

# Squidstat API User Manual

Generated by Admiral Instruments LLC

August 30, 2024



<b>1 Squidstat API User Manual</b>	<b>1</b>
<b>2 Identifying USB Serial Ports</b>	<b>3</b>
2.0.0.1 Introduction . . . . .	3
2.0.0.2 Windows . . . . .	3
2.0.0.3 Mac . . . . .	3
2.0.0.4 Linux . . . . .	4
<b>3 Building API using CMake</b>	<b>5</b>
3.0.0.1 Introduction . . . . .	5
3.0.0.2 Clone API from Git . . . . .	5
3.0.0.3 Mac ARM64 (Apple Silicon) . . . . .	6
3.0.0.4 Cmake Installtion . . . . .	6
3.0.0.5 Build project . . . . .	7
<b>4 Running the API with Qt</b>	<b>9</b>
4.0.0.1 Introduction . . . . .	9
4.0.0.2 Clone API from Git . . . . .	9
4.0.0.3 Qt Installation. . . . .	10
4.0.0.4 Open Project with Qt and Cmake . . . . .	11
<b>5 Updating Firmware</b>	<b>15</b>
5.0.0.1 Introduction . . . . .	15
<b>6 The Basics of Running Experiments</b>	<b>17</b>
6.0.1 Creating A Custom Experiment . . . . .	17
6.0.2 Controlling The Experiment . . . . .	18
6.0.2.1 Creating Control Flow Logic Specific To A Handler . . . . .	19
6.0.2.2 Connecting Slots To Device-Tracker Signals . . . . .	20
<b>7 Manual Experiments</b>	<b>23</b>
<b>8 Automatically Update Firmware</b>	<b>25</b>
8.0.0.1 Introduction . . . . .	25
8.0.0.2 Implementation . . . . .	25
8.0.0.3 Full Example . . . . .	26
<b>9 Advanced Control Flow</b>	<b>29</b>
<b>10 Python Example</b>	<b>33</b>
10.0.0.1 Introduction . . . . .	33
10.0.0.2 How To Use Squidstatlibrary with Python. . . . .	33
10.0.0.3 Building a Custom Experiment with Python . . . . .	34
<b>11 Operating Squidstats Remotely</b>	<b>37</b>
11.0.0.1 Introduction . . . . .	37

11.0.0.2 Server Implementation . . . . .	37
11.0.0.3 Client Implementation . . . . .	39
11.0.0.4 Full Example . . . . .	40
<b>12 Hierarchical Index</b>	<b>43</b>
12.1 Class Hierarchy . . . . .	43
<b>13 Class Index</b>	<b>45</b>
13.1 Class List . . . . .	45
<b>14 File Index</b>	<b>47</b>
14.1 File List . . . . .	47
<b>15 Class Documentation</b>	<b>49</b>
15.1 AisACData Struct Reference . . . . .	49
15.1.1 Detailed Description . . . . .	50
15.1.2 Member Data Documentation . . . . .	50
15.1.2.1 numberOfCycles . . . . .	50
15.2 AisCompRange Class Reference . . . . .	50
15.2.1 Detailed Description . . . . .	51
15.2.2 Constructor & Destructor Documentation . . . . .	51
15.2.2.1 AisCompRange() . . . . .	51
15.2.3 Member Function Documentation . . . . .	51
15.2.3.1 getBandwidthIndex() . . . . .	51
15.2.3.2 getCompRangeName() . . . . .	52
15.2.3.3 getStabilityFactor() . . . . .	52
15.2.3.4 setBandwidthIndex() . . . . .	52
15.2.3.5 setCompRangeName() . . . . .	52
15.2.3.6 setStabilityFactor() . . . . .	53
15.3 AisConstantCurrentElement Class Reference . . . . .	53
15.3.1 Detailed Description . . . . .	55
15.3.2 Constructor & Destructor Documentation . . . . .	55
15.3.2.1 AisConstantCurrentElement() . . . . .	55
15.3.3 Member Function Documentation . . . . .	55
15.3.3.1 getApproxMaxCurrent() . . . . .	56
15.3.3.2 getApproxMaxVoltage() . . . . .	56
15.3.3.3 getCategory() . . . . .	56
15.3.3.4 getCurrent() . . . . .	57
15.3.3.5 getMaxCapacity() . . . . .	57
15.3.3.6 getMaxDuration() . . . . .	57
15.3.3.7 getMaxVoltage() . . . . .	57
15.3.3.8 getMinSamplingVoltageDifference() . . . . .	58
15.3.3.9 getMinVoltage() . . . . .	58
15.3.3.10 getName() . . . . .	58

---

15.3.3.11	getSamplingInterval()	59
15.3.3.12	isAutoRange()	59
15.3.3.13	isAutoVoltageRange()	59
15.3.3.14	setApproxMaxCurrent()	59
15.3.3.15	setApproxMaxVoltage()	60
15.3.3.16	setAutoRange()	60
15.3.3.17	setAutoVoltageRange()	60
15.3.3.18	setCurrent()	60
15.3.3.19	setMaxCapacity()	61
15.3.3.20	setMaxDuration()	61
15.3.3.21	setMaxVoltage()	61
15.3.3.22	setMinSamplingVoltageDifference()	62
15.3.3.23	setMinVoltage()	62
15.3.3.24	setSamplingInterval()	62
15.4	AisConstantPotElement Class Reference	63
15.4.1	Detailed Description	64
15.4.2	Constructor & Destructor Documentation	64
15.4.2.1	AisConstantPotElement()	65
15.4.3	Member Function Documentation	65
15.4.3.1	getApproxMaxCurrent()	65
15.4.3.2	getCategory()	65
15.4.3.3	getMaxCapacity()	66
15.4.3.4	getMaxCurrent()	66
15.4.3.5	getMaxDuration()	66
15.4.3.6	getMinCurrent()	67
15.4.3.7	getMindIdt()	67
15.4.3.8	getName()	67
15.4.3.9	getPotential()	68
15.4.3.10	getSamplingInterval()	68
15.4.3.11	isAutoRange()	68
15.4.3.12	isVoltageVsOCP()	68
15.4.3.13	setApproxMaxCurrent()	68
15.4.3.14	setAutoRange()	69
15.4.3.15	setMaxCapacity()	69
15.4.3.16	setMaxCurrent()	69
15.4.3.17	setMaxDuration()	70
15.4.3.18	setMinCurrent()	70
15.4.3.19	setMindIdt()	70
15.4.3.20	setPotential()	71
15.4.3.21	setSamplingInterval()	71
15.4.3.22	setVoltageRange()	71
15.4.3.23	setVoltageVsOCP()	71

15.5 AisConstantPowerElement Class Reference	72
15.5.1 Detailed Description	73
15.5.2 Constructor & Destructor Documentation	74
15.5.2.1 AisConstantPowerElement()	74
15.5.3 Member Function Documentation	74
15.5.3.1 getCategory()	74
15.5.3.2 getMaxCapacity()	74
15.5.3.3 getMaxCurrent()	75
15.5.3.4 getMaxDuration()	75
15.5.3.5 getMaxVoltage()	75
15.5.3.6 getMinCurrent()	76
15.5.3.7 getMinVoltage()	76
15.5.3.8 getName()	76
15.5.3.9 getPower()	77
15.5.3.10 getSamplingInterval()	77
15.5.3.11 isCharge()	77
15.5.3.12 isMaximumVoltageVsOCP()	77
15.5.3.13 isMinimumVoltageVsOCP()	78
15.5.3.14 setCharge()	78
15.5.3.15 setMaxCapacity()	78
15.5.3.16 setMaxCurrent()	79
15.5.3.17 setMaxDuration()	79
15.5.3.18 setMaximumVoltageVsOCP()	79
15.5.3.19 setMaxVoltage()	80
15.5.3.20 setMinCurrent()	80
15.5.3.21 setMinimumVoltageVsOCP()	80
15.5.3.22 setMinVoltage()	81
15.5.3.23 setPower()	81
15.5.3.24 setSamplingInterval()	81
15.6 AisConstantResistanceElement Class Reference	82
15.6.1 Detailed Description	83
15.6.2 Constructor & Destructor Documentation	83
15.6.2.1 AisConstantResistanceElement()	83
15.6.3 Member Function Documentation	84
15.6.3.1 getCategory()	84
15.6.3.2 getMaxCapacity()	84
15.6.3.3 getMaxCurrent()	84
15.6.3.4 getMaxDuration()	85
15.6.3.5 getMaxVoltage()	85
15.6.3.6 getMinCurrent()	85
15.6.3.7 getMinVoltage()	86
15.6.3.8 getName()	86

15.6.3.9	<a href="#">getResistance()</a>	86
15.6.3.10	<a href="#">getSamplingInterval()</a>	86
15.6.3.11	<a href="#">isMaximumVoltageVsOCP()</a>	87
15.6.3.12	<a href="#">isMinimumVoltageVsOCP()</a>	87
15.6.3.13	<a href="#">setMaxCapacity()</a>	87
15.6.3.14	<a href="#">setMaxCurrent()</a>	88
15.6.3.15	<a href="#">setMaxDuration()</a>	88
15.6.3.16	<a href="#">setMaximumVoltageVsOCP()</a>	88
15.6.3.17	<a href="#">setMaxVoltage()</a>	89
15.6.3.18	<a href="#">setMinCurrent()</a>	89
15.6.3.19	<a href="#">setMinimumVoltageVsOCP()</a>	89
15.6.3.20	<a href="#">setMinVoltage()</a>	90
15.6.3.21	<a href="#">setResistance()</a>	90
15.6.3.22	<a href="#">setSamplingInterval()</a>	90
15.7	<a href="#">AisCyclicVoltammetryElement Class Reference</a>	91
15.7.1	<a href="#">Detailed Description</a>	92
15.7.2	<a href="#">Constructor &amp; Destructor Documentation</a>	93
15.7.2.1	<a href="#">AisCyclicVoltammetryElement()</a>	93
15.7.3	<a href="#">Member Function Documentation</a>	93
15.7.3.1	<a href="#">getAlphaFactor()</a>	93
15.7.3.2	<a href="#">getApproxMaxCurrent()</a>	94
15.7.3.3	<a href="#">getCategory()</a>	94
15.7.3.4	<a href="#">getdEdt()</a>	94
15.7.3.5	<a href="#">getEndVoltage()</a>	94
15.7.3.6	<a href="#">getFirstVoltageLimit()</a>	95
15.7.3.7	<a href="#">getName()</a>	95
15.7.3.8	<a href="#">getNumberOfCycles()</a>	95
15.7.3.9	<a href="#">getQuietTime()</a>	95
15.7.3.10	<a href="#">getQuietTimeSamplingInterval()</a>	96
15.7.3.11	<a href="#">getSamplingInterval()</a>	96
15.7.3.12	<a href="#">getSecondVoltageLimit()</a>	96
15.7.3.13	<a href="#">getStartVoltage()</a>	96
15.7.3.14	<a href="#">isAutoRange()</a>	97
15.7.3.15	<a href="#">isEndVoltageVsOCP()</a>	97
15.7.3.16	<a href="#">isFirstVoltageLimitVsOCP()</a>	97
15.7.3.17	<a href="#">isSecondVoltageLimitVsOCP()</a>	98
15.7.3.18	<a href="#">isStartVoltageVsOCP()</a>	98
15.7.3.19	<a href="#">setAlphaFactor()</a>	98
15.7.3.20	<a href="#">setApproxMaxCurrent()</a>	98
15.7.3.21	<a href="#">setAutoRange()</a>	99
15.7.3.22	<a href="#">setdEdt()</a>	99
15.7.3.23	<a href="#">setEndVoltage()</a>	99

