 0
00010 INSTRUMENT_NAME = "Plus2000"
00011
00012 app = QApplication()
00013
00014 tracker = AisDeviceTracker.Instance()
00015
00016 success = True
00017
00018 customExperiment = AisExperiment()
00019 # Step 1
00020
00021 ocpElement = AisOpenCircuitElement(1, 10)
00022 success &= customExperiment.appendElement(ocpElement)
00023
00026 \text{ voltage} = 0
00027 for i in range(0, 4):
                 cvElement = AisConstantPotElement(voltage, 0.1, 5)
00028
                 vuccess &= customExperiment.appendElement(cvElement, 1)
voltage = voltage + 0.1
00029
00031
00034 eisSubExperiment = AisExperiment()
00035
00036 galvEISElement = AisEISGalvanostaticElement(10, 10000, 10, 0.01, 0.1)
00037 ocpElement2 = AisOpenCircuitElement(1, 5)
00038
00039 success &= eisSubExperiment.appendElement(galvEISElement, 1)
00040 success &= eisSubExperiment.appendElement(ocpElement2, 1)
00041
00042 success &= customExperiment.appendSubExperiment(eisSubExperiment, 3)
00043
00046 ccElement = AisConstantCurrentElement(0.1, 1, 10)
00047 success &= customExperiment.appendElement(ccElement, 2)
00048
00049 if not success:
00050
                 print ("Error building experiment")
00051
                 sys.exit()
00052
00054 # When device is connected, setup connections, and upload/start the experiment
00055 def connectSignals(handler):
00056
                 handler.activeDCDataReady.connect(lambda channel, data: print(f"Timestamp: {data.timestamp}
          {\tt Current: \{data.current\}\ Voltage: \{data.working{\tt ElectrodeVoltage}\}\ CE\ Voltage
           {data.counterElectrodeVoltage}"))
00057
                 handler.activeACDataReady.connect(lambda channel, data: print(f"Timestamp: {data.timestamp}
          Frequency: {data.frequency} Absolute Impedance: {data.absoluteImpedance}"))
00058
                  handler.experimentNewElementStarting.connect(lambda channel, info: print(f"New element starting:
           {info.stepName}"))
00059
                 handler.experimentStopped.connect(lambda channel, reason: (print(f"Experiment Stopped Signal
          {channel}, {reason}"), app.quit()))
00060
                 handler.deviceError.connect(lambda channel, error: print(f"Device Error: {error}"))
00061
00062
00063 def startExperiment():
00064
                 handler = tracker.getInstrumentHandler(INSTRUMENT_NAME)
00065
00066
                  connectSignals(handler)
00067
00068
                  error = handler.uploadExperimentToChannel(CHANNEL, customExperiment)
00069
                  if error.value() != AisErrorCode.Success:
00070
                         print(error.message())
00071
                         app.quit()
00072
                  error = handler.startUploadedExperiment(CHANNEL)
00074
                  if error.value() != AisErrorCode.Success:
00075
                         print(error.message())
00076
                         app.quit()
00077
00078 tracker.newDeviceConnected.connect(startExperiment)
```

17.11 async.py

```
00001 """! @example async.py """
00002 import sys
00003 import struct
00004 import asyncio
00005 from PySide6.QtWidgets import QApplication
00006 from SquidstatPyLibrary import AisDeviceTracker
00007 from SquidstatPyLibrary import AisCompRange
00008 from SquidstatPyLibrary import AisDCData
00009 from SquidstatPyLibrary import AisACData
00010 from SquidstatPyLibrary import AisExperimentNode
00011 from SquidstatPyLibrary import AisErrorCode
00012 from SquidstatPyLibrary import AisExperiment
00013 from SquidstatPyLibrary import AisInstrumentHandler
{\tt 00014} \ {\tt from} \ {\tt SquidstatPyLibrary} \ {\tt import} \ {\tt AisConstantPotElement}
00015 from SquidstatPyLibrary import AisEISPotentiostaticElement 00016 from SquidstatPyLibrary import AisConstantCurrentElement
00018 # initialize the application
00019 app = QApplication([])
00020
00021 # Add delay before quitting
00022 async def delayed_quit():
          await asyncio.sleep(5)
00024
           app.quit()
00025
00026 \# function to print when experiment has completed
00027 def experiment_complete_handler(channel, reason):
00028     print(f"Experiment Completed on channel {channel} , {reason}")
00029
           asyncio.run(delayed_quit())
00030
00031
00032 # main function setup as async
00033 async def main():
00034
           # initialize a device tracker
           tracker = AisDeviceTracker.Instance()
00036
            # connect to device associated with the tracker
00037
            # print device serial number
00038
           tracker.newDeviceConnected.connect(lambda deviceName: print(f"Connected Device: {deviceName}"))
00039
           \ensuremath{\text{\#}} connect to device on com port 4
00040
           error = tracker.connectToDeviceOnComPort("COM4")
00041
           if error.value() != AisErrorCode.Success:
00042
               print(error.message())
00043
                app.quit()
00044
00045
            # Add initial delay before asking for hanlder (5 s)
00046
           await asyncio.sleep(5)
00047
00048
            # use serial number to get handler for instrument
           handler = tracker.getInstrumentHandler("Cycler1518")
00049
00050
            # manages DC data input and output
00051
            \ensuremath{\sharp} add more variables if you want to print more data to the console
00052
           handler.activeDCDataReady.connect(lambda channel, data: print("timestamp:",
        \{:.9f\}".format(data.timestamp), "workingElectrodeVoltage: '
       "{:.9f}".format(data.workingElectrodeVoltage)))
00053
            # manages AC data input and output
00054
            # add more variables if you want to print more data to the console
       handler.activeACDataReady.connect(lambda channel, data: print("frequency:",
"{:.9f}".format(data.frequency), "absoluteImpedance: ", "{:.9f}".format(data.absoluteImpedance),
"phaseAngle: ", "{:.9f}".format(data.phaseAngle)))
00055
00056
           # print when a new node starts to the console
           handler.experimentNewElementStarting.connect(lambda channel, data: print("New Node beginning:",
           i.stepName, "step number: ", data.stepNumber, " step sub : ", data.substepNumber)) # called when an experiment has completed
00058
00059
           \verb|handler.experimentStopped.connect(experiment\_complete\_handler)|\\
00060
00061
            # initialize an experiment
00062
           experiment = AisExperiment()
00063
            ^{\#} define a constant potential experiment at 0.2 V, with 1 s sampling time, and a duration of 10 s
           cvElement = AisConstantPotElement(0.2, 1, 10)
# define a constant current experiment at 0.1 A, with 0.1 s sampling time, and a duration of 5 s
00064
00065
00066
           ccElement = AisConstantCurrentElement (0.1, 0.1, 5)
00067
00068
           success = True
```

```
00069
          # initialize a sub experiment
00070
00071
          subExperiment = AisExperiment()
00072
          \# add constant current experiment to position 1 of the sub experiment
00073
          # this experiment will run 1 time
00074
          success &= subExperiment.appendElement(ccElement, 1)
          # add constant potential experiment to positions 2 and 3 of the sub experiment
00076
          # this experiment will run 2 times
00077
          success &= subExperiment.appendElement(cvElement, 2)
00078
00079
          \# add constant current experiment to position 1 and 2 of the main experiment
00080
          # this experiment will run 2 times
00081
          success &= experiment.appendElement(ccElement, 2)
00082
          # add constant potential experiment to position 3 of the main experiment
00083
          # this experiment will run 1 time
00084
          success &= experiment.appendElement(cvElement, 1)
00085
00086
          # add the sub experiment to the main experiment
          # the sub experiment will run 2 times
00087
00088
          success &= experiment.appendSubExperiment(subExperiment, 2)
00089
00090
          if not success:
             print("Error building experiment")
00091
00092
              app.quit()
00093
00094
          # upload experiment list to the given channel
00095
          error = handler.uploadExperimentToChannel(0, experiment)
00096
          if error.value() != AisErrorCode.Success:
00097
              print(error.message())
00098
              app.quit()
00099
00100
          # start the expiment on channel
00101
          error = handler.startUploadedExperiment(0)
00102
          if error.value() != AisErrorCode.Success:
00103
              print(error.message())
00104
              app.quit()
00105
00106
          # Add initial delay (5 s)
00107
         await asyncio.sleep(5)
00108
00109 # setup main to be ran as async
00110 startFunc = main()
00111 asyncio.run(startFunc)
00112
00113 # exit application
00114 sys.exit(app.exec()) # Start the event loop
00115
```

17.12 basicExperiment.py

```
00001 """! @example basicExperiment.py """
00003 import sys
00004 from PySide6.QtWidgets import QApplication
00005 from SquidstatPyLibrary import AisDeviceTracker
00006 from SquidstatPyLibrary import AisExperiment, AisErrorCode 00007 from SquidstatPyLibrary import AisInstrumentHandler
00008 from SquidstatPyLibrary import AisConstantPotElement
00009
00010 \# Define relavant device information, for easy access 00011 COMPORT = "COM16"
00012 CHANNEL = 0
00013
00014 app = QApplication()
00015
00016 tracker = AisDeviceTracker.Instance()
00017
00018
00019
00021 cvElement = AisConstantPotElement(1, 1, 30)
00023 # After this point, the experiment is empty, so we need to add some elements to it
00024 experiment = AisExperiment()
00025 # Append the constant potential element, and tell the experiment to execute that element 1 time
00026 success = experiment.appendElement(cvElement, 1)
00027
00028 # Check if the element was added successfully
00029 if not success:
00030
          print("Error adding element to experiment")
00031
           app.quit()
00032
00033
```

```
00035 def connectSignals(handler):
00037
                  handler.activeDCDataReady.connect(lambda channel, data: print(f"Timestamp: {data.timestamp}
           {\tt Current: \{data.current\}\ Voltage: \{data.working{\tt ElectrodeVoltage}\}\ CE\ Voltage
           {data.counterElectrodeVoltage}"))
00038
                  handler.activeACDataReady.connect(lambda channel, data: print(f"Timestamp: {data.timestamp}
           Frequency: {data.frequency} Absolute Impedance: {data.absoluteImpedance}"))
00039
00041
00042
                   # Whenever a new node in the element starts, note: some Ais Elements contain multiple logical
           nodes
00043
                  # i.e AisCyclicVoltammatryElement contains 4 nodes for each linear segment of each cycle plus a
           quiet time node if enabled
                   # So this lambda would be executed atleast 4 times for each cycle
00044
00045
                   handler.experimentNewElementStarting.connect(lambda channel, info: print(f"New element starting:
           {info.stepName}"))
00046
                   # Whenever an experiment completes or is manually stopped, this will execute
                  handler.experimentStopped.connect(lambda channel, reason: (print(f"Experiment Stopped Signal
00047
           {channel}, {reason}"), app.quit()))
00048
                  handler.deviceError.connect(lambda channel, error: print(f"Device Error: {error}"))
00049
00051
00052 def startExperiment (deviceName):
00053
                  print(f"New Device Connected: {deviceName}")
00054
00055
                   handler = tracker.getInstrumentHandler(deviceName)
00056
00057
                   connectSignals(handler)
00058
00059
                   error = handler.uploadExperimentToChannel(CHANNEL, experiment)
                   # Exit the application if there is any error uploading experiment
00060
00061
                   if error.value() != AisErrorCode.Success:
00062
                         print(error.message())
00063
                           app.quit()
00064
                   error = handler.startUploadedExperiment(CHANNEL)
00065
00066
                   # Exit the application if there is any error starting experiment
                   if error.value() != AisErrorCode.Success:
00068
                          print(error.message())
00069
                           app.quit()
00070
00071
00072
00073 tracker.newDeviceConnected.connect(startExperiment)
00074 tracker.deviceDisconnected.connect(lambda deviceName: print(f"Device Disconnected: {deviceName}"))
00075
00076
00077
00078 error = tracker.connectToDeviceOnComPort(COMPORT)
00079 # Check if connection was successful
00080 if error.value() != AisErrorCode.Success:
                  print(error.message())
00081
00082
                   sys.exit()
00083
00084
00085 # Calling sys.exit(app.exec()) will keep the program running until the application is exited
00086 sys.exit(app.exec())
```

17.13 dataOutput.py

```
00001 """! @example dataOutput.py """
00002 import sys
00003 from PySide6.OtWidgets import OApplication
00004 from PySide6.QtCore import QTextStream, QFile, QStandardPaths, QIODevice
00005 from SquidstatPyLibrary import AisDeviceTracker, AisErrorCode
00006 from SquidstatPyLibrary import AisExperiment
00007 from SquidstatPyLibrary import AisInstrumentHandler
00008 from SquidstatPyLibrary import AisConstantPotElement 00009 from SquidstatPyLibrary import AisConstantCurrentElement
00010
00011 # Define relavant device information, for easy access
00012 COMPORT = "COM1"
00013 CHANNEL = 0
00014
00015 app = QApplication()
00016
00017 tracker = AisDeviceTracker.Instance()
00018
00019 cvElement = AisConstantPotElement(1, 1, 30)
00020 ccElement = AisConstantCurrentElement(0.001, 1, 60)
00021
00022 experiment = AisExperiment()
```