15.7.3.24 setEndVoltageVsOCP()	100
15.7.3.25 setFirstVoltageLimit()	100
15.7.3.26 setFirstVoltageLimitVsOCP()	100
15.7.3.27 setNumberOfCycles()	101
15.7.3.28 setQuietTime()	101
15.7.3.29 setQuietTimeSamplingInterval()	101
15.7.3.30 setSamplingInterval()	101
15.7.3.31 setSecondVoltageLimit()	102
15.7.3.32 setSecondVoltageLimitVsOCP()	102
15.7.3.33 setStartVoltage()	102
15.7.3.34 setStartVoltageVsOCP()	103
15.8 AisDataManipulator Class Reference	103
15.8.1 Detailed Description	104
15.8.2 Member Function Documentation	104
15.8.2.1 getBaseCurrent()	104
15.8.2.2 getBaseVoltage()	104
15.8.2.3 getFrequency()	105
15.8.2.4 getPulseCurrent()	105
15.8.2.5 getPulsePeriod()	105
15.8.2.6 getPulseVoltage()	105
15.8.2.7 getPulseWidth()	106
15.8.2.8 isPulseCompleted()	106
15.8.2.9 loadPrimaryData()	106
15.8.2.10 setPulseType() [1/2]	106
15.8.2.11 setPulseType() [2/2]	107
15.9 AisDCCurrentSweepElement Class Reference	107
15.9.1 Detailed Description	109
15.9.2 Constructor & Destructor Documentation	109
15.9.2.1 AisDCCurrentSweepElement()	109
15.9.3 Member Function Documentation	110
15.9.3.1 getAlphaFactor()	110
15.9.3.2 getCategory()	110
15.9.3.3 getEndingCurrent()	110
15.9.3.4 getMaxVoltage()	111
15.9.3.5 getMinVoltage()	111
15.9.3.6 getName()	111
15.9.3.7 getQuietTime()	112
15.9.3.8 getQuietTimeSamplingInterval()	112
15.9.3.9 getSamplingInterval()	112
15.9.3.10 getScanRate()	112
15.9.3.11 getStartingCurrent()	113
15.9.3.12 setAlphaFactor()	113



15.9.3.13 setEndingCurrent()	113
15.9.3.14 setMaxVoltage()	113
15.9.3.15 setMinVoltage()	115
15.9.3.16 setQuietTime()	115
15.9.3.17 setQuietTimeSamplingInterval()	115
15.9.3.18 setSamplingInterval()	116
15.9.3.19 setScanRate()	116
15.9.3.20 setStartingCurrent()	116
15.10 AisDCData Struct Reference	116
15.10.1 Detailed Description	117
15.11 AisDCPotentialSweepElement Class Reference	117
15.11.1 Detailed Description	119
15.11.2 Constructor & Destructor Documentation	119
15.11.2.1 AisDCPotentialSweepElement()	119
15.11.3 Member Function Documentation	120
15.11.3.1 getAlphaFactor()	120
15.11.3.2 getApproxMaxCurrent()	120
15.11.3.3 getCategory()	120
15.11.3.4 getEndingPot()	121
15.11.3.5 getMaxAbsoluteCurrent()	121
15.11.3.6 getMinAbsoluteCurrent()	121
15.11.3.7 getName()	122
15.11.3.8 getQuietTime()	122
15.11.3.9 getQuietTimeSamplingInterval()	122
15.11.3.10 getSamplingInterval()	122
15.11.3.11 getScanRate()	123
15.11.3.12 getStartingPot()	123
15.11.3.13 isAutoRange()	123
15.11.3.14 isEndVoltageVsOCP()	123
15.11.3.15 isStartVoltageVsOCP()	124
15.11.3.16 setAlphaFactor()	124
15.11.3.17 setApproxMaxCurrent()	124
15.11.3.18 setAutoRange()	125
15.11.3.19 setEndingPot()	125
15.11.3.20 setEndVoltageVsOCP()	125
15.11.3.21 setMaxAbsoluteCurrent()	125
15.11.3.22 setMinAbsoluteCurrent()	126
15.11.3.23 setQuietTime()	126
15.11.3.24 setQuietTimeSamplingInterval()	126
15.11.3.25 setSamplingInterval()	127
15.11.3.26 setScanRate()	127
15.11.3.27 setStartingPot()	127

15.11.3.28 setStartVoltageVsOCP()	128
15.12 AisDeviceTracker Class Reference	128
15.12.1 Detailed Description	129
15.12.2 Member Function Documentation	129
15.12.2.1 connectAllPluggedInDevices()	129
15.12.2.2 connectToDeviceOnComPort()	130
15.12.2.3 deviceDisconnected	131
15.12.2.4 getConnectedDevices()	131
15.12.2.5 getInstrumentHandler()	131
15.12.2.6 newDeviceConnected	132
15.12.2.7 saveLogToFile()	132
15.12.2.8 setLogFilePath()	133
15.12.2.9 updateFirmwareOnAllAvailableDevices()	133
15.12.2.10 updateFirmwareOnComPort()	134
15.13 AisDiffPulseVoltammetryElement Class Reference	134
15.13.1 Detailed Description	136
15.13.2 Constructor & Destructor Documentation	137
15.13.2.1 AisDiffPulseVoltammetryElement()	137
15.13.3 Member Function Documentation	137
15.13.3.1 getAlphaFactor()	137
15.13.3.2 getApproxMaxCurrent()	138
15.13.3.3 getCategory()	138
15.13.3.4 getEndVoltage()	138
15.13.3.5 getName()	138
15.13.3.6 getPulseHeight()	139
15.13.3.7 getPulsePeriod()	139
15.13.3.8 getPulseWidth()	139
15.13.3.9 getQuietTime()	140
15.13.3.10 getQuietTimeSamplingInterval()	140
15.13.3.11 getStartVoltage()	140
15.13.3.12 getVStep()	140
15.13.3.13 isAutoRange()	141
15.13.3.14 isEndVoltageVsOCP()	141
15.13.3.15 isStartVoltageVsOCP()	141
15.13.3.16 setAlphaFactor()	141
15.13.3.17 setApproxMaxCurrent()	142
15.13.3.18 setAutoRange()	142
15.13.3.19 setEndVoltage()	142
15.13.3.20 setEndVoltageVsOCP()	142
15.13.3.21 setPulseHeight()	143
15.13.3.22 setPulsePeriod()	143
15.13.3.23 setPulseWidth()	144

---

15.13.3.24 setQuietTime()	144
15.13.3.25 setQuietTimeSamplingInterval()	144
15.13.3.26 setStartVoltage()	144
15.13.3.27 setStartVoltageVsOCP()	145
15.13.3.28 setVStep()	145
15.14 AisEISGalvanostaticElement Class Reference	146
15.14.1 Detailed Description	147
15.14.2 Constructor & Destructor Documentation	147
15.14.2.1 AisEISGalvanostaticElement()	147
15.14.3 Member Function Documentation	148
15.14.3.1 getAmplitude()	148
15.14.3.2 getBiasCurrent()	148
15.14.3.3 getCategory()	148
15.14.3.4 getEndFreq()	148
15.14.3.5 getMinimumCycles()	149
15.14.3.6 getName()	149
15.14.3.7 getQuietTime()	149
15.14.3.8 getQuietTimeSamplingInterval()	149
15.14.3.9 getStartFreq()	150
15.14.3.10 getStepsPerDecade()	150
15.14.3.11 setAmplitude()	150
15.14.3.12 setBiasCurrent()	150
15.14.3.13 setEndFreq()	151
15.14.3.14 setMinimumCycles()	151
15.14.3.15 setQuietTime()	151
15.14.3.16 setQuietTimeSamplingInterval()	151
15.14.3.17 setStartFreq()	152
15.14.3.18 setStepsPerDecade()	152
15.15 AisEISPotentiostaticElement Class Reference	152
15.15.1 Detailed Description	154
15.15.2 Constructor & Destructor Documentation	154
15.15.2.1 AisEISPotentiostaticElement()	154
15.15.3 Member Function Documentation	155
15.15.3.1 getAmplitude()	155
15.15.3.2 getBiasVoltage()	155
15.15.3.3 getCategory()	155
15.15.3.4 getEndFreq()	155
15.15.3.5 getMinimumCycles()	156
15.15.3.6 getName()	156
15.15.3.7 getQuietTime()	156
15.15.3.8 getQuietTimeSamplingInterval()	156
15.15.3.9 getStartFreq()	157

15.15.3.10 getStepsPerDecade()	157
15.15.3.11 isBiasVoltageVsOCP()	157
15.15.3.12 setAmplitude()	157
15.15.3.13 setBiasVoltage()	158
15.15.3.14 setBiasVoltageVsOCP()	158
15.15.3.15 setEndFreq()	158
15.15.3.16 setMinimumCycles()	159
15.15.3.17 setQuietTime()	159
15.15.3.18 setQuietTimeSamplingInterval()	159
15.15.3.19 setStartFreq()	159
15.15.3.20 setStepsPerDecade()	160
15.16 AisErrorCode Class Reference	160
15.16.1 Detailed Description	161
15.16.2 Member Enumeration Documentation	161
15.16.2.1 ErrorCode	161
15.16.3 Member Function Documentation	162
15.16.3.1 message()	162
15.16.3.2 value()	163
15.17 AisExperiment Class Reference	163
15.17.1 Detailed Description	164
15.17.2 Constructor & Destructor Documentation	164
15.17.2.1 AisExperiment()	164
15.17.3 Member Function Documentation	164
15.17.3.1 appendElement()	164
15.17.3.2 appendSubExperiment()	165
15.17.3.3 getCategory()	165
15.17.3.4 getDescription()	166
15.17.3.5 getExperimentName()	166
15.17.3.6 operator=()	166
15.17.3.7 setDescription()	166
15.17.3.8 setExperimentName()	167
15.18 AisExperimentNode Struct Reference	167
15.18.1 Detailed Description	167
15.19 AisInstrumentHandler Class Reference	168
15.19.1 Detailed Description	170
15.19.2 Member Function Documentation	170
15.19.2.1 activeACDataReady	170
15.19.2.2 activeDCDataReady	171
15.19.2.3 deviceDisconnected	171
15.19.2.4 deviceError	171
15.19.2.5 eraseRecoverData()	171
15.19.2.6 experimentNewElementStarting	172

15.19.2.7 experimentPaused	172
15.19.2.8 experimentResumed	172
15.19.2.9 experimentStopped	173
15.19.2.10 getExperimentUTCStartTime()	173
15.19.2.11 getFreeChannels()	173
15.19.2.12 getLinkedChannels()	174
15.19.2.13 getManualModeCurrentRangeList()	174
15.19.2.14 getManualModeVoltageRangeList()	174
15.19.2.15 getNumberOfChannels()	175
15.19.2.16 groundFloatStateChanged	175
15.19.2.17 hasBipolarMode()	175
15.19.2.18 idleDCDataReady	176
15.19.2.19 isChannelBusy()	176
15.19.2.20 isChannelPaused()	176
15.19.2.21 pauseExperiment()	177
15.19.2.22 recoverDataErased	177
15.19.2.23 recoveryACDataReady	178
15.19.2.24 recoveryDCDataReady	178
15.19.2.25 resumeExperiment()	178
15.19.2.26 setBipolarLinkedChannels()	179
15.19.2.27 setCompRange()	179
15.19.2.28 setIRComp()	180
15.19.2.29 setLinkedChannels()	180
15.19.2.30 setManualModeConstantCurrent()	181
15.19.2.31 setManualModeConstantVoltage() [1/2]	181
15.19.2.32 setManualModeConstantVoltage() [2/2]	182
15.19.2.33 setManualModeCurrentAutorange()	182
15.19.2.34 setManualModeCurrentRange()	183
15.19.2.35 setManualModeOCP()	183
15.19.2.36 setManualModeSamplingInterval()	184
15.19.2.37 setManualModeVoltageAutorange()	184
15.19.2.38 setManualModeVoltageRange()	185
15.19.2.39 skipExperimentStep()	185
15.19.2.40 startIdleSampling()	186
15.19.2.41 startManualExperiment()	186
15.19.2.42 startUploadedExperiment()	187
15.19.2.43 stopExperiment()	187
15.19.2.44 uploadExperimentToChannel() [1/2]	188
15.19.2.45 uploadExperimentToChannel() [2/2]	189
15.20 AisMottSchottkyElement Class Reference	189
15.20.1 Detailed Description	191
15.20.2 Constructor & Destructor Documentation	192