```
00024 success = True
00025
00026 success &= experiment.appendElement(cvElement, 1)
00027 success &= experiment.appendElement(ccElement, 1)
00028
00029 if not success:
         print("Error building experiment")
00031
          sys.exit()
00032
00033 filePath = ""
00034 \text{ fileNum} = 1
00035
00036 def onNewElementStarting(channel, info):
00037
          name = f"/{fileNum}_{info.stepNumber}_{info.stepName}.csv"
00038
          filePath = QStandardPaths.writableLocation(QStandardPaths.DesktopLocation) + name
00039
00040
          file = OFile(filePath)
00041
          if not file.open(QIODevice.WriteOnly | QIODevice.Text):
00042
              return
00043
00044
          # Writing headers to file
          out = QTextStream(file)
out « "Time Stamp, Counter Electrode Voltage, Working Electrode Voltage, Current \n"
00045
00046
00047
          file.close()
00048
00049
          print(f"New element beginning: {info.stepName} step: {info.stepNumber}")
00050
00051 def onActiveDCDataReady(channel, data):
00052
          print(f"current: {data.current} voltage: {data.workingElectrodeVoltage} counter electrode:
      {data.counterElectrodeVoltage} timestamp: {data.timestamp}")
00053
00054
           Save the DC data to the file created at element beginning
00055
          file = QFile(filePath)
00056
          if not file.open(QIODevice.Append | QIODevice.WriteOnly | QIODevice.Text):
00057
00058
00059
          out = OTextStream(file)
         out « data.timestamp « "," « data.counterElectrodeVoltage « "," « data.workingElectrodeVoltage «
00060
     "," « data.current « "\n"
00061
         file.close()
00062
00063 def connectSignals(handler):
          handler.activeDCDataReady.connect(onActiveDCDataReady)
00064
          handler.activeACDataReady.connect(lambda channel, data: print(f"Timestamp: {data.timestamp}
00065
      Frequency: {data.frequency} Absolute Impedance: {data.absoluteImpedance}"))
00066
          handler.experimentNewElementStarting.connect(onNewElementStarting)
00067
         handler.experimentStopped.connect(lambda channel, reason: (print(f"Experiment Stopped Signal
      {channel}, {reason}"), app.quit()))
00068
          handler.deviceError.connect(lambda channel, error: print(f"Device Error: {error}"))
00069
00070 def runExperiment(deviceName):
00071
          handler = tracker.getInstrumentHandler(deviceName)
00072
00073
          connectSignals(handler)
00074
00075
          error = handler.uploadExperimentToChannel(CHANNEL, experiment)
00076
          # Exit the application if there is any error uploading experiment
00077
          if error.value() != AisErrorCode.Success:
00078
             print(error.message())
00079
              app.quit()
08000
00081
          error = handler.startUploadedExperiment(CHANNEL)
00082
          # Exit the application if there is any error starting experiment
00083
          if error.value() != AisErrorCode.Success:
00084
              print(error.message())
00085
              app.quit()
00086
00087 tracker.newDeviceConnected.connect(runExperiment)
00088 tracker.deviceDisconnected.connect(lambda deviceName: print(f"Device Disconnected: {deviceName}"))
00090 error = tracker.connectToDeviceOnComPort(COMPORT)
00091 # Check if connection was successful
00092 if error.value() != AisErrorCode.Success:
00093
         print(error.message())
00094
          svs.exit()
00095
00096 # Calling sys.exit(app.exec()) will keep the program running until the application is exited
00097 sys.exit(app.exec())
```

17.14 firmwareUpdate.py

```
00001 """! @example firmwareUpdate.py """
```

```
00003 import sys
00004 from PySide6.QtWidgets import QApplication
00005
{\tt 00006 from \ SquidstatPyLibrary \ import \ AisDeviceTracker}
00007 from SquidstatPyLibrary import AisInstrumentHandler
00008 from SquidstatPyLibrary import AisErrorCode
00010 # Define relavant device information, for easy access
00011 COMPORT = "COM16"
00012
00013 app = QApplication()
00014
00015 tracker = AisDeviceTracker.Instance()
00016
00017 def onProgressMessage(message):
00018
         print (message)
          if message._
00019
                       _contains__("firmware is updated"):
00020
              app.quit()
00021
00022
00023 tracker.firmwareUpdateNotification.connect(onProgressMessage)
00024
00025
00026 \# Attempt to connect to the device
00027 error = tracker.connectToDeviceOnComPort(COMPORT)
00028 if error.value() == AisErrorCode.FirmwareNotSupported:
         error = tracker.updateFirmwareOnComPort(COMPORT)
00029
00030
00031
         # Some other error occured
00032
         if error.value() != AisErrorCode.Success:
             print(f"Error: {error.message()}")
00033
00034
              sys.exit()
00035 elif error.value() != AisErrorCode.Success:
       print(f"Error: {error.message()}")
00036
00037
         sys.exit()
00038 else:
00039
         print ("Device is already up to date.")
          sys.exit()
00041
00042 sys.exit(app.exec())
00043
```

17.15 linkedChannels.py

This example will show you how linked channels can be used to combine the multiple channels on a device, in order to amplify the output for a single experiment.

This example will show you how linked channels can be used to combine the multiple channels on a device, in order to amplify the output for a single experiment. AisInstrumentHandler.setLinkedChannels MUST be called before each experiment that uses paralleled channels. Once linked, these channels must be controlled by ONLY the master channel, which is returned by the AlsInstrumentHandler::setLinkedChannels function.

Note

This feature is only available on Cycler models.

```
00001 """! @example linkedChannels.py
          This example will show you how linked channels can be used to combine the multiple channels on a
00002
     device, in order to amplify the output for a single experiment.
00003
         AisInstrumentHandler::setLinkedChannels MUST be called before each experiment that uses paralleled
     channels.
00004
         Once linked, these channels must be controlled by ONLY the master channel, which is returned by
     the AIsInstrumentHandler::setLinkedChannels function.
00005
00006
         Onote This feature is only available on Cycler models.
00007 """
00008 import sys
00009 from PySide6.QtWidgets import QApplication
00010 from SquidstatPyLibrary import AisDeviceTracker, AisInstrumentHandler, AisErrorCode,
     AisConstantCurrentElement, AisExperiment
00011
00012 # Define relavant device information, for easy access
00013 COMPORT = "COM1"
00014
00015 app = QApplication()
00016
```

```
00017 tracker = AisDeviceTracker.Instance()
00019 # Build an experiment
00020 ccElement = AisConstantCurrentElement(10, 1, 30)
00021 experiment = AisExperiment()
00022 success = experiment.appendElement(ccElement)
00024 if not success:
00025
                   print("Error adding element to experiment")
00026
                    app.quit()
00027
00028 def connectSignals(handler):
                   handler.activeDCDataReady.connect(lambda channel, data: print(f"Timestamp: {data.timestamp}
00029
           Current: {data.current} Voltage: {data.workingElectrodeVoltage} CE Voltage:
            {data.counterElectrodeVoltage}"))
00030
                    handler.deviceError.connect(lambda channel, error: print(f"Device Error: {error}"))
                   \verb|handler.experimentStopped.connect(lambda channel, reason: (print(f"Experiment Stopped Signal Institute of the context of t
00031
           {channel}, {reason}"), app.quit()))
00032
00033 def startExperiment(deviceName):
00034
                    handler = tracker.getInstrumentHandler(deviceName)
00035
00036
                    # Here we want to link channels 0 and 1 together, so we pass in a vector of the channels to link
00037
                   # It will return which of the channels is the master channel, this should be used to control the
           experiment
00038
                   masterChannel = handler.setLinkedChannels([ 0, 1 ])
00039
00040
                    connectSignals(handler)
00041
00042
                    error = handler.uploadExperimentToChannel(masterChannel, experiment)
00043
                    if error.value() != AisErrorCode.Success:
00044
                          print(error.message())
00045
                           app.quit()
00046
00047
                    # Start the previously uploaded experiment on the master channel
00048
                    error = handler.startUploadedExperiment(masterChannel)
00049
                    if error.value() != AisErrorCode.Success:
                           print (error.message())
00051
                            app.quit()
00052
00053
00054 tracker.newDeviceConnected.connect(startExperiment)
00055 tracker.deviceDisconnected.connect(lambda deviceName: print(f"Device Disconnected: {deviceName}"))
00057 error = tracker.connectToDeviceOnComPort(COMPORT)
00058 if error.value() != AisErrorCode.Success:
00059
                  print(error.message())
00060
                    sys.exit()
00061 # Calling sys.exit(app.exec()) will keep the program running until the application is exited
00062 svs.exit(app.exec())
```

17.16 manualExperiment.py

```
00001 """! @example manualExperiment.py """
00002 import sys
00003 from PySide6.QtWidgets import QApplication
00004 from PySide6.QtCore import QTimer
00005 from SquidstatPyLibrary import AisDeviceTracker
00006 from SquidstatPyLibrary import AisInstrumentHandler
00007 from SquidstatPyLibrary import AisErrorCode
80000
00009
00010 # Define relavant device information, for easy access
00011 COMPORT = "COM1"
00012 CHANNEL = 0
00013
00014 app = QApplication()
00015
00016 tracker = AisDeviceTracker.Instance()
00017
00018 def handleStopExperiment(handler, reason):
00019
00020
          print("Experiment has ended. Closing application.")
00021
          app.quit()
00022
00023
00024 def connectSignal(handler):
          handler.activeDCDataReady.connect(lambda channel, data: print(f"Timestamp: {data.timestamp}
      Current: {data.current} Voltage: {data.workingElectrodeVoltage} CE Voltage:
      {data.counterElectrodeVoltage}"))
00026
          handler.deviceError.connect(lambda channel, error: print(f"Device Error: {error}"))
00027
          handler.experimentStopped.connect(handleStopExperiment)
```

```
00029 def startExperiment(deviceName):
          handler = tracker.getInstrumentHandler(deviceName)
00031
00032
          connectSignal(handler)
00033
00034
00035
00036
          # The default starting mode is always Open Circut Potential.
00037
00038
          print("Starting manual mode at open circuit potential")
00039
          error = handler.startManualExperiment (CHANNEL)
00040
          if error.value() != AisErrorCode.Success:
00041
              print(error.message())
00042
              app.quit()
00043
00044
00045
00046
          # In this section we create wrapper functions for the manual mode changing functions.
00048
          # These wrappers are called asynchronously when singleshot QTimers expire.
00049
00050
          \ensuremath{\text{\#}} This function changes the instrument to Constant Current at .1A
00051
          {\tt def setConstantCurrent(channel):}
00052
              print("Switching to constant current at .1A")
00053
              error = handler.setManualModeConstantCurrent(channel, .1)
              if error.value() != AisErrorCode.Success:
00054
00055
                  print(error.message())
00056
          # It is called 5 seconds after the experiment starts
00057
          QTimer.singleShot(5000, lambda:setConstantCurrent(CHANNEL))
00058
00059
          # This function changes the instrument to Constant Voltage at 1V
00060
          def setConstantVoltage(channel):
00061
              print("Switching to constant voltage at 1V")
00062
              error = handler.setManualModeConstantVoltage(channel, 1)
00063
              if error.value() != AisErrorCode.Success:
00064
                  print(error.message())
00065
          # It is called 15 seconds after the experiment starts
          QTimer.singleShot(15000, lambda:setConstantVoltage(CHANNEL))
00066
00067
00068
          # This function changes the instrument to Open Circuit Potential
00069
          def setOpenCircuit(channel):
00070
              print("Switching to open circuit potential")
00071
              error = handler.setManualModeOCP(channel)
00072
              if error.value() != AisErrorCode.Success:
00073
                 print(error.message())
00074
          # It is called 25 seconds after the experiment starts
00075
          QTimer.singleShot(25000, lambda:setOpenCircuit(CHANNEL))
00076
00077
00078
00079
00080
00081
          # Stop experiment after 30 seconds
00082
          def stopExperiment(channel):
00083
              print("Stopping experiment.")
              error = handler.stopExperiment(channel)
00084
              if error.value() != AisErrorCode.Success:
00086
                  print(error.message())
00087
          QTimer.singleShot(30000, lambda:stopExperiment(CHANNEL))
00088
00089
00090
00091 tracker.newDeviceConnected.connect(startExperiment)
00092 tracker.deviceDisconnected.connect(lambda deviceName: print(f"Device Disconnected: {deviceName}"))
00093
00094 error = tracker.connectToDeviceOnComPort(COMPORT)
00095 if error.value() != AisErrorCode.Success:
00096
         print (error.message())
          svs.exit()
00098 # Calling sys.exit(app.exec()) will keep the program running until the application is exited
00099 sys.exit(app.exec())
```

17.17 nonblockingExperiment.py

```
00001 """! @example nonblockingExperiment.py """
00002 import sys
00003 from PySide6.QtWidgets import QApplication
00004 from PySide6.QtCore import QTimer
00005 from SquidstatPyLibrary import AisDeviceTracker
00006 from SquidstatPyLibrary import AisInstrumentHandler
00007 from SquidstatPyLibrary import AisErrorCode
00008
00009
```

17.18 pulseData.py 231

```
00010 # Define relavant device information, for easy access
00012 \text{ CHANNEL} = 0
00013
00014 app = QApplication()
00015
00016 tracker = AisDeviceTracker.Instance()
00017
00018 def connectSignals(handler):
00019
          handler.activeDCDataReady.connect(lambda channel, data: print(f"Timestamp: {data.timestamp}
      Current: {data.current} Voltage: {data.workingElectrodeVoltage} CE Voltage :
      {data.counterElectrodeVoltage}"))
00020
          handler.activeACDataReady.connect(lambda channel, data: print(f"Frequency: {data.frequency}
      Absolute Impedance: {data.absoluteImpedance} Phase Angle: {data.phaseAngle}")
00021
          handler.experimentNewElementStarting.connect(lambda channel, nodeInfo: print(f"New Node
00022
         handler.experimentStopped.connect(lambda channel, reason: print(f"Experiment Stopped Signal
      {channel}, {reason}"))
00023
00024 def startExperiment(deviceName):
00025
          handler = tracker.getInstrumentHandler(deviceName)
00026
00027
          connectSignals(handler)
00028
00029
          error = handler.startManualExperiment(CHANNEL)
00030
          if error.value() != AisErrorCode.Success:
00031
              print(error.message())
00032
              app.quit()
00033
00034
          error = handler.setManualModeSamplingInterval(CHANNEL, 2)
00035
          if error.value() != AisErrorCode.Success:
00036
              print(error.message())
00037
              app.quit()
00038
00039
          error = handler.setManualModeConstantVoltage(CHANNEL, 2)
00040
          if error.value() != AisErrorCode.Success:
00041
              print(error.message())
00042
              app.quit()
00043
00044
          error = handler.setManualModeOCP(CHANNEL)
00045
          if error.value() != AisErrorCode.Success:
00046
              print(error.message())
00047
              app.quit()
00048
00049
          def stopExperiment():
00050
              error = handler.stopExperiment(CHANNEL)
00051
              if error.value() != AisErrorCode.Success:
00052
                  print(error.message())
00053
                  app.quit()
00054
00055
          QTimer.singleShot(25000, stopExperiment)
00056
00057
          # Allow the recent loop to process other events while the current experiment is running
00058
          while handler.isChannelBusy(CHANNEL):
00059
              app.processEvents()
00060
00061
          # Process any remaining events
00062
          app.processEvents()
00063
00064 tracker.newDeviceConnected.connect(startExperiment)
00065
00066 numDevices = tracker.connectAllPluggedInDevices()
00067 if numDevices == 0:
         print("No devices connected")
00068
00069
          sys.exit()
00070
00071 \# Calling sys.exit(app.exec()) will keep the program running until the application is exited
00072 sys.exit(app.exec())
```

17.18 pulseData.py

```
00001 """! @example pulseData.py """
00002
00003 import sys
00004 from PySide6.QtWidgets import QApplication
00005 from SquidstatPyLibrary import AisDeviceTracker
00006 from SquidstatPyLibrary import AisCompRange
00007 from SquidstatPyLibrary import AisDCData
00008 from SquidstatPyLibrary import AisACData
00009 from SquidstatPyLibrary import AisExperimentNode
00010 from SquidstatPyLibrary import AisExperiment
00011 from SquidstatPyLibrary import AisExperiment
00012 from SquidstatPyLibrary import AisExperiment
```

```
00013 from SquidstatPyLibrary import AisCyclicVoltammetryElement
00014 from SquidstatPyLibrary import AisDiffPulseVoltammetryElement
00015 from SquidstatPyLibrary import AisDataManipulator
{\tt 00016} \ {\tt from} \ {\tt SquidstatPyLibrary} \ {\tt import} \ {\tt AisPulseType}
00017
00018 # do you want headers in your file?
00019 write_header = True
00020 # instantiate the data manipulator
00021 data_manipulator = AisDataManipulator()
00022 # setup COM port
00023 COMPORT = "COM5"
00024 CHANNEL = 0
00025
00026 def create_logic(handler):
00027
         def on_active_dc_data_ready(channel, data):
00028
              # define global parameters
00029
              global write_header, data_manipulator
00030
00031
              # convert time to UTC
00032
              # utc_time = handler.getExperimentUTCStartTime(0)
00033
              # read data via data manipulator
00034
              data_manipulator.loadPrimaryData(data)
00035
00036
              # upon completed pulse, write data with header
00037
              if data_manipulator.isPulseCompleted():
00038
00039
                  # if write_header = true
00040
                  # prints header for every pulse data point
00041
                  if write_header:
                      print("time, Pulse_current, base_current, diff_current, base_voltage, pulse_voltage")
00042
00043
                      write header = False
00044
00045
                  \# will print data as defined by the header print statement above
     00046
00047
00048
                      f"\{data\_manipulator.getBaseVoltage()\}, \{data\_manipulator.getPulseVoltage()\}")
00049
00050
          # function for printing ac data when it is received
00051
          def on_active_ac_data_ready(channel, data):
00052
     print(f"channel: {channel}, frequency: {data.frequency}, absoluteImpedance:
{data.absoluteImpedance}, "
00053
                  f"phaseAngle: {data.phaseAngle}, timestamp: {data.timestamp}")
00054
00055
          # function for printing a new node when it begins
00056
          def on_experiment_new_element_starting(channel, data):
00057
              print(f"New Node beginning {data.stepName}, step number {data.stepNumber}, step sub:
      {data.substepNumber}")
00058
00059
          # fucntion for printing an experiment has stopped
00060
          def on_experiment_stopped(channel, reason):
00061
              print(f"Experiment has completed on channel {channel}, {reason}")
00062
              QApplication.quit()
00063
00064
          # function to print an experiment has paused
00065
          def on_experiment_paused(channel):
              print(f"Experiment on channel {channel} has been paused")
00066
00067
00068
          # function to print an experiment has resumed
00069
          def on_experiment_resumed(channel):
00070
              \verb|print(f"Experiment resumed on channel {channel}")|\\
00071
00072
          # pass dc data to data manipulator function
00073
          handler.activeDCDataReady.connect(on_active_dc_data_ready)
00074
00075
          # pass ac data to print function
00076
          handler.activeACDataReady.connect(on_active_ac_data_ready)
00077
00078
          # pass new node starting to print function
00079
          handler.experimentNewElementStarting.connect(on_experiment_new_element_starting)
00080
00081
          # pass experiment stopped to print function
00082
          handler.experimentStopped.connect(on_experiment_stopped)
00083
00084
          # pass experiment puased to print function
          handler.experimentPaused.connect(on_experiment_paused)
00085
00086
          handler.experimentResumed.connect(on_experiment_resumed)
00087
00088 # setup experiment
00089 def main():
00090
          # initialize the application
00091
          app = QApplication()
00092
00093
          # get a device tracker
00094
          tracker = AisDeviceTracker.Instance()
00095
00096
          # Create the AisDiffPulseVoltammetryElement pulse experiment.
```

17.19 tcpClient.py 233

```
00097
          # startPotential (V), endPotential (V), potentialStep (V), pulseHeight (V), pulseWidth (s),
      pulsePeriod (s)
          dpv_element = AisDiffPulseVoltammetryElement(-0.115, 0.115, 0.005, 0.01, 0.02, 0.2)
00098
00099
00100
          # set start voltage VS reference
00101
          dpv element.setStartVoltageVsOCP(False)
00102
00103
          # set end voltage VS reference
00104
          dpv_element.setEndVoltageVsOCP(False)
00105
00106
          # set current range
          # will range up, but will not range down
00107
00108
          dpv_element.setApproxMaxCurrent(0.2)
00109
00110
          # initialize an experiment
00111
          experiment = AisExperiment()
00112
00113
          # append differential pulse element to the experiment list
00114
          # will run 1 time
00115
          success = experiment.appendElement(dpv_element, 1)
00116
          if not success:
00117
              print("Error building experiment")
00118
              sys.exit()
00119
          # Create an `AisDataManipulator' class for calculating advance parameters.
00120
00121
          data_manipulator.setPulseType(AisPulseType.DifferentialPulse, dpv_element.getPulseWidth(),
      dpv_element.getPulsePeriod())
00122
00123
          # if device is connected, get name and use logic function to handle events
00124
          def on_new_device_connected(device_name):
00125
00126
              # get instrument handler using device name
00127
              handler = tracker.getInstrumentHandler(device_name)
00128
00129
              # create the required connections for the handler.
00130
              create_logic(handler)
00131
00132
              # uplaod experiment to device.
00133
              error = handler.uploadExperimentToChannel(CHANNEL, experiment)
00134
              if error.value() != AisErrorCode.ErrorCode.Success:
00135
                  print(f"Error: {error.message()}")
00136
                  app.quit()
00137
00138
              # start experiment on device on defined channel
              error = handler.startUploadedExperiment(CHANNEL)
00139
00140
              if error.value() != AisErrorCode.ErrorCode.Success:
00141
                  print(f"Error: {error.message()}")
00142
                  app.quit()
00143
00144
          # connect call back handler which is called on connection of device.
00145
          tracker.newDeviceConnected.connect(on_new_device_connected)
00146
00147
          # print which device has disconnected to terminal
00148
          tracker.deviceDisconnected.connect(lambda device_name: print(f"{device_name}) has been
     disconnected"))
00149
00150
          # if error is encountered, print to terminal
00151
          error = tracker.connectToDeviceOnComPort(COMPORT)
00152
          if error.value() != AisErrorCode.ErrorCode.Success:
00153
              print(f"Error: {error.message()}")
00154
              sys.exit()
00155
00156
          # exit application
          sys.exit(app.exec())
00157
00158
00159 # run main
                _ == "__main__":
00160 if ___name_
00161
         main()
```

17.19 tcpClient.py

This example file shows how to create a TCP client which can send commands to a server that is connected to a Squidstat.

This example file shows how to create a TCP client which can send commands to a server that is connected to a Squidstat. You can build experiments and recieve data from the server without having to interact with the instrument directly.

00001 """! @example tcpClient.py This example file shows how to create a TCP client which can send commands to a server that is connected to a Squidstat.

234 Examples

```
00002
00003 You can build experiments and recieve data from the server without having to interact with the
      instrument directly.
00004 """
00005
00006 import os
00007 import socket
00008 import threading
00009 import time
00010
00011 # Create a TCP/IP socket
00012 client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
00013 activeSockets = [client_socket]
00014
00015 \# Define the server address and port
00016 SERVER_HOST = "localhost"
00017 SERVER_PORT = 12345
00018
00019 # Function to send a command to the server
00020 def send_command(command):
00021
           # Send the command to the server
00022
          try:
              client_socket.send(command.encode())
00023
00024
          except:
00025
              print ("Connection was closed by host")
00026
              os._exit(1)
00027
00028
          \ensuremath{\mathtt{\#}} Receive and print the response from the server
00029
          response = client_socket.recv(1024).decode()
          print ("Server response:", response)
00030
00031
00032 # listens for <CTRL>+c to stop the client script
00033 def interupt_listener():
00034
          print("Press <CTRL>+c to stop the program at any time.")
00035
              while True:
00036
00037
                   input()
00038
          except (EOFError, KeyboardInterrupt):
00039
00040
          for socket in activeSockets:
00041
              socket.close()
00042
          os._exit(1)
00043
00044 # Try and open a socket to the server
00045 try:
00046
          client_socket.connect((SERVER_HOST, SERVER_PORT))
00047 except Exception as ex:
          print("Unable to establish connection to server:\n%s" % ex)
00048
00049
          exit()
00050
00051 print("Connected to the server.")
00052
00053 \# Get a duration from the user
00054 \text{ duration} = 0
00055 while duration == 0:
00056
          try:
00057
              duration = int(input("Enter a duration for Open Circuit Potential: "))
00058
          except ValueError:
00059
               duration = 0
00060
          if (duration < 1):
              print("Invalid entry.")
00061
00062
              duration = 0
00063
{\tt 00064}\ {\tt \#}\ {\tt Send}\ {\tt the}\ {\tt start}\ {\tt command}\ {\tt to}\ {\tt the}\ {\tt server}\ {\tt with}\ {\tt the}\ {\tt duration}
00065 send_command(f'startExperiment {duration}')
00066
00067 interupt_thread = threading.Thread(target=interupt_listener)
00068 interupt_thread.start()
00069
00070 # Listen for information from the server, which at this point will be data and the experiment stop
     message
00071 while True:
00072
00073
              data = client socket.recv(1024).decode()
00074
          except (ConnectionAbortedError, BrokenPipeError):
00075
               # This exception will be raised when the user presses <ENTER>
00076
               print("Finishing connection")
00077
00078
          except ConnectionResetError:
00079
               print("The server closed the connection suddenly.")
00080
00081
00082
          if not data:
00083
00084
          # Handle the data that was received.
00085
00086
          print (data)
```

17.20 tcpServer.py 235

```
00087
00088 if("Experiment Completed: " in data):
00089 break
00090
00091 os._exit(1)
```

17.20 tcpServer.py

This example file shows how to create a server that connects to an instrument.

This example file shows how to create a server that connects to an instrument. It can recieve TCP commands from a client to trigger experiments and send data back to the client.