15.20.2.1 AisMottSchottkyElement() [1/2]	192
15.20.2.2 AisMottSchottkyElement() [2/2]	193
15.20.3 Member Function Documentation	193
15.20.3.1 getAmplitude()	193
15.20.3.2 getCategory()	193
15.20.3.3 getEndFrequency()	193
15.20.3.4 getEndingPotential()	194
15.20.3.5 getMinCycles()	194
15.20.3.6 getName()	194
15.20.3.7 getQuietTime()	194
15.20.3.8 getQuietTimeSamplInterval()	195
15.20.3.9 getStartFrequency()	195
15.20.3.10 getStartingPotential()	195
15.20.3.11 getStepQuietSamplInterval()	195
15.20.3.12 getStepQuietTime()	196
15.20.3.13 getStepsPerDecade()	196
15.20.3.14 getVoltageStep()	196
15.20.3.15 isEndVoltageVsOCP()	196
15.20.3.16 isStartVoltageVsOCP()	197
15.20.3.17 operator=()	197
15.20.3.18 setAmplitude()	197
15.20.3.19 setEndFrequency()	197
15.20.3.20 setEndingPotential()	198
15.20.3.21 setEndVoltageVsOCP()	198
15.20.3.22 setMinCycles()	198
15.20.3.23 setQuietTime()	199
15.20.3.24 setQuietTimeSamplInterval()	199
15.20.3.25 setStartFrequency()	199
15.20.3.26 setStartingPotential()	199
15.20.3.27 setStartVoltageVsOCP()	200
15.20.3.28 setStepQuietSamplInterval()	200
15.20.3.29 setStepQuietTime()	200
15.20.3.30 setStepsPerDecade()	201
15.20.3.31 setVoltageStep()	201
15.21 AisNormalPulseVoltammetryElement Class Reference	201
15.21.1 Detailed Description	203
15.21.2 Constructor & Destructor Documentation	203
15.21.2.1 AisNormalPulseVoltammetryElement()	203
15.21.3 Member Function Documentation	204
15.21.3.1 getAlphaFactor()	204
15.21.3.2 getApproxMaxCurrent()	204
15.21.3.3 getCategory()	205

15.21.3.4 getEndVoltage()	205
15.21.3.5 getName()	205
15.21.3.6 getPulsePeriod()	205
15.21.3.7 getPulseWidth()	206
15.21.3.8 getQuietTime()	206
15.21.3.9 getQuietTimeSamplingInterval()	206
15.21.3.10 getStartVoltage()	206
15.21.3.11 getVStep()	207
15.21.3.12 isAutoRange()	207
15.21.3.13 isEndVoltageVsOCP()	207
15.21.3.14 isStartVoltageVsOCP()	208
15.21.3.15 setAlphaFactor()	208
15.21.3.16 setApproxMaxCurrent()	208
15.21.3.17 setAutoRange()	209
15.21.3.18 setEndVoltage()	209
15.21.3.19 setEndVoltageVsOCP()	209
15.21.3.20 setPulsePeriod()	210
15.21.3.21 setPulseWidth()	210
15.21.3.22 setQuietTime()	210
15.21.3.23 setQuietTimeSamplingInterval()	210
15.21.3.24 setStartVoltage()	211
15.21.3.25 setStartVoltageVsOCP()	211
15.21.3.26 setVStep()	211
15.22 AisOpenCircuitElement Class Reference	212
15.22.1 Detailed Description	213
15.22.2 Constructor & Destructor Documentation	213
15.22.2.1 AisOpenCircuitElement()	213
15.22.3 Member Function Documentation	214
15.22.3.1 getApproxMaxVoltage()	214
15.22.3.2 getCategory()	214
15.22.3.3 getMaxDuration()	214
15.22.3.4 getMaxVoltage()	215
15.22.3.5 getMindVdt()	215
15.22.3.6 getMinVoltage()	215
15.22.3.7 getName()	216
15.22.3.8 getSamplingInterval()	216
15.22.3.9 isAutoVoltageRange()	216
15.22.3.10 setApproxMaxVoltage()	216
15.22.3.11 setAutoVoltageRange()	217
15.22.3.12 setMaxDuration()	217
15.22.3.13 setMaxVoltage()	217
15.22.3.14 setMindVdt()	217

15.22.3.15 setMinVoltage()	218
15.22.3.16 setSamplingInterval()	218
15.23 AisSquareWaveVoltammetryElement Class Reference	218
15.23.1 Detailed Description	220
15.23.2 Constructor & Destructor Documentation	221
15.23.2.1 AisSquareWaveVoltammetryElement()	221
15.23.3 Member Function Documentation	221
15.23.3.1 getAlphaFactor()	221
15.23.3.2 getApproxMaxCurrent()	222
15.23.3.3 getCategory()	222
15.23.3.4 getEndVoltage()	222
15.23.3.5 getName()	222
15.23.3.6 getPulseAmp()	223
15.23.3.7 getPulseFreq()	223
15.23.3.8 getQuietTime()	223
15.23.3.9 getQuietTimeSamplingInterval()	223
15.23.3.10 getStartVoltage()	224
15.23.3.11 getVStep()	224
15.23.3.12 isAutoRange()	224
15.23.3.13 isEndVoltageVsOCP()	224
15.23.3.14 isStartVoltageVsOCP()	225
15.23.3.15 setAlphaFactor()	225
15.23.3.16 setApproxMaxCurrent()	225
15.23.3.17 setAutoRange()	226
15.23.3.18 setEndVoltage()	226
15.23.3.19 setEndVoltageVsOCP()	226
15.23.3.20 setPulseAmp()	227
15.23.3.21 setPulseFreq()	227
15.23.3.22 setQuietTime()	227
15.23.3.23 setQuietTimeSamplingInterval()	227
15.23.3.24 setStartVoltage()	228
15.23.3.25 setStartVoltageVsOCP()	228
15.23.3.26 setVStep()	228
15.24 AisStaircasePotentialVoltammetryElement Class Reference	229
15.24.1 Detailed Description	231
15.24.2 Constructor & Destructor Documentation	231
15.24.2.1 AisStaircasePotentialVoltammetryElement() [1/2]	231
15.24.2.2 AisStaircasePotentialVoltammetryElement() [2/2]	232
15.24.3 Member Function Documentation	232
15.24.3.1 getApproxMaxCurrent()	232
15.24.3.2 getCategory()	232
15.24.3.3 getEndVoltage()	232



15.24.3.4	<a href="#">getFirstVoltageLimit()</a>	233
15.24.3.5	<a href="#">getName()</a>	233
15.24.3.6	<a href="#">getNumberOfCycles()</a>	233
15.24.3.7	<a href="#">getQuietTime()</a>	233
15.24.3.8	<a href="#">getQuietTimeSamplingInterval()</a>	234
15.24.3.9	<a href="#">getSamplingInterval()</a>	234
15.24.3.10	<a href="#">getSecondVoltageLimit()</a>	234
15.24.3.11	<a href="#">getStartVoltage()</a>	234
15.24.3.12	<a href="#">getStepDuration()</a>	235
15.24.3.13	<a href="#">getStepSize()</a>	235
15.24.3.14	<a href="#">isAutorange()</a>	235
15.24.3.15	<a href="#">isEndVoltageVsOCP()</a>	235
15.24.3.16	<a href="#">isFirstVoltageLimitVsOCP()</a>	236
15.24.3.17	<a href="#">isSecondVoltageLimitVsOCP()</a>	236
15.24.3.18	<a href="#">isStartVoltageVsOCP()</a>	236
15.24.3.19	<a href="#">operator=()</a>	236
15.24.3.20	<a href="#">setApproxMaxCurrent()</a>	237
15.24.3.21	<a href="#">setEndVoltage()</a>	237
15.24.3.22	<a href="#">setEndVoltageVsOCP()</a>	237
15.24.3.23	<a href="#">setFirstVoltageLimit()</a>	237
15.24.3.24	<a href="#">setFirstVoltageLimitVsOCP()</a>	238
15.24.3.25	<a href="#">setNumberOfCycles()</a>	238
15.24.3.26	<a href="#">setQuietTime()</a>	238
15.24.3.27	<a href="#">setQuietTimeSamplingInterval()</a>	239
15.24.3.28	<a href="#">setSamplingInterval()</a>	239
15.24.3.29	<a href="#">setSecondVoltageLimit()</a>	239
15.24.3.30	<a href="#">setSecondVoltageLimitVsOCP()</a>	239
15.24.3.31	<a href="#">setStartVoltage()</a>	240
15.24.3.32	<a href="#">setStartVoltageVsOCP()</a>	240
15.24.3.33	<a href="#">setStepDuration()</a>	240
15.24.3.34	<a href="#">setStepSize()</a>	241
15.25	<a href="#">AisSteppedCurrentElement Class Reference</a>	241
15.25.1	<a href="#">Detailed Description</a>	242
15.25.2	<a href="#">Constructor &amp; Destructor Documentation</a>	242
15.25.2.1	<a href="#">AisSteppedCurrentElement()</a>	243
15.25.3	<a href="#">Member Function Documentation</a>	243
15.25.3.1	<a href="#">getApproxMaxVoltage()</a>	243
15.25.3.2	<a href="#">getApproxMinVoltage()</a>	243
15.25.3.3	<a href="#">getCategory()</a>	244
15.25.3.4	<a href="#">getEndCurrent()</a>	244
15.25.3.5	<a href="#">getName()</a>	244
15.25.3.6	<a href="#">getSamplingInterval()</a>	244

15.25.3.7	getStartCurrent()	245
15.25.3.8	getStepDuration()	245
15.25.3.9	getStepSize()	245
15.25.3.10	setApproxMaxVoltage()	245
15.25.3.11	setApproxMinVoltage()	246
15.25.3.12	setEndCurrent()	246
15.25.3.13	setSamplingInterval()	246
15.25.3.14	setStartCurrent()	246
15.25.3.15	setStepDuration()	248
15.25.3.16	setStepSize()	248
15.26	AisSteppedVoltageElement Class Reference	248
15.26.1	Detailed Description	250
15.26.2	Constructor & Destructor Documentation	250
15.26.2.1	AisSteppedVoltageElement() [1/2]	250
15.26.2.2	AisSteppedVoltageElement() [2/2]	251
15.26.3	Member Function Documentation	251
15.26.3.1	getApproxMaxCurrent()	251
15.26.3.2	getCategory()	251
15.26.3.3	getEndVoltage()	251
15.26.3.4	getName()	252
15.26.3.5	getSamplingInterval()	252
15.26.3.6	getStartVoltage()	252
15.26.3.7	getStepDuration()	252
15.26.3.8	getStepSize()	253
15.26.3.9	isAutoRange()	253
15.26.3.10	isEndVoltageVsOCP()	253
15.26.3.11	isStartVoltageVsOCP()	253
15.26.3.12	operator=()	253
15.26.3.13	setApproxMaxCurrent()	254
15.26.3.14	setEndVoltage()	254
15.26.3.15	setEndVoltageVsOCP()	254
15.26.3.16	setSamplingInterval()	255
15.26.3.17	setStartVoltage()	255
15.26.3.18	setStartVoltageVsOCP()	255
15.26.3.19	setStepDuration()	255
15.26.3.20	setStepSize()	256
<b>16</b>	<b>File Documentation</b>	<b>257</b>
16.1	AisCompRange.h	257
16.2	AisDataManipulator.h	257
16.3	AisDataPoints.h	258
16.4	AisDeviceTracker.h	259

16.5 AisErrorCode.h . . . . .	259
16.6 AisExperiment.h . . . . .	260
16.7 AisInstrumentHandler.h . . . . .	261
16.8 AisManipulatorType.h . . . . .	262
16.9 AisSquidstatGlobal.h . . . . .	262
16.10 AisAbstractElement.h . . . . .	263
16.11 AisConstantCurrentElement.h . . . . .	263
16.12 AisConstantPotElement.h . . . . .	264
16.13 AisConstantPowerElement.h . . . . .	265
16.14 AisConstantResistanceElement.h . . . . .	266
16.15 AisCyclicVoltammetryElement.h . . . . .	267
16.16 AisDCCurrentSweepElement.h . . . . .	268
16.17 AisDCPotentialSweepElement.h . . . . .	269
16.18 AisDiffPulseVoltammetryElement.h . . . . .	270
16.19 AisEISGalvanostaticElement.h . . . . .	271
16.20 AisEISPotentiostaticElement.h . . . . .	272
16.21 AisMottSchottkyElement.h . . . . .	272
16.22 AisNormalPulseVoltammetryElement.h . . . . .	273
16.23 AisOpenCircuitElement.h . . . . .	274
16.24 AisSquareWaveVoltammetryElement.h . . . . .	275
16.25 AisStaircasePotentialVoltammetryElement.h . . . . .	276
16.26 AisSteppedCurrentElement.h . . . . .	277
16.27 AisSteppedVoltageElement.h . . . . .	278
<b>Index</b>	<b>281</b>



## Chapter 1

# Squidstat API User Manual

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

Our API lets you programmatically start an experiment, pause an experiment and stop an experiment. You can [Download](#) our API from our git repository.