```
00001 """! @example tcpServer.py
00002 @brief This example file shows how to create a server that connects to an instrument.
00003
00004 It can recieve TCP commands from a client to trigger experiments and send data back to the client.
00006 import os
00007 import socket
00008 import threading
00009 from PySide6.QtWidgets import QApplication
00010 from SquidstatPyLibrary import AisDeviceTracker 00011 from SquidstatPyLibrary import AisExperiment
00012 from SquidstatPyLibrary import AisOpenCircuitElement
00013 from SquidstatPyLibrary import AisErrorCode
00014
00015 # Define the server address and port
00016 HOST = 'localhost'
00017 PORT = 12345
00019 # The COM port the Squidstat is connected to 00020 SQUIDCOMPORT = "COM1"
00021 SQUIDNAME = "Plus2000"
00022
00023 # Create the QT application
00024 app = QApplication([])
00025 activeSockets = []
00026
00027 \# This will build and start the Open Circuit Potential experiment
00028 def start_ocp_experiment(handler, durationSec=60):
00029  # Create an experiment with elements
           experiment = AisExperiment()
00031
          ocpElement = AisOpenCircuitElement(durationSec, 1)
00032
00033
           success = experiment.appendElement(ocpElement, 1)
00034
          if not success:
00035
              print ("Error adding element to experiment")
00036
               return error. Experiment Not Uploaded
00037
           \mbox{\tt\#} Upload the experiment to channel \mbox{\tt 0}
00038
00039
           error = handler.uploadExperimentToChannel(0, experiment)
00040
          if error.value() != AisErrorCode.ErrorCode.Success:
00041
               return error
00042
00043
           # Start the experiment
00044
           return(handler.startUploadedExperiment(0))
00045
00046 # Send a specified command to our Squidstat
00047 def command_to_device(command, handler):
00048  #Check if we had an argument associated with the command
00049
           splitCommand = command.split(" ")
00050
           action = splitCommand[0]
00051
           actionArg = 0
           if(len(splitCommand) > 1):
00052
00053
               try:
00054
                   actionArg = int(splitCommand[1])
00055
               except:
00056
                   actionArg = 0
00057
00058
           # Here you can add various commands which can be send from the tcpClient to directly interact with
     the Squidstat
00059
          response = None
00060
           if action == 'startExperiment':
00061
               response = start_ocp_experiment(handler, actionArg)
           elif action == 'stopExperiment':
00062
00063
               response = handler.stopExperiment(0)
00064
           else:
00065
               #print("Invalid command:", command)
00066
          return response
```

236 Examples

```
00069 # Handle commands from the client
00070 def handle_command(command, handler, client_socket):
          # Send a response back to the client
responseMsg = "Unknown Command"
00071
00072
00073
          response = command_to_device(command, handler)
00074
          if(response != None):
00075
              responseMsg = response.message()
00076
          response = "{}".format(responseMsg)
00077
          client_socket.send(response.encode())
00078
00079 # Listen for the client's messages, and disconnect signals and terminate program when finished
00080 def handle_client(handler, client_socket):
          print ("Client connected")
00081
00082
00083
          while True:
00084
              # Receive data from the client
00085
              try:
00086
                 data = client_socket.recv(1024).decode()
00087
              except ConnectionResetError:
00088
00089
00090
              # Check if the client has closed the connection
00091
              if not data:
00092
00093
00094
              # Handle the command
00095
              handle_command(data, handler, client_socket)
00096
00097
00098
          handler.activeDCDataReadv.disconnect()
00099
          handler.activeACDataReady.disconnect()
00100
          handler.experimentNewElementStarting.disconnect()
00101
          handler.experimentStopped.disconnect()
00102
          command_to_device("stopExperiment", handler)
00103
          # Close the client socket
00104
          client socket.close()
          print("Client disconnected")
00105
00106
          os._exit(1)
00107
00108 # Send data the the client based on the type of event (Hooked up to signals)
00109 def send_data_to_client(client_socket, event_type, data):
         if event_type == "DCData":
    message = "timestamp: {:.9f}, workingElectrodeVoltage: {:.9f}".format(data.timestamp,
00110
00111
     data.workingElectrodeVoltage)
        elif event_type == "ACData":
00112
00113
             message = "frequency: {:.9f}, absoluteImpedance: {:.9f}, phaseAngle:
     {:.9f}".format(data.frequency, data.absoluteImpedance, data.phaseAngle)
00114
         elif event_type == "NewElement":
    message = "New Node beginning: {}, step number: {}, step sub: {}".format(data.stepName,
00115
     data.stepNumber, data.substepNumber)
00116
         elif event_type == "ExperimentCompleted":
00117
             message = "Experiment Completed: {}".format(data)
00118
          else:
00119
              return
00120
00121
          client_socket.send(message.encode())
00122
00123 def terminate_program():
00124
          print("Press <CTRL>+c to close the server")
00125
          try:
00126
              while True:
00127
                  input()
00128
          except (EOFError, KeyboardInterrupt):
00129
00130
          for socket in activeSockets:
00131
             socket.close()
00132
          app.quit()
00133
          os. exit(1)
00134
00135
00136 \# Create the device tracker and connect to the Squidstat we will be using
00137 print(f"Attempting to connect to the Squidstat {SQUIDNAME} on {SQUIDCOMPORT}...")
00138 tracker = AisDeviceTracker.Instance()
00139 tracker.newDeviceConnected.connect(lambda deviceName: print("Device is Connected: %s" % deviceName))
00140 error = tracker.connectToDeviceOnComPort(SQUIDCOMPORT)
00141
00142 if error.value() != AisErrorCode.ErrorCode.Success:
00143
          print(error.message())
00144
          exit()
00145
00146 # Create the instrument handler
00147 handler = tracker.getInstrumentHandler(SQUIDNAME)
00148 print("Connection successful\n")
00149
00150 \mbox{\#} Create the TCP/IP socket and bind it to our host
00151 print("Starting server...")
```

17.21 writeinCSV.py 237

```
00152 server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
00153 activeSockets.append(server_socket)
00154 server_socket.bind((HOST, PORT))
00155
00156 # Listen for incoming connections
00157 server socket.listen(1)
00158
00159 print ("Server started successfully. Waiting for client connection...")
00160
00161 terminal_thread = threading.Thread(target=terminate_program)
00162 terminal_thread.start()
00163
00164 # Accept a client connection
00165 client_socket, client_address = server_socket.accept()
00166 activeSockets.append(client_socket)
00167
00168 # Connect the signals to send data to the client
00169 handler.activeDCDataReady.connect(lambda channel, data: send_data_to_client(client_socket, "DCData",
      data))
00170 handler.activeACDataReady.connect(lambda channel, data: send_data_to_client(client_socket, "ACData",
00171 handler.experimentNewElementStarting.connect(lambda channel, data: send_data_to_client(client_socket,
      "NewElement", data))
00172 handler.experimentStopped.connect(lambda channel: send_data_to_client(client_socket,
      "ExperimentCompleted", channel))
00174 \# Start the listening process in a separate thread
00175 listening_thread = threading.Thread(target=handle_client, args=(handler, client_socket))
00176 listening_thread.start()
00177
00178 # Start the OT event loop
00179 app.exec()
```

17.21 writeinCSV.py

This is an example of the writeinCSV.py file, which helps control Squidstat in parallel with other devices.

This is an example of the writeinCSV.py file, which helps control Squidstat in parallel with other devices.In this example, we inform other devices using SerialPortReader::writeData when a new element starts executing inside Squidstat, and also print the data received from the other device using the SerialPortReader::dataReceived signal. All operations occur in parallel with the Squidstat operation.

The Squidstat data is also written to a CSV file.

In detail:

- 1. The SerialPortReader class handles reading from and writing to a serial port of other device.
- 2. The WriteCSV class is responsible for writing data to a CSV file which is received from Squidstat.
- 3. The writingThread class is used to manage the data writing in csv file process in a separate thread.

```
00001 """! @example writeinCSV.py
00002 This is an example of the writeinCSV.py file, which helps control Squidstat in parallel with other
       devices.
00003 In this example, we inform other devices using SerialPortReader::writeData when a new element starts
00004 inside Squidstat, and also print the data received from the other device using the
      SerialPortReader::dataReceived signal.
00005 All operations occur in parallel with the Squidstat operation.
00006
00007 The Squidstat data is also written to a CSV file.
00008
00009 In detail:
00010 1. The `SerialPortReader` class handles reading from and writing to a serial port of other device. 00011 2. The `WriteCSV` class is responsible for writing data to a CSV file which is received from
       Squidstat.
00012 3. The `writingThread` class is used to manage the data writing in csv file process in a separate
       thread.
00013 """
00014
00015 import sys
00016 from PySide6.QtCore import QIODevice, QThread, QObject, Signal
00017 from PySide6.QtSerialPort import QSerialPort
```

238 Examples

```
00018 from PySide6.QtWidgets import QApplication
00019 from SquidstatPyLibrary import AisDeviceTracker
00020 from SquidstatPyLibrary import AisErrorCode
{\tt 00021~from~SquidstatPyLibrary~import~AisExperiment}
00022 from SquidstatPyLibrary import AisConstantCurrentElement 00023 from SquidstatPyLibrary import AisOpenCircuitElement
00025 # initialize the application
00026 app = QApplication([])
00027
00028 # convert incoming data to string with single line
00029 def convert_to_csv_line(data_list):
00030    return ','.join(str(item) for item in data_list)
00031
00032 # \cond EXCLUDE_FROM_DOX
00033 \# class for reading and writing data from serial port of other device.
00034 class SerialPortReader(QObject):
          # define the Qt signal.
dataReceived = Signal(str)
00035
00037
           # initialize self and port
00038
00039
           def __init__(self, port):
00040
               super().__init__()
00041
               self.port = port
00042
00043
           # open port if closed
00044
           # get data and decode
00045
           # emit data
00046
           def run(self):
               if not self.port.isOpen():
00047
00048
                   self.port.open(QIODevice.ReadOnly)
00049
               while self.port.isOpen():
00050
                   if self.port.waitForReadyRead():
00051
                        data = self.port.readAll().data().decode()
00052
                        self.dataReceived.emit(data)
00053
00054
          # open port if closed
# write data and encode
00055
           def writeData(self, data):
00056
00057
               if not self.port.isOpen():
00058
                    successfullyOpen = self.port.open(QIODevice.WriteOnly)
00059
                    if not successfullyOpen:
                        print("USB port is not open.")
00060
00061
               self.port.write(data.encode())
00062
00063
           # check if port is open, close port if open
00064
           def closePort(self):
00065
               if self.port.isOpen():
00066
                   self.port.close()
00067
00068
00069 # class for writing data to a csv file. data received from Squidstat.
00070 class WriteCSV:
00071
           \ensuremath{\mbox{\#}} init filename and file
          def __init__(self, filename):
    self.filename = filename
00072
00073
00074
               self.file = None
00075
00076
           # open file and write headers
00077
           def write_header(self, header):
00078
               if self.file is None:
                   self.file = open(self.filename, 'w')
00079
00080
               self.file.write(convert_to_csv_line(header) + '\n')
00081
00082
           # write data to file
00083
           def write_data(self, data):
00084
               if self.file is not None:
00085
                    self.file.write(convert to csv line(data) + '\n')
00086
00087
           # close file when we are done
00088
           def close(self):
00089
               if self.file is not None:
00090
                   self.file.close()
00091
00092
00093 # class to handle the write funcationality on seprate thread in pareller operation of Squidstat and
      other device.
00094 class writingThread(QThread):
           # define the signal.
writeData = Signal(float, float)
00095
00096
           stopTowrite = Signal()
00097
00098
00099
          # init self with values to be written
00100
           def __init__(self, csv_writer):
00101
               super().__init__()
00102
               self.timestamps = []
00103
               self.voltages = []
```

17.21 writeinCSV.py 239