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

All examples included in this documentation, as well as additional examples, can be downloaded from our git repository for [C++](#) and [Python](#).

Let us start by going through [the basics](#).



## Chapter 2

# Identifying USB Serial Ports

### 2.0.0.1 Introduction

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

### 2.0.0.2 Windows

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

### 2.0.0.3 Mac

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

```
ls /dev/cu.*
```

4. Identify the USB serial port. Look for the entry in the list that corresponds to your USB device.  
Example output:

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

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

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

#### 2.0.0.4 Linux

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

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



## Chapter 3

# Building API using CMake

### 3.0.0.1 Introduction

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

### 3.0.0.2 Clone API from Git

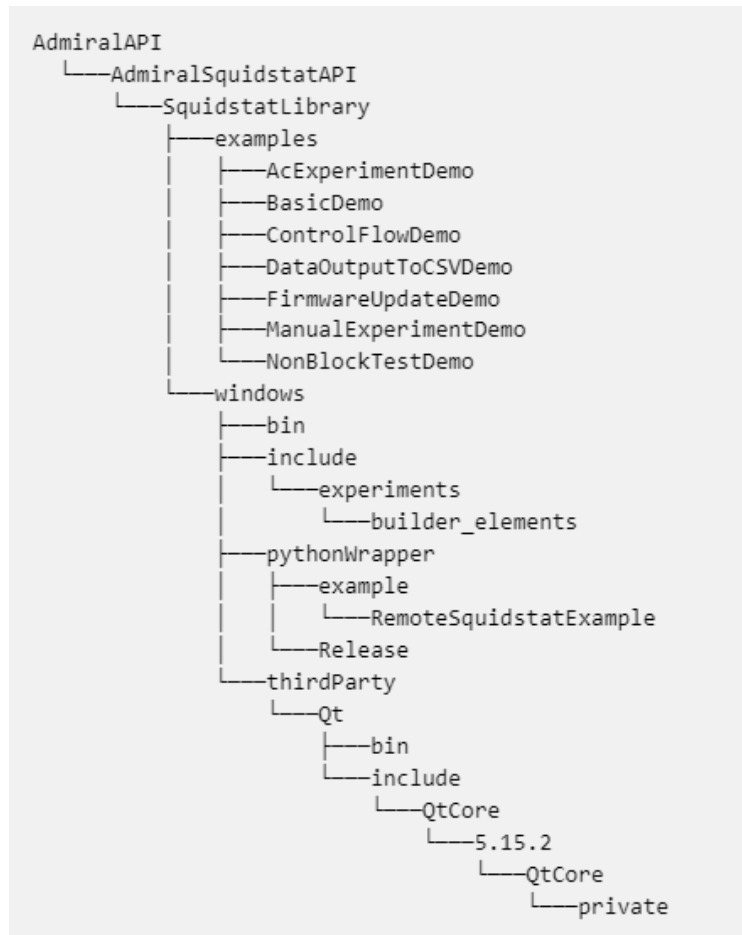
1. To clone the repository, you will need the Git tool. Depending on your platform, you can download the Git tool from [this link](#).
2. To verify if Git is properly installed, you can follow these steps:
  - Go to your desktop.
  - Open the command prompt.
  - Type `git -v` and press Enter.
  - If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

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

The result in the command prompt will look like this:

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

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



**Figure 3.1 API Bundler Directory Structure**

### 3.0.0.3 Mac ARM64 (Apple Silicon)

If the device you are on is a newer Mac with an ARM64 processor (M1, M2, etc), you will need to do an additional set of steps to include the Squidstat API.

1. Open a new Finder window in the cloned repository and navigate to `AdmiralSquidstatAPI/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 with the next steps as normal.

### 3.0.0.4 Cmake Installtion

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

### 3.0.0.5 Build project

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

```
cmake -B build -S "AdmiralSquidstatAPI/SquidstatLibrary/"
```

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

3. Build the project using the command below, which will compile all examples present the in `SquidstatLibrary` directory.

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



## Chapter 4

# Running the API with Qt

### 4.0.0.1 Introduction

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

### 4.0.0.2 Clone API from Git

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

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

The result in the command prompt will look like this:

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

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

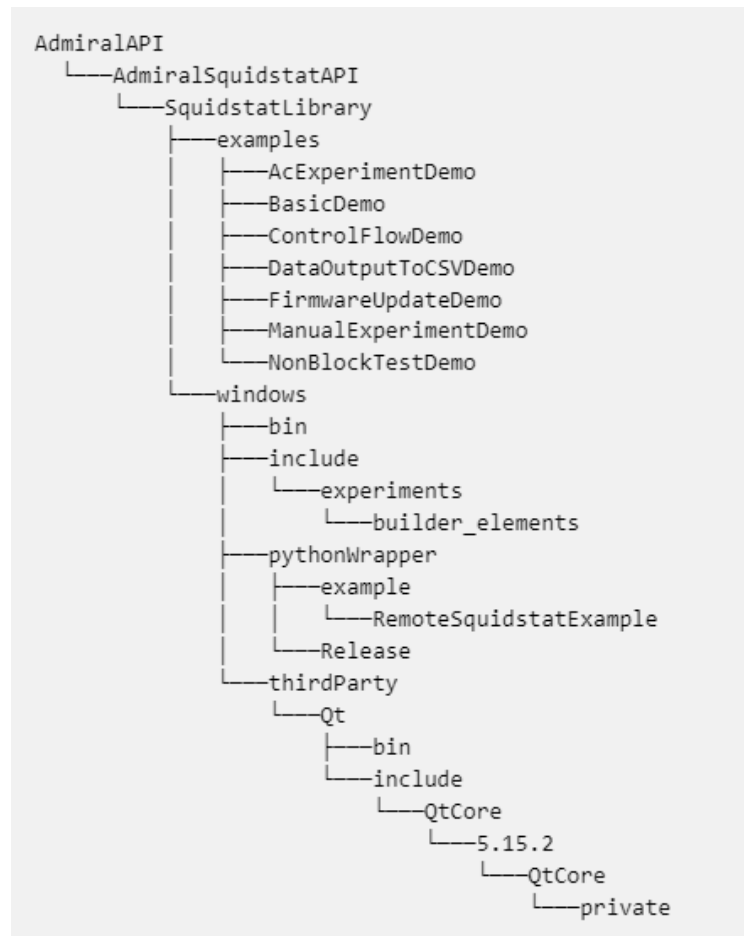


Figure 4.1 API Bundler Directory Structure

#### 4.0.0.3 Qt Installation.

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

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

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



Generated by Admiral Instruments LLC

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

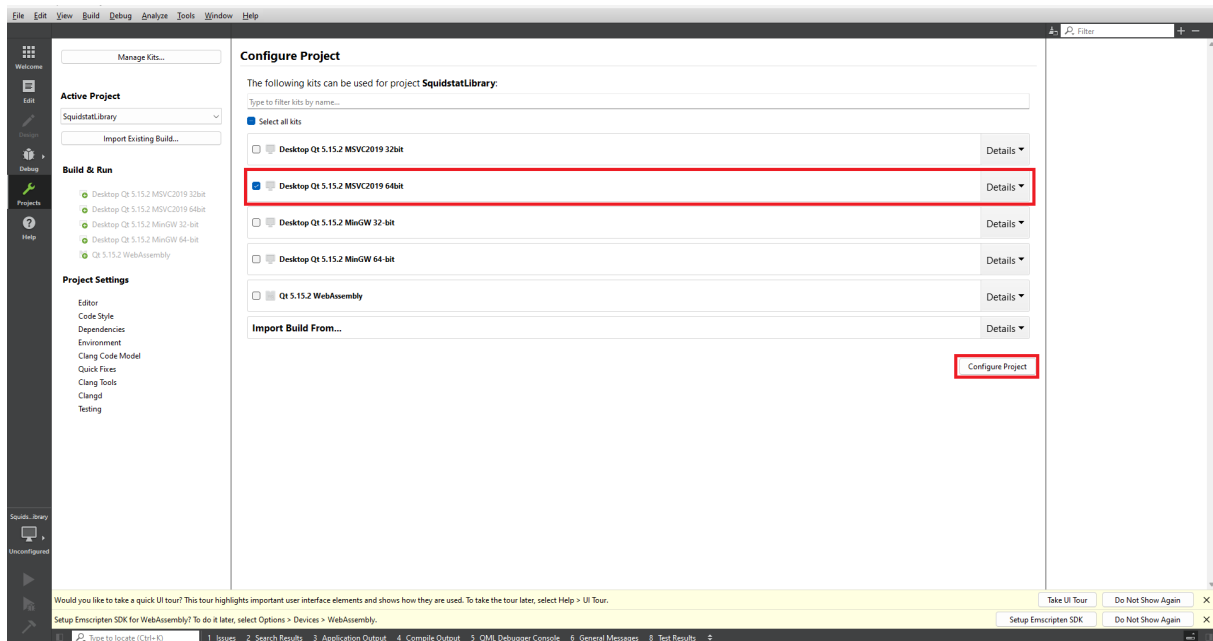


Figure 4.4 MSVC kit check mark

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



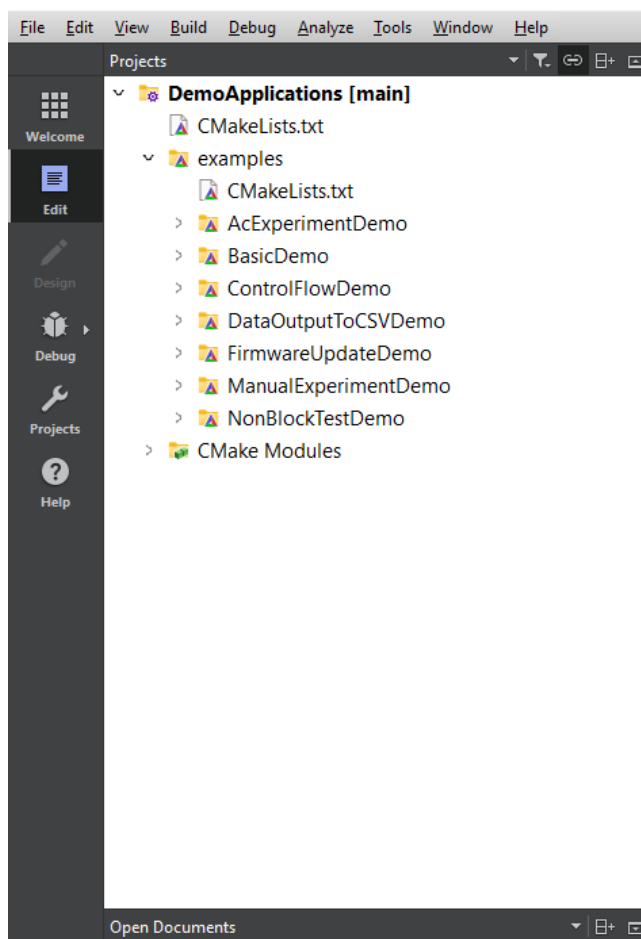


Figure 4.5 Qt project solution image

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

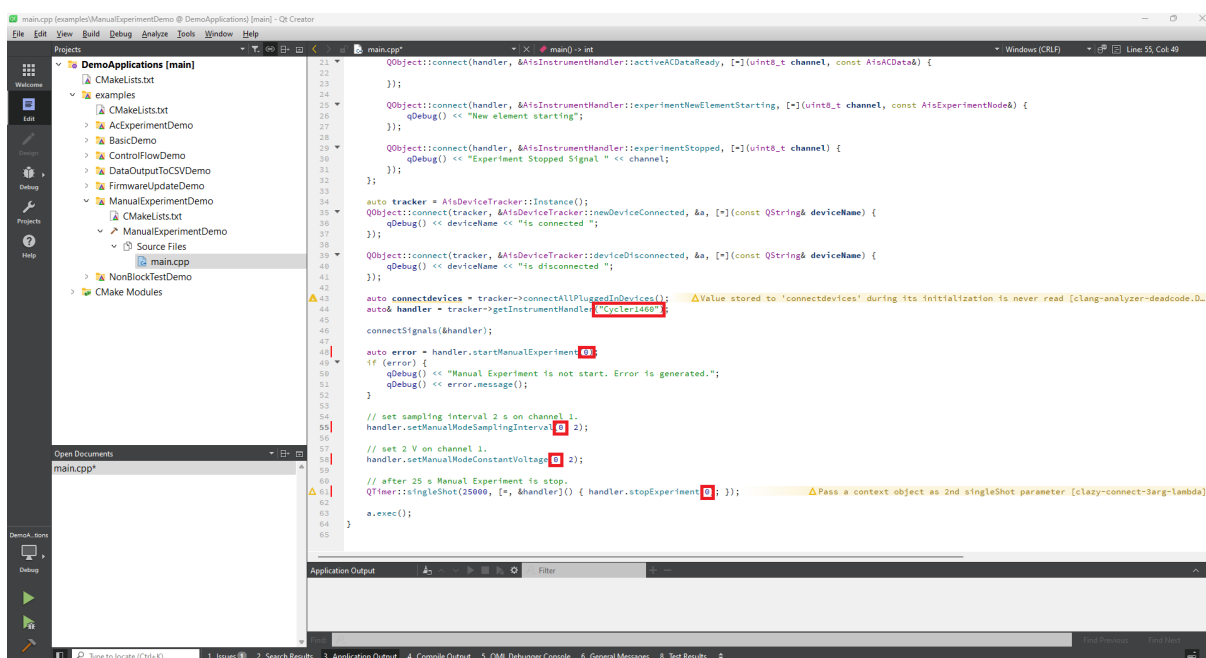


Figure 4.6 Qt manual experiment code

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

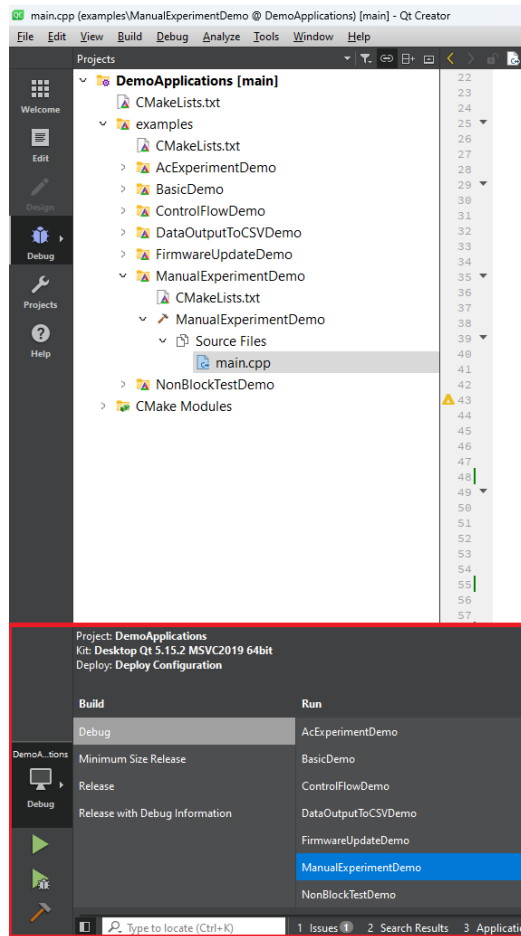


Figure 4.7 Qt selection of Debug and Release

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

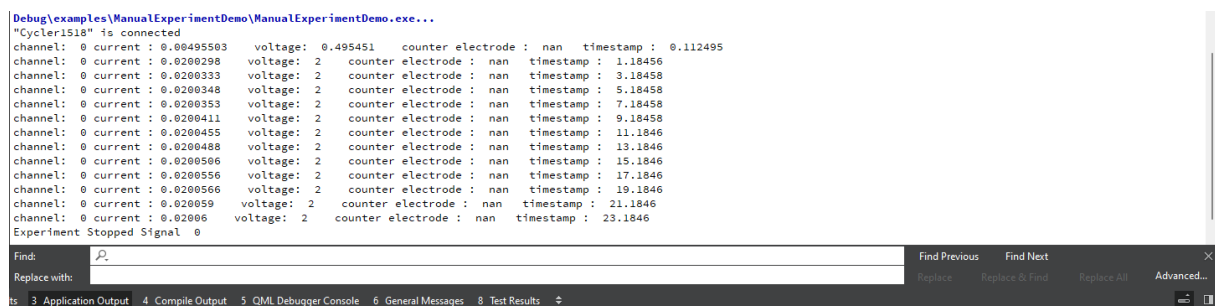


Figure 4.8 Qt output window

## Chapter 5

# Updating Firmware

### 5.0.0.1 Introduction

When connecting a new Squidstat or switching to a different version of the API, users may be required to update the firmware of the device. The following example will walk through the FirmwareUpdateDemo to demonstrate how to update the firmware for all connected Squidstats or select Squidstats.

>Note: If you are a Linux user and are having issues updating the firmware with your Squidstat, please refer to the announcement posted on the discussions page of our [GitHub](#) for troubleshooting guidelines.

**5.0.0.1.1 Import all required classes from SquidstatLibrary and Qt Library** `#include "AisDeviceTracker.h"`  
`#include <QCoreApplication>`  
`#include <qdebug.h>`  
`#include <qfileinfo.h>`

**5.0.0.1.2 Connect to the firmware progress notification** As the device updates the firmware, it will periodically send a notification that will include a percentage representing its progress. Here we will directly relay that message to the command line via the standard QT message handler `qInfo`. Because it is connected to a signal, it will be handled asynchronously.

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

**5.0.0.1.3 Request to update the firmware in all connected devices** At the point where we call `tracker->updateFirmwareOnAllAvailableDevices()`, the API will look for all devices connected and send them the signal to update their firmware. `numberOfDevices` will hold the number of devices that responded that their firmware was out of date. Once we call `exec` on our QT application, it will begin updating the firmware and reporting the update progress via the `firmwareUpdateNotification` signal that we connected to earlier.

```
auto numberOfDevices = tracker->updateFirmwareOnAllAvailableDevices();  
if (numberOfDevices == 0) {  
    qInfo() << "No devices need to be updated at this time. If this is incorrect, ensure all devices are  
        connected and powered on.";  
} else {  
    qInfo() << "Firmware update starting on " << numberOfDevices << "devices.";  
}
```

**5.0.0.1.4 Request to update the firmware to a specific device by comport** If the user wishes to specify a comport to update a specific Squidstat, that can be put in as an argument in this script. Here we check that the value follows the proper format via regex, and if it does, we begin the update via `tracker->updateFirmwareOnComPort (comPort)`. We will also report any error that may have occurred during this process.

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

**5.0.0.1.5 Start the QT Application** Once we call `exec` on our QT application, it will begin reporting the update progress via the `firmwareUpdateNotification` signal that we connected to earlier.

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

**5.0.0.1.6 Full Example** `#include "AisDeviceTracker.h"`

```
#include <QCoreApplication>
#include <qdebug.h>
#include <qfileinfo.h>
int main(int argc, char* argv[])
{
    QCoreApplication a(argc, argv);
    auto tracker = AisDeviceTracker::Instance();
    QObject::connect(tracker, &AisDeviceTracker::firmwareUpdateNotification, &a, [=](const QString& message)
    {
        qInfo() << message;
    });
    if (argc == 1) {
        auto numberOfDevices = tracker->updateFirmwareOnAllAvailableDevices();
        if (numberOfDevices == 0) {
            qInfo() << "No devices need to be updated at this time. If this is incorrect, ensure all devices
            are connected and powered on.";
        } else {
            qInfo() << "Firmware update starting on " << numberOfDevices << "devices.";
        }
    } else {
        if (argc == 2) {
            auto comPort = argv[1];

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

## Chapter 6

# The Basics of Running Experiments

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

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

### 6.0.1 Creating A Custom Experiment

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

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

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

Let us go through it step by step:

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

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

#### Note

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

We create another element of a different type.

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

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

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

**Note**

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

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

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

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

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

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

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

## 6.0.2 Controlling The Experiment

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

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

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

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

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

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

### 6.0.2.1 Creating Control Flow Logic Specific To A Handler

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

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

For a more complex logic for running a sequence of experiments, please refer to [this example](#)

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

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

```

        qDebug() << "Experiment Completed Signal " << channel;
    });
};

```

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

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

### 6.0.2.2 Connecting Slots To Device-Tracker Signals

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

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

```

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

```

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

#### Note

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

```

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

```



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

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

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

#### Note

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

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

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

To connect specific devices, you may alternatively call [AisDeviceTracker::connectToDeviceOnComPort](#) with a specific COM port.

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

Finally, we can start the application by calling:

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

In the [next section](#), we introduce a more advanced control flow.



## Chapter 7

# Manual Experiments

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

First, we do environment setup as usual:

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

Since we are doing a manual experiment, we will not create a custom experiment but jump to creating the logic.

The following is a simple logic:

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

You can see more advanced logic in in the [Advanced Control Flow](#).

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

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

### Note

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

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

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

You can see all the manual operations in [AisInstrumentHandler](#)

Finally, we start the application:

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



## Chapter 8

# Automatically Update Firmware

### 8.0.0.1 Introduction

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

### 8.0.0.2 Implementation

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

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

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

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

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

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

### 8.0.0.3 Full Example

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

```
    }  
  }  
});  
QObject::connect(tracker, &AisDeviceTracker::deviceDisconnected, &a, [=](const QString& deviceName) {  
    qDebug() << deviceName << "is disconnected";  
});  
// Attempt to connect to the device just before starting the QT app.  
auto error = tracker->connectToDeviceOnComPort(COMPORT);  
if (error != error.Success) {  
    if (error == error.FirmwareNotSupported) {  
        qDebug() << "Firmware is out of date for the device on" << COMPORT;  
        tracker->updateFirmwareOnComPort(COMPORT);  
    }  
    else {  
        qDebug() << "Error: " << error.message();  
    }  
}  
}  
a.exec();  
}
```





## Chapter 9

# Advanced Control Flow

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

First we will set the environment and create the experiments:

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

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

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

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

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

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

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

We then connect the tracker's signals as we have explained in more details [before](#).

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

---

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

Finally, you can start the application as follows:

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

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

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

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



## Chapter 10

# Python Example

### 10.0.0.1 Introduction

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

### 10.0.0.2 How To Use Squidstatlibrary with Python.

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

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

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

### 10.0.0.3 Building a Custom Experiment with Python

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

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

#### 10.0.0.3.2 Import experiment modules depended on the requirement.

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

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

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

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

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

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

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

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





# Chapter 11

## Operating Squidstats Remotely

### 11.0.0.1 Introduction

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

### 11.0.0.2 Server Implementation

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

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

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

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

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

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

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

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

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

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

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

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

### 11.0.0.3 Client Implementation

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

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

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

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

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

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

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

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

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

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

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

#### 11.0.0.4 Full Example

```
11.0.0.4.1 TCP_Server.py import os
import socket
import threading
from PySide2.QtWidgets import QApplication
from SquidstatPyLibrary import AisDeviceTracker
from SquidstatPyLibrary import AisExperiment
from SquidstatPyLibrary import AisOpenCircuitElement
from SquidstatPyLibrary import AisErrorCode
# Define the server address and port
HOST = 'localhost'
PORT = 12345
# The comport the Squidstat is connected to
SQUIDCOMPORT = "COM4"
SQUIDNAME = "Plus1700"
# Create the QT application
app = QApplication([])
activeSockets = []
# This will build a start the Open Circuit Potential experiment
def start_ocp_experiment(handler, durationSec=60):
    # Create an experiment with elements
    experiment = AisExperiment()
    ocpElement = AisOpenCircuitElement(durationSec, 1)
    experiment.appendElement(ocpElement, 1)
    # Upload the experiment to channel 0
    error = handler.uploadExperimentToChannel(0, experiment)
    if error.value() != AisErrorCode.ErrorCode.Success:
        return error
    # Start the experiment
    return(handler.startUploadedExperiment(0))
# Send a specified command to our Squidstat
def command_to_device(command, handler):
    #Check if we had an argument associated with the command
    splitCommand = command.split(" ")
    action = splitCommand[0]
    actionArg = 0
    if(len(splitCommand) > 1):
        try:
            actionArg = int(splitCommand[1])
        except:
            actionArg = 0
    response = None
    if action == 'startExperiment':
        #print("Starting experiment...")
        response = start_ocp_experiment(handler, actionArg)
    elif action == 'stopExperiment':
        #print("Stopping experiment...")
        response = handler.stopExperiment(0)
    else:
        #print("Invalid command:", command)
        pass
```