```
00104
               self.csv_writer = csv_writer
00105
00106
           # setup data file with headers and connect the call back function on emitting of Qt signal.
          def run(self):
00107
00108
               self.csv_writer.write_header(['Timestamp', 'Working Electrode Voltage'])
00109
               self.writeData.connect(self.add data)
00110
               self.stopTowrite.connect(self.close)
00111
00112
           # add data into list as well as call back handler you can use to write the data in csv file.
00113
          def add_data(self, timestamp, voltage):
               self.timestamps.append(timestamp)
00114
00115
               self.voltages.append(voltage)
00116
               self.csv writer.write data([timestamp, voltage])
00117
00118
           # close writer for a channel
00119
          def close(self):
00120
              self.csv_writer.close()
00121 # \endcond
00123 # setup serial port of other device.
00124 serialPort = QSerialPort("COM3")
00125 # setup baud rate of other device.
00126 serialPort.setBaudRate(QSerialPort.Baud9600)
00127 # set up communication data type of other device.
00128 serialPort.setDataBits(QSerialPort.Data8)
00129 # define a serial port reader thread.
00130 serialPortReader = SerialPortReader(serialPort)
00131
00132
00133 \# function to write data from the serial port of other device.
00134 def writeDataToPort(data):
00135
          serialPortReader.writeData(data)
00136
00137 # instantiate a Squidstat device tracker
00138 tracker = AisDeviceTracker.Instance()
00139
00140 # interact with data and send experiments to Squidstat device
00141 def onNewDeviceConnected(deviceName):
00142
           # print which device has been connected
          print(f"Connected to: {deviceName}")
00143
00144
           # get handler using device name.
00145
          handler = tracker.getInstrumentHandler(deviceName)
           # if handler is present for the particular device then we can interact with the data and
00146
     upload/start/stop/puase/resume experiments
00147
          if handler:
00148
               # setup file name
00149
               csv_writer = WriteCSV('dataFile.csv')
00150
               \ensuremath{\text{\#}} add csv file to writing thread.
00151
               writingThread = writingThread(csv writer)
00152
               # start sub thread for write funcationality.
00153
               writingThread.start()
00154
00155
               # manages DC data input and output
               # add more variables if you want to print more data to the console
# send the signal to writing thread to write the information in csv file.
00156
00157
               handler.activeDCDataReady.connect(lambda channel, data: (
    print("timestamp:", "{:.9f}".format(data.timestamp), "workingElectrodeVoltage: ",
00158
00160
                         "{:.9f}".format(data.workingElectrodeVoltage)),
00161
                   writingThread.writeData.emit(data.timestamp, data.workingElectrodeVoltage)
00162
              ))
00163
00164
               # manages AC data input and output
00165
               # add more variables if you want to print more data to the console
               handler.activeACDataReady.connect(lambda channel, data: print("frequency:",
00166
      "\{:.9f\}".format(data.frequency),
00167
                                                                                  "absoluteImpedance: ",
      "{:.9f}".format(
00168
                                                                                      data.absoluteImpedance),
      "phaseAngle: ",
00169
      "{:.9f}".format(data.phaseAngle)))
00170
               # write when new node is starting
00171
              handler.experimentNewElementStarting.connect(lambda channel, data:
     writeDataToPort(data.stepName))
               # print when experiment has stopped, stop writeting thread
# send the signal to writing thread experiment is completed, which will close the csv file.
00172
00173
               handler.experimentStopped.connect(lambda channel: (print(f"Experiment completed on channel
00174
      {channel}"), writingThread.stopTowrite.emit(), app.quit()))
00175
00176
               # initialize an experiment
00177
               experiment = AisExperiment()
00178
00179
               \# define a constant current experiment at 0.1 A, with 1 s sampling time, and a duration of 10
00180
               ccElement = AisConstantCurrentElement(0.1, 1, 10)
               \# define an open circuit experiment with a duration of 10 s and a sampling time of 2 s
00181
00182
               opencircuitElement = AisOpenCircuitElement(10, 2)
```

240 Examples