```

    return response
# Handle commands from the client
def handle_command(command, handler, client_socket):
    # Send a response back to the client
    responseMsg = "Unknown Command"
    response = command_to_device(command, handler)
    if(response != None):
        responseMsg = response.message()
        response = "{}".format(responseMsg)
        client_socket.send(response.encode())
# Listen for the client's messages, and disconnect signals and terminate program when finished
def handle_client(handler, client_socket):
    print("Client connected")
    while True:
        # Receive data from the client
        try:
            data = client_socket.recv(1024).decode()
        except ConnectionResetError:
            break
        # Check if the client has closed the connection
        if not data:
            break
        # Handle the command
        handle_command(data, handler, client_socket)
        handler.activeDCDataReady.disconnect()
        handler.activeACDataReady.disconnect()
        handler.experimentNewElementStarting.disconnect()
        handler.experimentStopped.disconnect()
        command_to_device("stopExperiment", handler)
        # Close the client socket
        client_socket.close()
        print("Client disconnected")
        os._exit(1)
# Send data the the client based on the type of event (Hooked up to signals)
def send_data_to_client(client_socket, event_type, data):
    if event_type == "DCData":
        message = "timestamp: {:.9f}, workingElectrodeVoltage: {:.9f}".format(data.timestamp,
            data.workingElectrodeVoltage)
    elif event_type == "ACData":
        message = "frequency: {:.9f}, absoluteImpedance: {:.9f}, phaseAngle: {:.9f}".format(data.frequency,
            data.absoluteImpedance, data.phaseAngle)
    elif event_type == "NewElement":
        message = "New Node beginning: {}, step number: {}, step sub: {}".format(data.stepName,
            data.stepNumber, data.substepNumber)
    elif event_type == "ExperimentCompleted":
        message = "Experiment Completed: {}".format(data)
    else:
        return
    client_socket.send(message.encode())
def terminate_program():
    print("Press <CTRL>+c to close the server")
    try:
        while True:
            input()
    except (EOFError, KeyboardInterrupt):
        pass
    for socket in activeSockets:
        socket.close()
    app.quit()
    os._exit(1)
# Create the device tracker and connect to the Squidstat we will be using
print(f"Attempting to connect to the Squidstat {SQUIDNAME} on {SQUIDCOMPORT}...")
tracker = AisDeviceTracker.Instance()
tracker.newDeviceConnected.connect(lambda deviceName: print("Device is Connected: %s" % deviceName))
error = tracker.connectToDeviceOnComPort(SQUIDCOMPORT)
if error.value() != AisErrorCode.ErrorCode.Success:
    print(error.message())
    exit()
# Create the instrument handler
handler = tracker.getInstrumentHandler(SQUIDNAME)
print("Connection successful\n")
# Create the TCP/IP socket and bind it to our host
print("Starting server...")
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
activeSockets.append(server_socket)
server_socket.bind((HOST, PORT))
# Listen for incoming connections
server_socket.listen(1)
print("Server started successfully. Waiting for client connection...")
terminal_thread = threading.Thread(target=terminate_program)
terminal_thread.start()
# Accept a client connection
client_socket, client_address = server_socket.accept()
activeSockets.append(client_socket)
# Connect the signals to send data to the client
handler.activeDCDataReady.connect(lambda channel, data: send_data_to_client(client_socket, "DCData", data))
handler.activeACDataReady.connect(lambda channel, data: send_data_to_client(client_socket, "ACData", data))

```

```

handler.experimentNewElementStarting.connect(lambda channel, data: send_data_to_client(client_socket,
    "NewElement", data))
handler.experimentStopped.connect(lambda channel: send_data_to_client(client_socket, "ExperimentCompleted",
    channel))
# Start the listening process in a separate thread
listening_thread = threading.Thread(target=handle_client, args=(handler, client_socket))
listening_thread.start()
# Start the QT event loop
app.exec_()

```

#### 11.0.0.4.2 TCP\_Client.py

```

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

```

## Chapter 12

# Hierarchical Index

### 12.1 Class Hierarchy

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

AisACData . . . . .	49
AisCompRange . . . . .	50
AisConstantCurrentElement . . . . .	53
AisConstantPotElement . . . . .	63
AisConstantPowerElement . . . . .	72
AisConstantResistanceElement . . . . .	82
AisCyclicVoltammetryElement . . . . .	91
AisDataManipulator . . . . .	103
AisDCCurrentSweepElement . . . . .	107
AisDCData . . . . .	116
AisDCPotentialSweepElement . . . . .	117
AisDiffPulseVoltammetryElement . . . . .	134
AisEISGalvanostaticElement . . . . .	146
AisEISPotentiostaticElement . . . . .	152
AisErrorCode . . . . .	160
AisExperiment . . . . .	163
AisExperimentNode . . . . .	167
AisMottSchottkyElement . . . . .	189
AisNormalPulseVoltammetryElement . . . . .	201
AisOpenCircuitElement . . . . .	212
AisSquareWaveVoltammetryElement . . . . .	218
AisStaircasePotentialVoltammetryElement . . . . .	229
AisSteppedCurrentElement . . . . .	241
AisSteppedVoltageElement . . . . .	248
QObject	
AisDeviceTracker . . . . .	128
AisInstrumentHandler . . . . .	168





## Chapter 13

# Class Index

### 13.1 Class List

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

<a href="#">AisACData</a>	Structure containing AC data information . . . . .	49
<a href="#">AisCompRange</a>	This class has advanced options controlling the device stability including the bandwidth index and the stability factor . . . . .	50
<a href="#">AisConstantCurrentElement</a>	Experiment that simulates a constant current flow with more advance options for stopping the experiment. 53	
<a href="#">AisConstantPotElement</a>	Experiment that simulates a constant applied voltage. 63	
<a href="#">AisConstantPowerElement</a>	This experiment simulates a constant power, charge or discharge". 72	
<a href="#">AisConstantResistanceElement</a>	This element/experiment simulates a constant resistance load. 82	
<a href="#">AisCyclicVoltammetryElement</a>	This experiment sweeps the potential of the working electrode back and forth between the <b>first voltage-limit</b> and the <b>second voltage-limit</b> at a constant <b>scan rate (dE/dt)</b> for a specified number of <b>cycles</b> . . . . .	91
<a href="#">AisDataManipulator</a>	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) . . . . .	103
<a href="#">AisDCCurrentSweepElement</a>	This experiment performs a DC current sweep from the <b>starting current</b> to the <b>ending current</b> which progresses linearly according to the <b>scan rate</b> . . . . .	107
<a href="#">AisDCData</a>	Structure containing DC data information . . . . .	116
<a href="#">AisDCPotentialSweepElement</a>	This experiment performs a DC potential sweep from the <b>starting current</b> to the <b>ending current</b> which progresses linearly according to the <b>scan rate</b> . . . . .	117

<a href="#">AisDeviceTracker</a>	
This class is used track device connections to the computer. It can establish connection with plugged-in devices. It also provides instrument handlers specific to each connected device which can provide control of the specific device like starting experiments . . . . .	128
<a href="#">AisDiffPulseVoltammetryElement</a>	
In this experiment, the working electrode holds at a <b>starting potential</b> during the <b>quiet time</b> . Then it applies a train of pulses superimposed on a staircase waveform, with a uniform <b>potential step</b> size. The potential continues to step until the <b>final potential</b> is reached . . . . .	134
<a href="#">AisEISGalvanostaticElement</a>	
This experiment records the complex impedance of the experimental cell in galvanostatic mode, starting from the <b>start frequency</b> and sweeping through towards the <b>end frequency</b> , with a fixed number of frequency <b>steps per decade</b> . . . . .	146
<a href="#">AisEISPotentiostaticElement</a>	
This experiment records the complex impedance of the experimental cell in potentiostatic mode, starting from the <b>start frequency</b> and sweeping through towards the <b>end frequency</b> , with a fixed number of frequency <b>steps per decade</b> . . . . .	152
<a href="#">AisErrorCode</a>	
This class contains the possible error codes returned to the user when working with the API . .	160
<a href="#">AisExperiment</a>	
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	163
<a href="#">AisExperimentNode</a>	
Structure containing some information regarding the running element . . . . .	167
<a href="#">AisInstrumentHandler</a>	
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 . . . . .	168
<a href="#">AisMottSchottkyElement</a>	
This class performs Mott-Schottky analysis on the working electrode for a specified range of potentials . . . . .	189
<a href="#">AisNormalPulseVoltammetryElement</a>	
This experiment holds the working electrode at a <b>baseline potential</b> during the <b>quiet time</b> , then applies a train of pulses, which increase in amplitude until the <b>final potential</b> is reached . . . .	201
<a href="#">AisOpenCircuitElement</a>	
This experiment observes the <b>open circuit potential</b> of the working electrode for a specific period of time.	212
<a href="#">AisSquareWaveVoltammetryElement</a>	
This experiment holds the working electrode at the <b>starting potential</b> during the <b>quiet time</b> . Then it applies a train of square pulses superimposed on a staircase waveform with a uniform <b>potential step</b> magnitude . . . . .	218
<a href="#">AisStaircasePotentialVoltammetryElement</a>	
<a href="#">AisStaircasePotentialVoltammetryElement</a> class represents an element for staircase potential voltammetry experiments. It inherits from <a href="#">AisAbstractElement</a> . . . . .	229
<a href="#">AisSteppedCurrentElement</a>	
A class representing an experiment to apply the stepped current.	241
<a href="#">AisSteppedVoltageElement</a>	
A class representing an experiment to apply the stepped voltage . . . . .	248

## Chapter 14

# File Index

### 14.1 File List

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

<a href="#">AisCompRange.h</a>	257
<a href="#">AisDataManipulator.h</a>	257
<a href="#">AisDataPoints.h</a>	258
<a href="#">AisDeviceTracker.h</a>	259
<a href="#">AisErrorCode.h</a>	259
<a href="#">AisExperiment.h</a>	260
<a href="#">AisInstrumentHandler.h</a>	261
<a href="#">AisManipulatorType.h</a>	262
<a href="#">AisSquidstatGlobal.h</a>	262
<a href="#">AisAbstractElement.h</a>	263
<a href="#">AisConstantCurrentElement.h</a>	263
<a href="#">AisConstantPotElement.h</a>	264
<a href="#">AisConstantPowerElement.h</a>	265
<a href="#">AisConstantResistanceElement.h</a>	266
<a href="#">AisCyclicVoltammetryElement.h</a>	267
<a href="#">AisDCCurrentSweepElement.h</a>	268
<a href="#">AisDCPotentialSweepElement.h</a>	269
<a href="#">AisDiffPulseVoltammetryElement.h</a>	270
<a href="#">AisEISGalvanostaticElement.h</a>	271
<a href="#">AisEISPotentiostaticElement.h</a>	272
<a href="#">AisMottSchottkyElement.h</a>	272
<a href="#">AisNormalPulseVoltammetryElement.h</a>	273
<a href="#">AisOpenCircuitElement.h</a>	274
<a href="#">AisSquareWaveVoltammetryElement.h</a>	275
<a href="#">AisStaircasePotentialVoltammetryElement.h</a>	276
<a href="#">AisSteppedCurrentElement.h</a>	277
<a href="#">AisSteppedVoltageElement.h</a>	278



# Chapter 15

## Class Documentation

### 15.1 AisACData Struct Reference

a structure containing AC data information.

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

#### Public Attributes

- double **timestamp**  
*the time at which the AC data arrived.*
- double **frequency**  
*the applied frequency in Hz.*
- double **absoluteImpedance**  
*the magnitude of the complex impedance.*
- double **realImpedance**  
*the real part of the complex impedance.*
- double **imagImpedance**  
*the imaginary part of the complex impedance.*
- double **phaseAngle**  
*the phase angle between the real and the imaginary parts of the impedance.*
- double **totalHarmonicDistortion**  
*the percentage of the total harmonic distortion in the AC signal.*
- double **numberOfCycles**  
*the number of cycles specific to the reported frequency.*
- double **workingElectrodeDCVoltage**  
*the DC working electrode voltage in volts.*
- double **DCCurrent**  
*the DC electric current value in Amps*
- double **currentAmplitude**  
*the amplitude of the AC current.*
- double **voltageAmplitude**  
*the amplitude of the AC voltage.*

### 15.1.1 Detailed Description

a structure containing AC data information.

### 15.1.2 Member Data Documentation

#### 15.1.2.1 numberOfCycles