```
00184
              # add constant current as the first element in the list
00185
              # element runs 1 time
              successfullyadd = experiment.appendElement(ccElement, 1)
# add open circuit as the second and thirds elements in the list
00186
00187
00188
               # element runs 2 times
00189
              successfullyadd |= experiment.appendElement(opencircuitElement, 2)
00190
00191
              if not successfullyadd:
00192
                  print("Error adding element to experiment")
00193
                  app.quit()
00194
00195
              # upload experiment to channel 1
00196
              error = handler.uploadExperimentToChannel(0, experiment)
00197
              if error.value() != AisErrorCode.Success:
00198
                  print(error.message())
00199
                   app.quit()
00200
00201
              # start experiment on channel 1
00202
              error = handler.startUploadedExperiment(0)
00203
              if error.value() != AisErrorCode.Success:
00204
                  print(error.message())
00205
                  app.quit()
00206
00207 # connect to device associated with the tracker
00208 tracker.newDeviceConnected.connect(onNewDeviceConnected)
00209
00210 \# Request the device to connect using com port 4
00211 error = tracker.connectToDeviceOnComPort("COM4")
00212 if error:
00213
          print(error.message())
00214
          sys.exit()
00215
00216 \# print the data which is received from another device (other than Squidstat)
00217 serialPortReader.dataReceived.connect(lambda data: print("Received data from COM port 3:", data))
00218 # setup a sub thread for read and write the information from the other device.
00219 serialPortThread = QThread()
00220 # pushes object to another thread
00221 serialPortReader.moveToThread(serialPortThread)
00222 # start the sub thread of other device.
00223 serialPortThread.start()
00224 # exit program
00225 sys.exit(app.exec())
```

Index

activeACDataReady	getMinAbsoluteCurrent, 59
AisInstrumentHandler, 140	getMinCurrent, 59
activeDCDataReady	getMindldt, 59
AisInstrumentHandler, 141	getName, 59
Advanced Experiment Logic, 3	getPotential, 60
AisACData, 45	getSamplingInterval, 60
numberOfCycles, 46	getVoltageRange, 60
AisCompRange, 46	isAutoRange, 60
AisCompRange, 46	isVoltageVsOCP, 60
getBandwidthIndex, 47	setApproxMaxCurrent, 60
getCompRangeName, 47	setAutoRange, 61
getStabilityFactor, 47	setMaxAbsoluteCurrent, 61
setBandwidthIndex, 47	setMaxCapacity, 61
setCompRangeName, 48	setMaxCurrent, 61
setStabilityFactor, 48	setMaxDuration, 62
AisConstantCurrentElement, 48	setMinAbsoluteCurrent, 62
AisConstantCurrentElement, 50	setMinCurrent, 62
getApproxMaxCurrent, 50	setMindIdt, 62
getApproxMaxVoltage, 50	setPotential, 63
getCategory, 50	setSamplingInterval, 63
getCurrent, 51	setVoltageRange, 63
getMaxCapacity, 51	setVoltageVsOCP, 63
getMaxDuration, 51	AisConstantPowerElement, 64
getMaxVoltage, 51	AisConstantPowerElement, 65
getMinSamplingVoltageDifference, 51	getCategory, 66
getMinVoltage, 52	getMaxCapacity, 66
getName, 52	getMaxCurrent, 66
getSamplingInterval, 52	getMaxDuration, 66
isAutoRange, 52	getMaxVoltage, 67
isAutoVoltageRange, 53	getMinCurrent, 67
setApproxMaxCurrent, 53	getMinVoltage, 67
setApproxMaxVoltage, 53	getName, 67
setAutoRange, 53	getPower, 68
setAutoVoltageRange, 53	getSamplingInterval, 68
setCurrent, 53	isCharge, 68
setMaxCapacity, 54	isMaximumVoltageVsOCP, 68
setMaxDuration, 54	isMinimumVoltageVsOCP, 68
setMaxVoltage, 54	
setMinSamplingVoltageDifference, 54	setCharge, 69 setMaxCapacity, 69
setMinVoltage, 55	setMaxCurrent, 69
setSamplingInterval, 55	setMaxDuration, 70
AisConstantPotElement, 55	setMaximumVoltageVsOCP, 70
AisConstant of Element, 55	setMaxVoltage, 70
getApproxMaxCurrent, 57	setMinCurrent, 70
	•
getCategory, 57	setMinimumVoltageVsOCP, 71
getMaxAbsoluteCurrent, 58	setMinVoltage, 71
getMaxCapacity, 58	setPower, 71
getMaxCurrent, 58	setSamplingInterval, 71
getMaxDuration, 58	AisConstantResistanceElement, 72

A:-O	+O++\/- +\/-OOD 00
AisConstantResistanceElement, 73	setStartVoltageVsOCP, 88
getCategory, 73	AisDataManipulator, 89
getMaxCapacity, 73	getBaseCurrent, 89
getMaxCurrent, 74	getBaseVoltage, 89
getMaxDuration, 74	getFrequency, 90
getMaxVoltage, 74	getPulseCurrent, 90
getMinCurrent, 74	getPulsePeriod, 90
getMinVoltage, 75	getPulseVoltage, 90
getName, 75	getPulseWidth, 90
getResistance, 75	isPulseCompleted, 90
-	•
getSamplingInterval, 75	loadPrimaryData, 91
isMaximumVoltageVsOCP, 75	setPulseType, 91
isMinimumVoltageVsOCP, 76	AisDCCurrentSweepElement, 92
setMaxCapacity, 76	AisDCCurrentSweepElement, 93
setMaxCurrent, 76	getAlphaFactor, 94
setMaxDuration, 77	getCategory, 94
setMaximumVoltageVsOCP, 77	getEndingCurrent, 94
setMaxVoltage, 77	getMaxVoltage, 94
setMinCurrent, 77	getMinVoltage, 94
setMinimumVoltageVsOCP, 78	getName, 95
•	
setMinVoltage, 78	getQuietTime, 95
setResistance, 78	getQuietTimeSamplingInterval, 95
setSamplingInterval, 78	getSamplingInterval, 95
AisCyclicVoltammetryElement, 79	getScanRate, 95
AisCyclicVoltammetryElement, 81	getStartingCurrent, 96
getAlphaFactor, 81	setAlphaFactor, 96
getApproxMaxCurrent, 81	setEndingCurrent, 96
getCategory, 81	setMaxVoltage, 96
getdEdt, 82	setMinVoltage, 97
getEndVoltage, 82	setQuietTime, 97
getFirstVoltageLimit, 82	setQuietTimeSamplingInterval, 97
getName, 82	setSamplingInterval, 97
getNumberOfCycles, 82	setScanRate, 97
getQuietTime, 83	setStartingCurrent, 98
getQuietTimeSamplingInterval, 83	AisDCData, 98
getSamplingInterval, 83	AisDCPotentialSweepElement, 99
getSecondVoltageLimit, 83	AisDCPotentialSweepElement, 100
getStartVoltage, 83	getAlphaFactor, 100
isAutoRange, 84	getApproxMaxCurrent, 101
isEndVoltageVsOCP, 84	getCategory, 101
isFirstVoltageLimitVsOCP, 84	getEndingPot, 101
isSecondVoltageLimitVsOCP, 84	getMaxAbsoluteCurrent, 101
isStartVoltageVsOCP, 85	getMinAbsoluteCurrent, 101
setAlphaFactor, 85	getName, 102
•	
setApproxMaxCurrent, 85	getQuietTime, 102
setAutoRange, 85	getQuietTimeSamplingInterval, 102
setdEdt, 85	getSamplingInterval, 102
setEndVoltage, 86	getScanRate, 102
setEndVoltageVsOCP, 86	getStartingPot, 103
setFirstVoltageLimit, 86	isAutoRange, 103
setFirstVoltageLimitVsOCP, 86	isEndVoltageVsOCP, 103
setNumberOfCycles, 87	isStartVoltageVsOCP, 103
setQuietTime, 87	setAlphaFactor, 103
setQuietTimeSamplingInterval, 87	setApproxMaxCurrent, 104
	• •
setSamplingInterval, 87	setAutoRange, 104
setSecondVoltageLimit, 87	setEndingPot, 104
setSecondVoltageLimitVsOCP, 88	setEndVoltageVsOCP, 104
setStartVoltage, 88	setMaxAbsoluteCurrent, 104

setMinAbsoluteCurrent, 105	getQuietTime, 123
setQuietTime, 105	getQuietTimeSamplingInterval, 123
setQuietTimeSamplingInterval, 105	getStartFreq, 124
setSamplingInterval, 105	getStepsPerDecade, 124
setScanRate, 106	setAmplitude, 124
setStartingPot, 106	setBiasCurrent, 124
setStartVoltageVsOCP, 106	setEndFreq, 124
AisDeviceTracker, 107	setMinimumCycles, 125
connectAllPluggedInDevices, 108	setQuietTime, 125
connectToDeviceOnComPort, 108	setQuietTimeSamplingInterval, 125
deviceDisconnected, 108	setStartFreq, 125
firmwareUpdateNotification, 109	setStepsPerDecade, 125
getConnectedDevices, 109	AisEISPotentiostaticElement, 126
getInstrumentHandler, 109	AisElSPotentiostaticElement, 127
Instance, 109	getAmplitude, 127
	- ·
newDeviceConnected, 110	getBiasVoltage, 127
saveLogToFile, 110	getCategory, 128
setLogFilePath, 110	getEndFreq, 128
updateFirmwareOnAllAvailableDevices, 111	getMinimumCycles, 128
updateFirmwareOnComPort, 111	getName, 128
AisDiffPulseVoltammetryElement, 112	getQuietTime, 128
AisDiffPulseVoltammetryElement, 113	getQuietTimeSamplingInterval, 128
getAlphaFactor, 114	getStartFreq, 129
getApproxMaxCurrent, 114	getStepsPerDecade, 129
getCategory, 114	isBiasVoltageVsOCP, 129
getEndVoltage, 115	setAmplitude, 129
getName, 115	setBiasVoltage, 129
getPulseHeight, 115	setBiasVoltageVsOCP, 130
getPulsePeriod, 115	setEndFreq, 130
getPulseWidth, 115	setMinimumCycles, 130
getQuietTime, 116	setQuietTime, 130
getQuietTimeSamplingInterval, 116	setQuietTimeSamplingInterval, 130
getStartVoltage, 116	setStartFreq, 131
getVStep, 116	setStepsPerDecade, 131
isAutoRange, 116	AisErrorCode, 131
isEndVoltageVsOCP, 117	BusyChannel, 132
isStartVoltageVsOCP, 117	ChannelNotBusy, 133
setAlphaFactor, 117	ConnectionFailed, 132
setApproxMaxCurrent, 118	DeviceCommunicationFailed, 133
setAutoRange, 118	DeviceNotFound, 132
setEndVoltage, 118	ErrorCode, 132
setEndVoltageVsOCP, 118	ExperimentAlreadyPaused, 133
setPulseHeight, 119	ExperimentAlreadyRun, 133
setPulsePeriod, 119	ExperimentIsEmpty, 132
setPulseWidth, 119	ExperimentNotUploaded, 132
setQuietTime, 119	ExperimentUploaded, 133
setQuietTimeSamplingInterval, 120	FailedToPauseExperiment, 133
setStartVoltage, 120	FailedToResumeExperiment, 133
setStartVoltageVsOCP, 120	FailedToSetChannelMaximumCurrent, 133
setVStep, 120	Failed To Set Channel Maximum Voltage, 133
•	Failed ToSetChannelMinimumCurrent, 133
AisEISGalvanostaticElement, 121	
AisEISGalvanostaticElement, 122	FailedToSetChannelMinimumTemperature, 133
getAmplitude, 122	FailedToSetChannelMinimumVoltage, 133
getBiasCurrent, 122	FailedToSetCompRange, 133
getCategory, 123	FailedToSetIRComp, 133
getEndFreq, 123	FailedToSetManualModeConstantCurrent, 133
getMinimumCycles, 123	FailedToSetManualModeConstantVoltage, 133
getName, 123	FailedToSetManualModeCurrentRange, 133

FailedToSetManualModeInOCP, 133	setFanSpeedVariable, 151
FailedToSetManualModeSamplingInterval, 133	setIRComp, 152
FailedToStopExperiment, 133	setLinkedChannels, 152
FailedToUploadExperiment, 133	setManualModeConstantCurrent, 152
FeatureNotSupported, 132	setManualModeConstantVoltage, 153
FirmwareNotSupported, 132	setManualModeCurrentAutorange, 154
InvalidChannel, 132	setManualModeCurrentRange, 154
InvalidParameters, 133	setManualModeOCP, 154
ManualExperimentNotRunning, 132	setManualModeSamplingInterval, 155
message, 134	setManualModeVoltageAutorange, 155
Success, 132	setManualModeVoltageRange, 156
Unknown, 132	skipExperimentStep, 156
value, 134	startIdleSampling, 157
AisExperiment, 134	startManualExperiment, 157
AisExperiment, 135	startUploadedExperiment, 157
appendElement, 135	stopExperiment, 158
appendSubExperiment, 135	uploadExperimentToChannel, 158, 159
···	•
getCategory, 136	AisMottSchottkyElement, 160
getDescription, 136	AisMottSchottkyElement, 161, 162
getExperimentName, 136	getAmplitude, 162
operator=, 136	getCategory, 162
setDescription, 137	getEndFrequency, 162
setExperimentName, 137	getEndingPotential, 162
AisExperimentNode, 137	getMinCycles, 163
AisInstrumentHandler, 138	getName, 163
activeACDataReady, 140	getQuietTime, 163
activeDCDataReady, 141	getQuietTimeSampInterval, 163
deviceError, 141	getStartFrequency, 163
eraseRecoverData, 141	getStartingPotential, 163
experimentNewElementStarting, 142	getStepQuietSampInterval, 164
experimentPaused, 142	getStepQuietTime, 164
experimentResumed, 142	getStepsPerDecade, 164
experimentStopped, 143	getVoltageStep, 164
getExperimentUTCStartTime, 143	isEndVoltageVsOCP, 164
getFreeChannels, 143	isStartVoltageVsOCP, 164
getLinkedChannels, 143	operator=, 165
getManualModeCurrentRangeList, 144	setAmplitude, 165
getManualModeVoltageRangeList, 144	setEndFrequency, 165
getNumberOfChannels, 144	setEndingPotential, 165
groundFloatStateChanged, 144	setEndVoltageVsOCP, 166
hasBipolarMode, 145	setMinCycles, 166
idleDCDataReady, 145	setQuietTime, 166
isChannelBusy, 145	setQuietTimeSampInterval, 166
isChannelPaused, 145	setStartFrequency, 166
pauseExperiment, 146	setStartingPotential, 167
recoverDataErased, 146	setStartVoltageVsOCP, 167
recoveryACDataReady, 146	setStepQuietSampInterval, 167
recoveryDCDataReady, 147	setStepQuietTime, 167
resetChannelLimits, 147	setStepsPerDecade, 167
resumeExperiment, 147	setVoltageStep, 168
setBipolarLinkedChannels, 147	AisNormalPulseVoltammetryElement, 168
setChannelMaximumCurrent, 148	AisNormalPulseVoltammetryElement, 170
setChannelMaximumTemperature, 148	getAlphaFactor, 170
setChannelMaximumVoltage, 149	getApproxMaxCurrent, 170
setChannelMinimumCurrent, 149	getCategory, 171
setChannelMinimumVoltage, 149	getEndVoltage, 171
setCompRange, 150	getName, 171
• •	
setFanSpeedMaximum, 150	getPulsePeriod, 171

getPulseWidth, 171	setAutoRange, 188
getQuietTime, 172	setEndVoltage, 188
getQuietTimeSamplingInterval, 172	setEndVoltageVsOCP, 188
getStartVoltage, 172	setPulseAmp, 188
getVStep, 172	setPulseFreq, 189
isAutoRange, 172	setQuietTime, 189
isEndVoltageVsOCP, 173	setQuietTimeSamplingInterval, 189
isStartVoltageVsOCP, 173	setStartVoltage, 189
setAlphaFactor, 173	setStartVoltageVsOCP, 189
setApproxMaxCurrent, 174	setVStep, 190
setAutoRange, 174	AisStaircasePotentialVoltammetryElement, 190
setEndVoltage, 174	AisStaircasePotentialVoltammetryElement, 192
setEndVoltageVsOCP, 174	getApproxMaxCurrent, 193
setPulsePeriod, 175	getCategory, 193
setPulseWidth, 175	getEndVoltage, 193
setQuietTime, 175	getFirstVoltageLimit, 193
setQuietTimeSamplingInterval, 175	getName, 193
setStartVoltage, 176	getNumberOfCycles, 193
setStartVoltageVsOCP, 176	getQuietTime, 194
setVStep, 176	getQuietTimeSamplingInterval, 194
AisOpenCircuitElement, 177	getSamplingInterval, 194
AisOpenCircuitElement, 178	getSecondVoltageLimit, 194
getApproxMaxVoltage, 178	getStartVoltage, 194
getCategory, 178	getStepDuration, 194
getMaxDuration, 178	getStepSize, 195
getMaxVoltage, 178	isAutorange, 195
getMindVdt, 179	isEndVoltageVsOCP, 195
getMinVoltage, 179	isFirstVoltageLimitVsOCP, 195
getName, 179	isSecondVoltageLimitVsOCP, 195
getSamplingInterval, 179	isStartVoltageVsOCP, 195
isAutoVoltageRange, 180	operator=, 196
setApproxMaxVoltage, 180	setApproxMaxCurrent, 196
setAutoVoltageRange, 180	setEndVoltage, 196
setMaxDuration, 180	setEndVoltageVsOCP, 196
setMaxVoltage, 180	setFirstVoltageLimit, 197
setMindVdt, 181	setFirstVoltageLimitVsOCP, 197
setMinVoltage, 181	setNumberOfCycles, 197
setSamplingInterval, 181	setQuietTime, 197
AisPulseType	setQuietTimeSamplingInterval, 197
Helpers, 43	setSamplingInterval, 198
AisSquareWaveVoltammetryElement, 182	setSecondVoltageLimit, 198
AisSquareWaveVoltammetryElement, 183	setSecondVoltageLimitVsOCP, 198
getAlphaFactor, 184	setStartVoltage, 198
getApproxMaxCurrent, 184	setStartVoltageVsOCP, 198
getCategory, 184	setStepDuration, 199
getEndVoltage, 185	setStepSize, 199
getName, 185	AisSteppedCurrentElement, 199
getPulseAmp, 185	AisSteppedCurrentElement, 200
getPulseFreq, 185	getApproxMaxVoltage, 201
getQuietTime, 185	getApproxMinVoltage, 201
getQuietTimeSamplingInterval, 186	getCategory, 201
getStartVoltage, 186	getEndCurrent, 201
getVStep, 186	getName, 202
isAutoRange, 186	getSamplingInterval, 202
isEndVoltageVsOCP, 186	getStartCurrent, 202
isStartVoltageVsOCP, 187	getStepDuration, 202
setAlphaFactor, 187	getStepSize, 202
setApproxMaxCurrent, 187	setApproxMaxVoltage, 202

setApproxMinVoltage, 203	Experiment Elements, 41
setEndCurrent, 203	ExperimentAlreadyPaused
setSamplingInterval, 203	AisErrorCode, 133
setStartCurrent, 203	ExperimentAlreadyRun
setStepDuration, 203	AisErrorCode, 133
setStepSize, 204	
•	ExperimentIsEmpty
AisSteppedVoltageElement, 204	AisErrorCode, 132
AisSteppedVoltageElement, 205	experimentNewElementStarting
getApproxMaxCurrent, 206	AisInstrumentHandler, 142
getCategory, 206	ExperimentNotUploaded
getEndVoltage, 206	AisErrorCode, 132
getName, 206	experimentPaused
getSamplingInterval, 206	AisInstrumentHandler, 142
getStartVoltage, 207	experimentResumed
getStepDuration, 207	AisInstrumentHandler, 142
getStepSize, 207	experimentStopped
isAutoRange, 207	AisInstrumentHandler, 143
isEndVoltageVsOCP, 207	ExperimentUploaded
isStartVoltageVsOCP, 207	AisErrorCode, 133
operator=, 208	Extra Examples, 9
setApproxMaxCurrent, 208	Extra Examples, o
setEndVoltage, 208	FailedToPauseExperiment
	AisErrorCode, 133
setEndVoltageVsOCP, 208	FailedToResumeExperiment
setSamplingInterval, 209	AisErrorCode, 133
setStartVoltage, 209	FailedToSetChannelMaximumCurrent
setStartVoltageVsOCP, 209	AisErrorCode, 133
setStepDuration, 209	
setStepSize, 209	FailedToSetChannelMaximumVoltage
appendElement	AisErrorCode, 133
AisExperiment, 135	FailedToSetChannelMinimumCurrent
appendSubExperiment	AisErrorCode, 133
AisExperiment, 135	FailedToSetChannelMinimumTemperature
	AisErrorCode, 133
BusyChannel	FailedToSetChannelMinimumVoltage
AisErrorCode, 132	AisErrorCode, 133
	FailedToSetCompRange
ChannelNotBusy	AisErrorCode, 133
AisErrorCode, 133	FailedToSetIRComp
connectAllPluggedInDevices	AisErrorCode, 133
AisDeviceTracker, 108	FailedToSetManualModeConstantCurrent
ConnectionFailed	AisErrorCode, 133
AisErrorCode, 132	FailedToSetManualModeConstantVoltage
connectToDeviceOnComPort	AisErrorCode, 133
AisDeviceTracker, 108	FailedToSetManualModeCurrentRange
Allo Bovico Hacker, 100	AisErrorCode, 133
Deprecated List, 35	FailedToSetManualModeInOCP
DeviceCommunicationFailed	
AisErrorCode, 133	AisErrorCode, 133
deviceDisconnected	FailedToSetManualModeSamplingInterval
	AisErrorCode, 133
AisDeviceTracker, 108	FailedToStopExperiment
deviceError	AisErrorCode, 133
AisInstrumentHandler, 141	FailedToUploadExperiment
DeviceNotFound	AisErrorCode, 133
AisErrorCode, 132	FeatureNotSupported
D	AisErrorCode, 132
eraseRecoverData	Finding COM Ports, 7
AisInstrumentHandler, 141	FirmwareNotSupported
ErrorCode	AisErrorCode, 132
AisErrorCode, 132	firmwareUpdateNotification

AisDeviceTracker, 109	AisCompRange, 47
	getConnectedDevices
getAlphaFactor	AisDeviceTracker, 109
AisCyclicVoltammetryElement, 81	getCurrent
AisDCCurrentSweepElement, 94	AisConstantCurrentElement, 51
AisDCPotentialSweepElement, 100	getdEdt
AisDiffPulseVoltammetryElement, 114	AisCyclicVoltammetryElement, 82
AisNormalPulseVoltammetryElement, 170	getDescription
AisSquareWaveVoltammetryElement, 184	AisExperiment, 136
getAmplitude	getEndCurrent
AisEISGalvanostaticElement, 122	AisSteppedCurrentElement, 201
AisEISPotentiostaticElement, 127	getEndFreq
AisMottSchottkyElement, 162	AisEISGalvanostaticElement, 123
getApproxMaxCurrent	AisEISPotentiostaticElement, 128
AisConstantCurrentElement, 50	getEndFrequency
AisConstantPotElement, 57	AisMottSchottkyElement, 162
AisCyclicVoltammetryElement, 81	getEndingCurrent