```
double AisACData::numberOfCycles
```

the number of cycles specific to the reported frequency.

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

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

- AisDataPoints.h

## 15.2 AisCompRange Class Reference

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

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

### Public Member Functions

- [AisCompRange](#) (const QString &compRangeName, uint8\_t bandwidthIndex, uint8\_t stabilityFactor)  
*constructor for the compensation-range object.*
- [AisCompRange](#) (const [AisCompRange](#) &)  
*copy constructor for the compensation-range object.*
- uint8\_t [getBandwidthIndex](#) () const  
*get the value set for the bandwidth index.*
- void [setBandwidthIndex](#) (uint8\_t index)  
*set the index value for the bandwidth.*
- uint8\_t [getStabilityFactor](#) () const  
*get the value set for the stability factor.*
- void [setStabilityFactor](#) (uint8\_t factor)  
*set a value for the stability factor.*
- void [setCompRangeName](#) (const QString &compRangeName)  
*set a name for the compensation range for reference purposes.*
- const QString & [getCompRangeName](#) () const  
*get the name set for the compensation range.*

### 15.2.1 Detailed Description

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

See also

[setBandwidthIndex](#)

[setStabilityFactor](#)

### 15.2.2 Constructor & Destructor Documentation

#### 15.2.2.1 AisCompRange()

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

constructor for the compensation-range object.

Parameters

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

See also

[setBandwidthIndex](#)

[setStabilityFactor](#)

### 15.2.3 Member Function Documentation

#### 15.2.3.1 getBandwidthIndex()

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

get the value set for the bandwidth index.

Returns

the set value for the bandwidth index.

See also

[setBandwidthIndex](#)

### 15.2.3.2 getCompRangeName()

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

get the name set for the compensation range.

#### Returns

the name set for the compensation range.

### 15.2.3.3 getStabilityFactor()

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

get the value set for the stability factor.

#### Returns

the value set for the stability factor.

### 15.2.3.4 setBandwidthIndex()

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

set the index value for the bandwidth.

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

#### Parameters

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

### 15.2.3.5 setCompRangeName()

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

set a name for the compensation range for reference purposes.



## Parameters

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

**15.2.3.6 setStabilityFactor()**

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

set a value for the stability factor.

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

## Parameters

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

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

- AisCompRange.h

**15.3 AisConstantCurrentElement Class Reference**

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

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

Inherits AisAbstractElement.

**Public Member Functions**

- [AisConstantCurrentElement](#) (double current, double samplingInterval, double duration)  
*the constant current element constructor.*
- [AisConstantCurrentElement](#) (const [AisConstantCurrentElement](#) &)  
*copy constructor for the [AisConstantCurrentElement](#) object.*
- [AisConstantCurrentElement](#) & **operator=** (const [AisConstantCurrentElement](#) &)  
*overload equal to operator for the [AisConstantCurrentElement](#) object.*
- QString [getName](#) () const override  
*get the name of the element.*
- QStringList [getCategory](#) () const override  
*get a list of applicable categories of the element.*
- double [getCurrent](#) () const  
*get the value set for the current.*

- void [setCurrent](#) (double current)  
*set the value for the current.*
- double [getSamplingInterval](#) () const  
*get how frequently we are sampling the data.*
- void [setSamplingInterval](#) (double samplingInterval)  
*set how frequently we are sampling the data.*
- double [getMinSamplingVoltageDifference](#) () const  
*get the minimum sampling voltage difference for reporting the data.*
- void [setMinSamplingVoltageDifference](#) (double minVoltageDifference)  
*set a minimum sampling voltage difference for reporting the voltage.*
- double [getMaxVoltage](#) () const  
*get the value set for the maximum voltage. The experiment will end when it reaches this value.*
- void [setMaxVoltage](#) (double maxVoltage)  
*set a maximum voltage to stop the experiment.*
- double [getMinVoltage](#) () const  
*get the value set minimum for the voltage in volts.*
- void [setMinVoltage](#) (double minVoltage)  
*set a minimum voltage to stop the experiment.*
- double [getMaxDuration](#) () const  
*get the maximum duration set for the experiment. The experiment will end when the duration of the experiment reaches this value.*
- void [setMaxDuration](#) (double maxDuration)  
*set the maximum duration for the experiment.*
- double [getMaxCapacity](#) () const  
*get the value set for the maximum capacity / cumulative charge.*
- void [setMaxCapacity](#) (double maxCapacity)  
*set the value for the maximum capacity / cumulative charge in Coulomb.*
- bool [isAutoRange](#) () const  
*tells whether the current range is set to auto-select or not.*
- void [setAutoRange](#) ()  
*set to auto-select the current range.*
- double [getApproxMaxCurrent](#) () const  
*get the value set for the expected maximum current.*
- void [setApproxMaxCurrent](#) (double approxMaxCurrent)  
*set maximum current expected, for manual current range selection.*
- bool [isAutoVoltageRange](#) () const  
*tells whether the voltage range is set to auto-select or not.*
- void [setAutoVoltageRange](#) ()  
*set to auto-select the voltage range.*
- double [getApproxMaxVoltage](#) () const  
*get the value set for the expected maximum voltage.*
- void [setApproxMaxVoltage](#) (double approxMaxVoltage)  
*set maximum voltage expected, for manual voltage range selection.*

### 15.3.1 Detailed Description

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

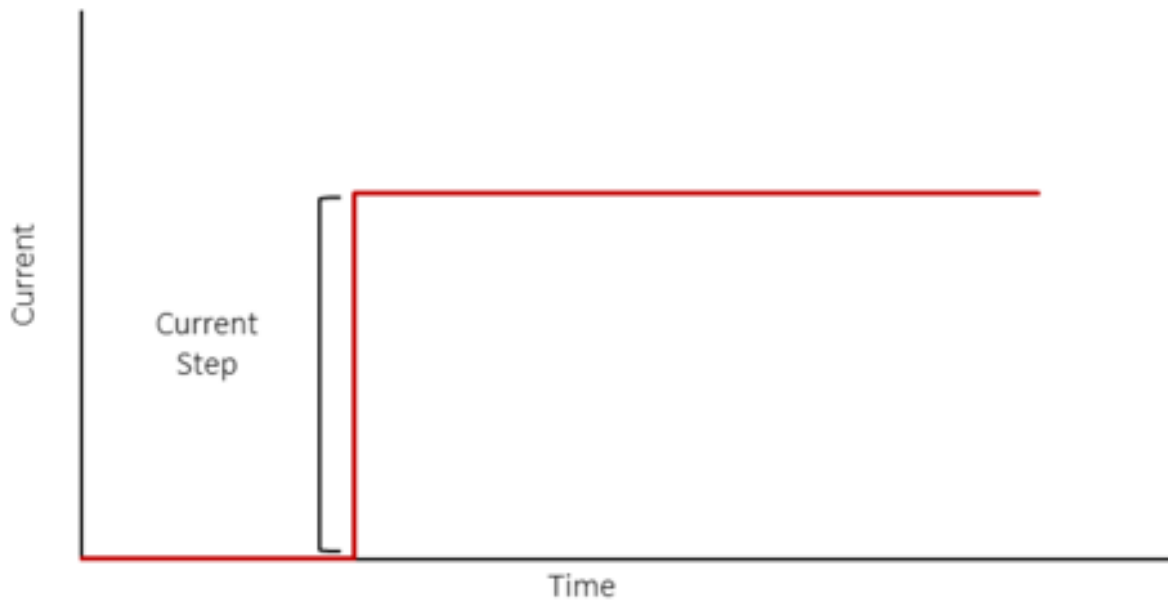


Figure 15.1 ConstantCurrent

### 15.3.2 Constructor & Destructor Documentation

#### 15.3.2.1 AisConstantCurrentElement()

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

the constant current element constructor.

#### Parameters

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

### 15.3.3 Member Function Documentation

#### 15.3.3.1 getApproxMaxCurrent()

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

get the value set for the expected maximum current.

##### Returns

the value set for the expected maximum current in Amps.

##### Note

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

#### 15.3.3.2 getApproxMaxVoltage()

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

get the value set for the expected maximum voltage.

##### Returns

the value set for the expected maximum Voltage in volt.

##### Note

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

#### 15.3.3.3 getCategory()

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

get a list of applicable categories of the element.

##### Returns

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

#### 15.3.3.4 `getCurrent()`

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

get the value set for the current.

##### Returns

the value for the current in Amps.

#### 15.3.3.5 `getMaxCapacity()`

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

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

##### Returns

the value set for the maximum capacity in Coulomb.

##### Note

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

#### 15.3.3.6 `getMaxDuration()`

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

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

##### Returns

the maximum duration for the experiment in seconds.

#### 15.3.3.7 `getMaxVoltage()`

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

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

##### Returns

the value set for the maximum voltage.

##### Note

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

#### 15.3.3.8 getMinSamplingVoltageDifference()

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

get the minimum sampling voltage difference for reporting the data.

get the value set for the minimum sampling voltage difference.

##### Returns

the value set for the minimum sampling voltage difference.

##### See also

[setMinSamplingVoltageDifference](#)

##### Note

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

#### 15.3.3.9 getMinVoltage()

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

get the value set minimum for the voltage in volts.

##### Returns

the value set for the minimum voltage in volts.

##### Note

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

#### 15.3.3.10 getName()

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

get the name of the element.

##### Returns

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

#### 15.3.3.11 getSamplingInterval()

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

get how frequently we are sampling the data.

##### Returns

the data sampling interval value in seconds.

#### 15.3.3.12 isAutoRange()

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

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

##### Returns

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

#### 15.3.3.13 isAutoVoltageRange()

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

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

##### Returns

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

#### 15.3.3.14 setApproxMaxCurrent()

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

set maximum current expected, for manual current range selection.

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

##### Parameters

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

### 15.3.3.15 setApproxMaxVoltage()

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

set maximum voltage expected, for manual voltage range selection.

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

#### Parameters

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

### 15.3.3.16 setAutoRange()

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

set to auto-select the current range.

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

### 15.3.3.17 setAutoVoltageRange()

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

set to auto-select the voltage range.

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

### 15.3.3.18 setCurrent()

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

set the value for the current.

#### Parameters

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



### 15.3.3.19 setMaxCapacity()

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

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

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

#### Parameters

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

### 15.3.3.20 setMaxDuration()

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

set the maximum duration for the experiment.

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

#### Parameters

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

### 15.3.3.21 setMaxVoltage()

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

set a maximum voltage to stop the experiment.

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

#### Parameters

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

### 15.3.3.22 setMinSamplingVoltageDifference()

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

set a minimum sampling voltage difference for reporting the voltage.

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

#### Note

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

#### Parameters

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

### 15.3.3.23 setMinVoltage()

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

set a minimum voltage to stop the experiment.

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

#### Parameters

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

### 15.3.3.24 setSamplingInterval()

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

set how frequently we are sampling the data.

#### Parameters

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