AisDCPotentialSweepElement, 101	AisDCCurrentSweepElement, 94
AisDiffPulseVoltammetryElement, 114	getEndingPot
AisNormalPulseVoltammetryElement, 170	AisDCPotentialSweepElement, 101
AisSquareWaveVoltammetryElement, 184	getEndingPotential
AisStaircasePotentialVoltammetryElement, 193	AisMottSchottkyElement, 162
AisSteppedVoltageElement, 206	getEndVoltage
getApproxMaxVoltage	AisCyclicVoltammetryElement, 82
AisConstantCurrentElement, 50	AisDiffPulseVoltammetryElement, 115
AisOpenCircuitElement, 178	AisNormalPulseVoltammetryElement, 171
AisSteppedCurrentElement, 201	AisSquareWaveVoltammetryElement, 185
getApproxMinVoltage	AisStaircasePotentialVoltammetryElement, 193
AisSteppedCurrentElement, 201	AisSteppedVoltageElement, 206
getBandwidthIndex	getExperimentName
AisCompRange, 47	AisExperiment, 136
getBaseCurrent	getExperimentUTCStartTime
AisDataManipulator, 89	AisInstrumentHandler, 143
getBaseVoltage	getFirstVoltageLimit
AisDataManipulator, 89	AisCyclicVoltammetryElement, 82
getBiasCurrent	AisStaircasePotentialVoltammetryElement, 193
AisEISGalvanostaticElement, 122	getFreeChannels
getBiasVoltage	_
AisEISPotentiostaticElement, 127	AisInstrumentHandler, 143
getCategory	getFrequency
AisConstantCurrentElement, 50	AisDataManipulator, 90
AisConstantPotElement, 57	getInstrumentHandler
AisConstant PowerElement, 66	AisDeviceTracker, 109
AisConstantResistanceElement, 73	getLinkedChannels
AisCyclicVoltammetryElement, 81	AisInstrumentHandler, 143
AisDCCurrentSweepElement, 94	getManualModeCurrentRangeList
AisDCPotentialSweepElement, 101	AisInstrumentHandler, 144
AisDiffPulseVoltammetryElement, 114	getManualModeVoltageRangeList
AisEISGalvanostaticElement, 123	AisInstrumentHandler, 144
AisEISPotentiostaticElement, 128	getMaxAbsoluteCurrent
	AisConstantPotElement, 58
AisExperiment, 136 AisMottSchottkyElement, 162	AisDCPotentialSweepElement, 101
	getMaxCapacity
AisOpenCircuitElement, 178	AisConstantCurrentElement, 51
AisOpenCircuitElement, 178	AisConstantPotElement, 58
AisSquareWaveVoltammetryElement, 184	AisConstantPowerElement, 66
AisStaircasePotentialVoltammetryElement, 193	AisConstantResistanceElement, 73
AisSteppedCurrentElement, 201	getMaxCurrent
AisSteppedVoltageElement, 206 getCompRangeName	AisConstantPotElement, 58
dercombhandename	

AisConstantPowerElement, 66	getNumberOfCycles
AisConstantResistanceElement, 74	AisCyclicVoltammetryElement, 82
getMaxDuration	AisStaircasePotentialVoltammetryElement, 193
AisConstantCurrentElement, 51	getPotential
AisConstantPotElement, 58	AisConstantPotElement, 60
AisConstantPowerElement, 66	getPower
AisConstantResistanceElement, 74	AisConstantPowerElement, 68
AisOpenCircuitElement, 178	getPulseAmp
getMaxVoltage	AisSquareWaveVoltammetryElement, 185
AisConstantCurrentElement, 51	getPulseCurrent
AisConstantPowerElement, 67	AisDataManipulator, 90
	•
AisConstantResistanceElement, 74	getPulseFreq
AisDCCurrentSweepElement, 94	AisSquareWaveVoltammetryElement, 185
AisOpenCircuitElement, 178	getPulseHeight
getMinAbsoluteCurrent	AisDiffPulseVoltammetryElement, 115
AisConstantPotElement, 59	getPulsePeriod
AisDCPotentialSweepElement, 101	AisDataManipulator, 90
getMinCurrent	AisDiffPulseVoltammetryElement, 115
AisConstantPotElement, 59	AisNormalPulseVoltammetryElement, 171
AisConstantPowerElement, 67	getPulseVoltage
AisConstantResistanceElement, 74	AisDataManipulator, 90
getMinCycles	getPulseWidth
AisMottSchottkyElement, 163	AisDataManipulator, 90
getMindIdt	AisDiffPulseVoltammetryElement, 115
AisConstantPotElement, 59	AisNormalPulseVoltammetryElement, 171
getMindVdt	getQuietTime
AisOpenCircuitElement, 179	AisCyclicVoltammetryElement, 83
getMinimumCycles	AisDCCurrentSweepElement, 95
AisEISGalvanostaticElement, 123	AisDCPotentialSweepElement, 102
AisEISPotentiostaticElement, 128	AisDiffPulseVoltammetryElement, 116
getMinSamplingVoltageDifference	AisEISGalvanostaticElement, 123
AisConstantCurrentElement, 51	AisEISPotentiostaticElement, 128
getMinVoltage	AisMottSchottkyElement, 163
AisConstantCurrentElement, 52	AisNormalPulseVoltammetryElement, 172
AisConstantPowerElement, 67	AisSquareWaveVoltammetryElement, 185
AisConstantResistanceElement, 75	AisStaircasePotentialVoltammetryElement, 194
AisDCCurrentSweepElement, 94	getQuietTimeSampInterval
AisOpenCircuitElement, 179	AisMottSchottkyElement, 163
	•
getName AisConstantCurrentFlorant F3	getQuietTimeSamplingInterval AisCyclicVoltammetryElement, 83
AisConstantCurrentElement, 52	· · · · · · · · · · · · · · · · · · ·
AisConstantPotElement, 59	AisDCCurrentSweepElement, 95
AisConstantPowerElement, 67	AisDCPotentialSweepElement, 102
AisConstantResistanceElement, 75	AisDiffPulseVoltammetryElement, 116
AisCyclicVoltammetryElement, 82	AisEISGalvanostaticElement, 123
AisDCCurrentSweepElement, 95	AisEISPotentiostaticElement, 128
AisDCPotentialSweepElement, 102	AisNormalPulseVoltammetryElement, 172
AisDiffPulseVoltammetryElement, 115	AisSquareWaveVoltammetryElement, 186
AisEISGalvanostaticElement, 123	AisStaircasePotentialVoltammetryElement, 194
AisEISPotentiostaticElement, 128	getResistance
AisMottSchottkyElement, 163	AisConstantResistanceElement, 75
AisNormalPulseVoltammetryElement, 171	getSamplingInterval
AisOpenCircuitElement, 179	AisConstantCurrentElement, 52
AisSquareWaveVoltammetryElement, 185	AisConstantPotElement, 60
AisStaircasePotentialVoltammetryElement, 193	AisConstantPowerElement, 68
AisSteppedCurrentElement, 202	AisConstantResistanceElement, 75
AisSteppedVoltageElement, 206	AisCyclicVoltammetryElement, 83
getNumberOfChannels	AisDCCurrentSweepElement, 95
AisInstrumentHandler, 144	AisDCPotentialSweepElement, 102
	·

AisOpenCircuitElement, 179	Guided Examples, 15
AisStaircasePotentialVoltammetryElement, 194	Llandling Cignala 21
AisSteppedCurrentElement, 202	Handling Signals, 31 hasBipolarMode
AisSteppedVoltageElement, 206	AisInstrumentHandler, 145
getScanRate	Helpers, 42
AisDCCurrentSweepElement, 95	AisPulseType, 43
AisDCPotentialSweepElement, 102	NormalPulse, 43
getSecondVoltageLimit	SquarewavePulse, 43
AisCyclicVoltammetryElement, 83	oquarewaver disc, 40
AisStaircasePotentialVoltammetryElement, 194	idleDCDataReady
getStabilityFactor AisCompRange, 47	AisInstrumentHandler, 145
getStartCurrent	Installing the Python Wrapper, 29
AisSteppedCurrentElement, 202	Installing with C++, 25
getStartFreq	Instance
AisEISGalvanostaticElement, 124	AisDeviceTracker, 109
AisEISPotentiostaticElement, 129	Instrument Control, 42
getStartFrequency	InvalidChannel
AisMottSchottkyElement, 163	AisErrorCode, 132
getStartingCurrent	InvalidParameters
AisDCCurrentSweepElement, 96	AisErrorCode, 133
getStartingPot	isAutoRange
AisDCPotentialSweepElement, 103	AisConstantCurrentElement, 52
getStartingPotential	AisConstantPotElement, 60
AisMottSchottkyElement, 163	AisCyclicVoltammetryElement, 84
getStartVoltage	AisDCPotentialSweepElement, 103
AisCyclicVoltammetryElement, 83	AisDiffPulseVoltammetryElement, 116
AisDiffPulseVoltammetryElement, 116	AisNormalPulseVoltammetryElement, 172
AisNormalPulseVoltammetryElement, 172	AisSquareWaveVoltammetryElement, 186
AisSquareWaveVoltammetryElement, 186	AisSteppedVoltageElement, 207
AisStaircasePotentialVoltammetryElement, 194	isAutorange
AisSteppedVoltageElement, 207	AisStaircasePotentialVoltammetryElement, 195
getStepDuration	isAutoVoltageRange
AisStaircasePotentialVoltammetryElement, 194	AisConstantCurrentElement, 53
AisSteppedCurrentElement, 202	AisOpenCircuitElement, 180
AisSteppedVoltageElement, 207	isBiasVoltageVsOCP
getStepQuietSampInterval	AisEISPotentiostaticElement, 129
AisMottSchottkyElement, 164	isChannelBusy
getStepQuietTime	AisInstrumentHandler, 145
AisMottSchottkyElement, 164	isChannelPaused
getStepSize	AisInstrumentHandler, 145
AisStaircasePotentialVoltammetryElement, 195	isCharge AisConstantPowerElement, 68
AisSteppedCurrentElement, 202	isEndVoltageVsOCP
AisSteppedVoltageElement, 207	AisCyclicVoltammetryElement, 84
getStepsPerDecade	AisDCPotentialSweepElement, 103
AisEISGalvanostaticElement, 124	AisDiffPulseVoltammetryElement, 117
AisEISPotentiostaticElement, 129	AisMottSchottkyElement, 164
AisMottSchottkyElement, 164	AisNormalPulseVoltammetryElement, 173
getVoltageRange	AisSquareWaveVoltammetryElement, 186
AisConstantPotElement, 60	AisStaircasePotentialVoltammetryElement, 195
getVoltageStep	AisSteppedVoltageElement, 207
AisMottSchottkyElement, 164	isFirstVoltageLimitVsOCP
getVStep AiaDiffPulseVoltammetryFlorent 116	AisCyclicVoltammetryElement, 84
AisDiffPulseVoltammetryElement, 116 AisNormalPulseVoltammetryElement, 172	AisStaircasePotentialVoltammetryElement, 195
AisSquareWaveVoltammetryElement, 172	isMaximumVoltageVsOCP
groundFloatStateChanged	AisConstantPowerElement, 68
AisInstrumentHandler, 144	AisConstantResistanceElement, 75
. nonea amona randon, 1111	isMinimumVoltageVsOCP

AisConstantPowerElement, 68	AisDCCurrentSweepElement, 96
AisConstantResistanceElement, 76	AisDCPotentialSweepElement, 103
isPulseCompleted	AisDiffPulseVoltammetryElement, 117
AisDataManipulator, 90	AisNormalPulseVoltammetryElement, 173
isSecondVoltageLimitVsOCP	AisSquareWaveVoltammetryElement, 187
AisCyclicVoltammetryElement, 84	setAmplitude
AisStaircasePotentialVoltammetryElement, 195	AisEISGalvanostaticElement, 124
isStartVoltageVsOCP	AisEISPotentiostaticElement, 129
AisCyclicVoltammetryElement, 85	AisMottSchottkyElement, 165
AisDCPotentialSweepElement, 103	setApproxMaxCurrent
AisDiffPulseVoltammetryElement, 117	AisConstantCurrentElement, 53
AisMottSchottkyElement, 164	AisConstantPotElement, 60
AisNormalPulseVoltammetryElement, 173	AisCyclicVoltammetryElement, 85
AisSquareWaveVoltammetryElement, 187	AisDCPotentialSweepElement, 104
AisStaircasePotentialVoltammetryElement, 195	AisDiffPulseVoltammetryElement, 118
AisSteppedVoltageElement, 207	AisNormalPulseVoltammetryElement, 174
isVoltageVsOCP	AisSquareWaveVoltammetryElement, 187
AisConstantPotElement, 60	AisStaircasePotentialVoltammetryElement, 196
land Driven w. Data	AisSteppedVoltageElement, 208
loadPrimaryData	setApproxMaxVoltage
AisDataManipulator, 91	AisConstantCurrentElement, 53
Manual Experiments, 17	AisOpenCircuitElement, 180
ManualExperimentNotRunning	AisSteppedCurrentElement, 202
	setApproxMinVoltage
AisErrorCode, 132	AisSteppedCurrentElement, 203
message	setAutoRange
AisErrorCode, 134	AisConstantCurrentElement, 53
newDeviceConnected	AisConstantPotElement, 61
AisDeviceTracker, 110	AisCyclicVoltammetryElement, 85
NormalPulse	AisDCPotentialSweepElement, 104
Helpers, 43	AisDiffPulseVoltammetryElement, 118
numberOfCycles	AisNormalPulseVoltammetryElement, 174
AisACData, 46	AisSquareWaveVoltammetryElement, 188
AlsAobala, 40	setAutoVoltageRange
operator=	AisConstantCurrentElement, 53
AisExperiment, 136	AisOpenCircuitElement, 180
AisMottSchottkyElement, 165	setBandwidthIndex
AisStaircasePotentialVoltammetryElement, 196	AisCompRange, 47
AisSteppedVoltageElement, 208	setBiasCurrent
,	AisEISGalvanostaticElement, 124
pauseExperiment	setBiasVoltage
AisInstrumentHandler, 146	AisEISPotentiostaticElement, 129
	setBiasVoltageVsOCP
recoverDataErased	AisEISPotentiostaticElement, 130
AisInstrumentHandler, 146	setBipolarLinkedChannels
recoveryACDataReady	AisInstrumentHandler, 147
AisInstrumentHandler, 146	setChannelMaximumCurrent
recoveryDCDataReady	AisInstrumentHandler, 148
AisInstrumentHandler, 147	setChannelMaximumTemperature
resetChannelLimits	AisInstrumentHandler, 148
AisInstrumentHandler, 147	setChannelMaximumVoltage
resumeExperiment	AisInstrumentHandler, 149
AisInstrumentHandler, 147	setChannelMinimumCurrent
Running an Experiment, 21	AisInstrumentHandler, 149
acyal agTaFile	setChannelMinimumVoltage
saveLogToFile	AisInstrumentHandler, 149
AisDeviceTracker, 110	setCharge
setAlphaFactor	AisConstantPowerElement, 69
AisCyclicVoltammetryElement, 85	

setCompRange	AisInstrumentHandler, 152
AisInstrumentHandler, 150	setManualModeConstantVoltage
setCompRangeName	AisInstrumentHandler, 153
AisCompRange, 48	setManualModeCurrentAutorange
setCurrent	AisInstrumentHandler, 154
AisConstantCurrentElement, 53	setManualModeCurrentRange
setdEdt	AisInstrumentHandler, 154
AisCyclicVoltammetryElement, 85	setManualModeOCP
setDescription	AisInstrumentHandler, 154
AisExperiment, 137	setManualModeSamplingInterval
setEndCurrent	AisInstrumentHandler, 155
AisSteppedCurrentElement, 203	setManualModeVoltageAutorange
setEndFreq	AisInstrumentHandler, 155
AisEISGalvanostaticElement, 124	setManualModeVoltageRange
AisEISPotentiostaticElement, 130	AisInstrumentHandler, 156
setEndFrequency	setMaxAbsoluteCurrent
·	
AisMottSchottkyElement, 165	AisConstantPotElement, 61
setEndingCurrent	AisDCPotentialSweepElement, 104
AisDCCurrentSweepElement, 96	setMaxCapacity
setEndingPot	AisConstantCurrentElement, 54
AisDCPotentialSweepElement, 104	AisConstantPotElement, 61
setEndingPotential	AisConstantPowerElement, 69
AisMottSchottkyElement, 165	AisConstantResistanceElement, 76
setEndVoltage	setMaxCurrent
AisCyclicVoltammetryElement, 86	AisConstantPotElement, 61
AisDiffPulseVoltammetryElement, 118	AisConstantPowerElement, 69
AisNormalPulseVoltammetryElement, 174	AisConstantResistanceElement, 76
AisSquareWaveVoltammetryElement, 188	setMaxDuration
AisStaircasePotentialVoltammetryElement, 196	AisConstantCurrentElement, 54
	AisConstantOdirentElement, 62
AisSteppedVoltageElement, 208	
setEndVoltageVsOCP	AisConstantPowerElement, 70
AisCyclicVoltammetryElement, 86	AisConstantResistanceElement, 77
AisDCPotentialSweepElement, 104	AisOpenCircuitElement, 180
AisDiffPulseVoltammetryElement, 118	setMaximumVoltageVsOCP
AisMottSchottkyElement, 166	AisConstantPowerElement, 70
AisNormalPulseVoltammetryElement, 174	AisConstantResistanceElement, 77
AisSquareWaveVoltammetryElement, 188	setMaxVoltage
AisStaircasePotentialVoltammetryElement, 196	AisConstantCurrentElement, 54
AisSteppedVoltageElement, 208	AisConstantPowerElement, 70
setExperimentName	AisConstantResistanceElement, 77
AisExperiment, 137	AisDCCurrentSweepElement, 96
setFanSpeedMaximum	AisOpenCircuitElement, 180
AisInstrumentHandler, 150	setMinAbsoluteCurrent
setFanSpeedVariable	AisConstantPotElement, 62
AisInstrumentHandler, 151	AisDCPotentialSweepElement, 105
setFirstVoltageLimit	setMinCurrent
AisCyclicVoltammetryElement, 86	AisConstantPotElement, 62
AisStaircasePotentialVoltammetryElement, 197	AisConstantPowerElement, 70
setFirstVoltageLimitVsOCP	AisConstantResistanceElement, 77
AisCyclicVoltammetryElement, 86	setMinCycles
AisStaircasePotentialVoltammetryElement, 197	AisMottSchottkyElement, 166
setIRComp	setMindIdt
AisInstrumentHandler, 152	AisConstantPotElement, 62
setLinkedChannels	setMindVdt
AisInstrumentHandler, 152	AisOpenCircuitElement, 181
setLogFilePath	setMinimumCycles
AisDeviceTracker, 110	AisEISGalvanostaticElement, 125
setManualModeConstantCurrent	AisEISPotentiostaticElement, 130
30tivianiamivoueOonstantOunent	AISEIGI OLEHLIOSLALIGERENEN, 130

setMinimumVoltageVsOCP	AisConstantCurrentElement, 55
AisConstantPowerElement, 71	AisConstantPotElement, 63
AisConstantResistanceElement, 78	AisConstantPowerElement, 71
setMinSamplingVoltageDifference	AisConstantResistanceElement, 78
AisConstantCurrentElement, 54	AisCyclicVoltammetryElement, 87
setMinVoltage	AisDCCurrentSweepElement, 97
AisConstantCurrentElement, 55	AisDCPotentialSweepElement, 105
AisConstantPowerElement, 71	AisOpenCircuitElement, 181
AisConstantResistanceElement, 78	AisStaircasePotentialVoltammetryElement, 198
AisDCCurrentSweepElement, 97	AisSteppedCurrentElement, 203
AisOpenCircuitElement, 181	AisSteppedVoltageElement, 209
setNumberOfCycles	setScanRate
AisCyclicVoltammetryElement, 87	AisDCCurrentSweepElement, 97
AisStaircasePotentialVoltammetryElement, 197	AisDCPotentialSweepElement, 106
setPotential	setSecondVoltageLimit
AisConstantPotElement, 63	AisCyclicVoltammetryElement, 87
setPower	AisStaircasePotentialVoltammetryElement, 198
AisConstantPowerElement, 71	setSecondVoltageLimitVsOCP
setPulseAmp	AisCyclicVoltammetryElement, 88
AisSquareWaveVoltammetryElement, 188	AisStaircasePotentialVoltammetryElement, 198
setPulseFreq	setStabilityFactor
AisSquareWaveVoltammetryElement, 189	AisCompRange, 48
setPulseHeight	setStartCurrent
AisDiffPulseVoltammetryElement, 119	AisSteppedCurrentElement, 203
setPulsePeriod	setStartFreq
AisDiffPulseVoltammetryElement, 119	AisEISGalvanostaticElement, 125
AisNormalPulseVoltammetryElement, 175	AisEISPotentiostaticElement, 131
setPulseType	setStartFrequency
AisDataManipulator, 91	AisMottSchottkyElement, 166
setPulseWidth	setStartingCurrent
AisDiffPulseVoltammetryElement, 119	AisDCCurrentSweepElement, 98
AisNormalPulseVoltammetryElement, 175	setStartingPot
setQuietTime	AisDCPotentialSweepElement, 106
AisCyclicVoltammetryElement, 87	setStartingPotential
AisDCCurrentSweepElement, 97	AisMottSchottkyElement, 167
AisDCPotentialSweepElement, 105	setStartVoltage
AisDiffPulseVoltammetryElement, 119	AisCyclicVoltammetryElement, 88
AisEISGalvanostaticElement, 125	AisDiffPulseVoltammetryElement, 120
AisEISPotentiostaticElement, 130	AisNormalPulseVoltammetryElement, 176
AisMottSchottkyElement, 166	AisSquareWaveVoltammetryElement, 189
AisNormalPulseVoltammetryElement, 175	AisStaircasePotentialVoltammetryElement, 198
AisSquareWaveVoltammetryElement, 189	AisSteppedVoltageElement, 209
AisStaircasePotentialVoltammetryElement, 197	setStartVoltageVsOCP
setQuietTimeSampInterval	AisCyclicVoltammetryElement, 88
AisMottSchottkyElement, 166	AisDCPotentialSweepElement, 106
setQuietTimeSamplingInterval	AisDiffPulseVoltammetryElement, 120
AisCyclicVoltammetryElement, 87	AisMottSchottkyElement, 167
AisDCCurrentSweepElement, 97	AisNormalPulseVoltammetryElement, 176
AisDCPotentialSweepElement, 105	AisSquareWaveVoltammetryElement, 189
AisDiffPulseVoltammetryElement, 120	AisStaircasePotentialVoltammetryElement, 198
AisEISGalvanostaticElement, 125	AisSteppedVoltageElement, 209
AisElSPotentiostaticElement, 130	setStepDuration
AisNormalPulseVoltammetryElement, 175	AisStaircasePotentialVoltammetryElement, 199
AisSquareWaveVoltammetryElement, 189	AisSteppedCurrentElement, 203
AisStaircasePotentialVoltammetryElement, 197	AisSteppedVoltageElement, 209
setResistance	setStepQuietSampInterval
AisConstantResistanceElement, 78	AisMottSchottkyElement, 167
setSamplingInterval	setStepQuietTime

```
AisMottSchottkyElement, 167
setStepSize
    AisStaircasePotentialVoltammetryElement, 199
    AisSteppedCurrentElement, 204
    AisSteppedVoltageElement, 209
setStepsPerDecade
    AisEISGalvanostaticElement, 125
    AisEISPotentiostaticElement, 131
    AisMottSchottkyElement, 167
setVoltageRange
    AisConstantPotElement, 63
setVoltageStep
    AisMottSchottkyElement, 168
setVoltageVsOCP
    AisConstantPotElement, 63
setVStep
    AisDiffPulseVoltammetryElement, 120
    AisNormalPulseVoltammetryElement, 176
    AisSquareWaveVoltammetryElement, 190
skipExperimentStep
    AisInstrumentHandler, 156
SquarewavePulse
    Helpers, 43
Squidstat API User Manual, 1
startIdleSampling
    AisInstrumentHandler, 157
startManualExperiment
    AisInstrumentHandler, 157
startUploadedExperiment
    AisInstrumentHandler, 157
stopExperiment
    AisInstrumentHandler, 158
Success
    AisErrorCode, 132
Unknown
    AisErrorCode, 132
updateFirmwareOnAllAvailableDevices
    AisDeviceTracker, 111
updateFirmwareOnComPort
    AisDeviceTracker, 111
Updating Firmware, 11
uploadExperimentToChannel
    AisInstrumentHandler, 158, 159
value
    AisErrorCode, 134
```