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

- AisConstantCurrentElement.h

## 15.4 AisConstantPotElement Class Reference

an experiment that simulates a constant applied voltage.

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

Inherits AisAbstractElement.

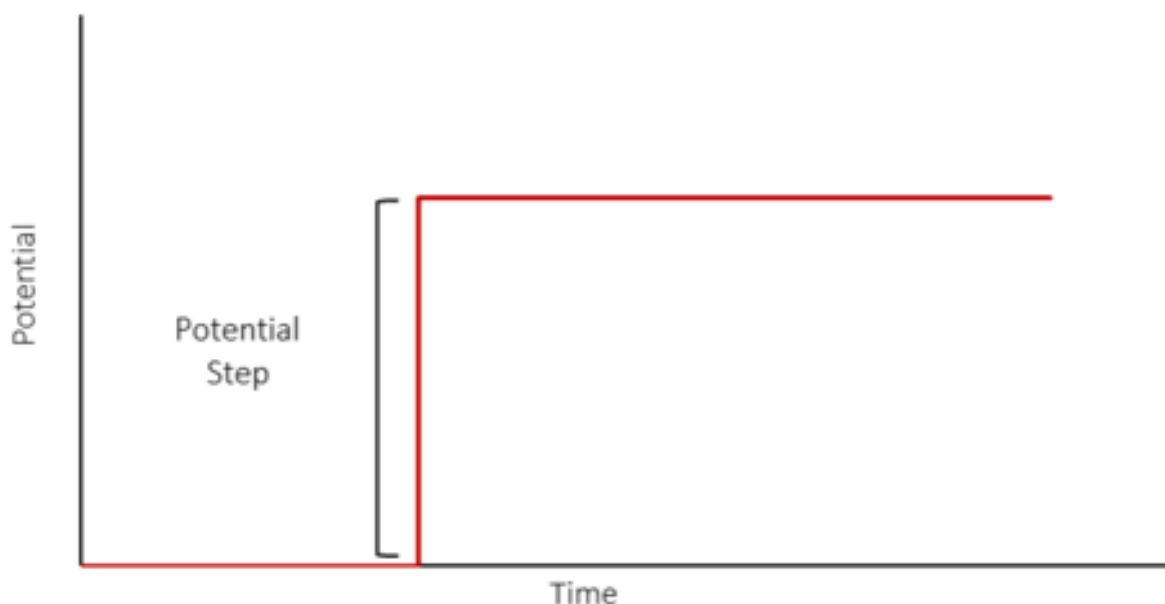
### Public Member Functions

- [AisConstantPotElement](#) (double voltage, double samplingInterval, double duration)  
*the constant potential element constructor.*
- [AisConstantPotElement](#) (const [AisConstantPotElement](#) &)  
*copy constructor for the [AisConstantPotElement](#) object.*
- [AisConstantPotElement](#) & **operator=** (const [AisConstantPotElement](#) &)  
*overload equal to operator for the [AisConstantPotElement](#) object.*
- QString [getName](#) () const override  
*get the name of the element.*
- QStringList [getCategory](#) () const override  
*get a list of applicable categories of the element.*
- double [getPotential](#) () const  
*get the value set for the potential in volts.*
- void [setPotential](#) (double potential)  
*set the value for the potential in volts.*
- bool [isVoltageVsOCP](#) () const  
*tells whether the specified voltage is set against the open-circuit voltage or the reference terminal.*
- void [setVoltageVsOCP](#) (bool vsOCP)  
*set whether to reference the specified voltage against the open-circuit voltage or the reference terminal.*
- double [getSamplingInterval](#) () const  
*get how frequently we are sampling the data.*
- void [setSamplingInterval](#) (double samplingInterval)  
*set how frequently we are sampling the data.*
- double [getMaxDuration](#) () const  
*get the maximum duration set for the experiment. The experiment will end when the duration of the experiment reaches this value.*
- void [setMaxDuration](#) (double maxDuration)  
*set the maximum duration for the experiment.*
- double [getMaxCurrent](#) () const  
*get the maximum value set for the absolute current in Amps. The experiment will end when the absolute current reaches this value.*
- void [setMaxCurrent](#) (double maxCurrent)  
*set the maximum value for the absolute current in Amps.*
- double [getMinCurrent](#) () const

- get the minimum value set for the absolute current in Amps. The experiment will end when the absolute current falls down to this value.*
- void `setMinCurrent` (double minCurrent)  
*set the minimum value for the absolute current in Amps.*
  - double `getMaxCapacity` () const  
*get the value set for the maximum capacity / cumulative charge.*
  - void `setMaxCapacity` (double maxCapacity)  
*set the value for the maximum capacity / cumulative charge in Coulomb.*
  - double `getMindIdt` () const  
*get the value set for the minimum current rate of change with respect to time (minimum di/dt).*
  - void `setMindIdt` (double mindIdt)  
*set the minimum value for the current rate of change with respect to time (minimum di/dt).*
  - bool `isAutoRange` () const  
*tells whether the current range is set to auto-select or not.*
  - void `setAutoRange` ()  
*set to auto-select the current range.*
  - double `getApproxMaxCurrent` () const  
*get the value set for the expected maximum current.*
  - void `setApproxMaxCurrent` (double approxMaxCurrent)  
*set maximum current expected, for manual current range selection.*
  - void `setVoltageRange` (int idx)  
*manually set the voltage control range.*

### 15.4.1 Detailed Description

an experiment that simulates a constant applied voltage.



### 15.4.2 Constructor & Destructor Documentation

### 15.4.2.1 AisConstantPotElement()

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

the constant potential element constructor.

#### Parameters

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

## 15.4.3 Member Function Documentation

### 15.4.3.1 getApproxMaxCurrent()

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

get the value set for the expected maximum current.

#### Returns

the value set for the expected maximum current in Amps.

#### Note

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

### 15.4.3.2 getCategory()

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

get a list of applicable categories of the element.

#### Returns

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

#### 15.4.3.3 getMaxCapacity()

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

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

##### Returns

the value set for the maximum capacity in Coulomb.

##### Note

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

#### 15.4.3.4 getMaxCurrent()

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

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

##### Returns

the maximum current value in Amps.

##### Note

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

#### 15.4.3.5 getMaxDuration()

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

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

##### Returns

the maximum duration for the experiment in seconds.

### 15.4.3.6 getMinCurrent()

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

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

#### Returns

the minimum current value in Amps.

#### Note

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

### 15.4.3.7 getMindIdt()

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

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

#### Returns

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

#### Note

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

### 15.4.3.8 getName()

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

get the name of the element.

#### Returns

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

#### 15.4.3.9 getPotential()

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

get the value set for the potential in volts.

##### Returns

the value set for the potential in volts.

#### 15.4.3.10 getSamplingInterval()

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

get how frequently we are sampling the data.

##### Returns

the data sampling interval value in seconds.

#### 15.4.3.11 isAutoRange()

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

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

##### Returns

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

#### 15.4.3.12 isVoltageVsOCP()

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

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

##### Returns

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

##### See also

setVsOcp

#### 15.4.3.13 setApproxMaxCurrent()

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

set maximum current expected, for manual current range selection.

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



## Parameters

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

**15.4.3.14 setAutoRange()**

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

set to auto-select the current range.

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

**15.4.3.15 setMaxCapacity()**

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

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

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

## Parameters

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

**15.4.3.16 setMaxCurrent()**

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

set the maximum value for the absolute current in Amps.

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

## Parameters

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

#### 15.4.3.17 setMaxDuration()

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

set the maximum duration for the experiment.

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

##### Parameters

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

#### 15.4.3.18 setMinCurrent()

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

set the minimum value for the absolute current in Amps.

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

##### Parameters

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

#### 15.4.3.19 setMindIdt()

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

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

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

##### Parameters

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

### 15.4.3.20 setPotential()

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

set the value for the potential in volts.

#### Parameters

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

### 15.4.3.21 setSamplingInterval()

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

set how frequently we are sampling the data.

#### Parameters

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

### 15.4.3.22 setVoltageRange()

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

manually set the voltage control range.

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

#### Parameters

<i>idx</i>	the corresponding voltage range index (see <a href="#">AisInstrumentHandler::getManualModeVoltageRangeList()</a> )
------------	--

### 15.4.3.23 setVoltageVsOCP()

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

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

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

## Parameters

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

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

- `AisConstantPotElement.h`

## 15.5 AisConstantPowerElement Class Reference

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

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

Inherits `AisAbstractElement`.

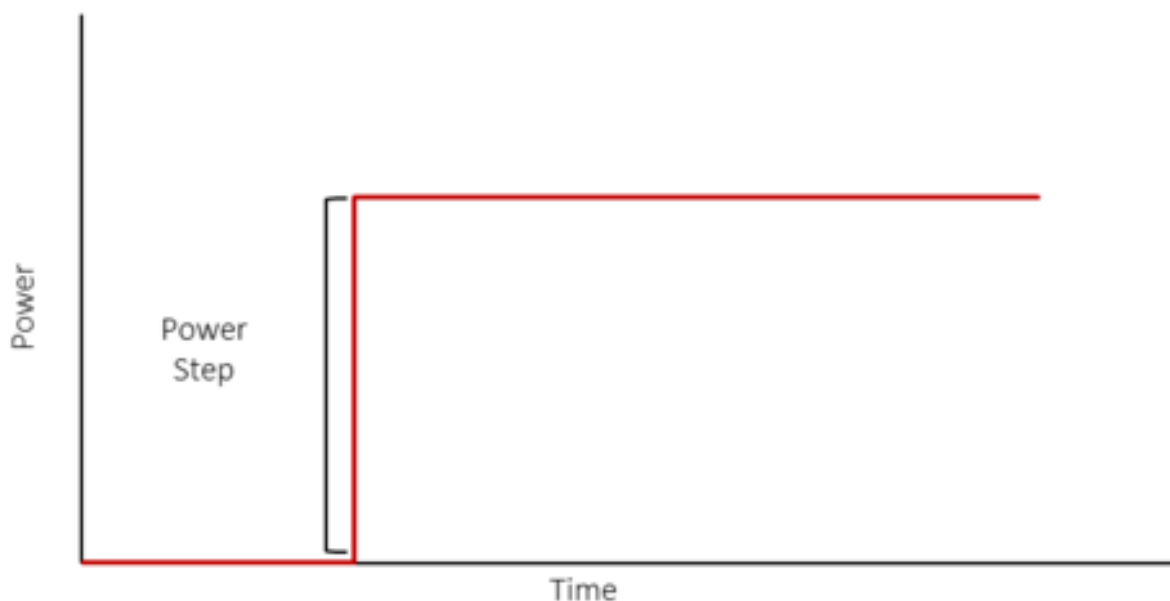
### Public Member Functions

- `AisConstantPowerElement` (bool `isCharge`, double power, double duration, double samplingInterval)  
*the constant power element constructor*
- `AisConstantPowerElement` (const `AisConstantPowerElement` &)  
*copy constructor for the `AisConstantPowerElement` object.*
- `AisConstantPowerElement` & `operator=` (const `AisConstantPowerElement` &)  
*overload equal to operator for the `AisConstantPowerElement` object.*
- `QString getName ()` const override  
*get the name of the element.*
- `QStringList getCategory ()` const override  
*get a list of applicable categories of the element.*
- bool `isCharge ()` const  
*tells whether the experiment is set to simulate charge or discharge.*
- void `setCharge` (bool `isCharge`)  
*set whether the experiment is to simulate charge or discharge.*
- double `getPower ()` const  
*get the value set for the power.*
- void `setPower` (double power)  
*set the value for the power.*
- double `getSamplingInterval ()` const  
*get how frequently we are sampling the data.*
- void `setSamplingInterval` (double samplingInterval)  
*set how frequently we are sampling the data.*
- double `getMaxVoltage ()` const  
*get the value set for the maximum voltage. The experiment will end when it reaches this value.*
- void `setMaxVoltage` (double maxVoltage)  
*set a maximum voltage to stop the experiment.*
- 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.*

### 15.5.1 Detailed Description

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



## 15.5.2 Constructor & Destructor Documentation

### 15.5.2.1 AisConstantPowerElement()

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

the constant power element constructor

#### Parameters

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

## 15.5.3 Member Function Documentation

### 15.5.3.1 getCategory()

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

get a list of applicable categories of the element.

#### Returns

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

### 15.5.3.2 getMaxCapacity()

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

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

#### Returns

the value set for the maximum capacity in Coulomb.

#### Note

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

### 15.5.3.3 getMaxCurrent()

```
double AisConstantPowerElement::getMaxCurrent ( ) const
```

get the value set maximum for the current in amps.

#### 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.

### 15.5.3.4 getMaxDuration()

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

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

#### Returns

the maximum duration for the experiment in seconds.

### 15.5.3.5 getMaxVoltage()

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

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

#### Returns

the value set for the maximum voltage.

#### Note

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

#### 15.5.3.6 getMinCurrent()

```
double AisConstantPowerElement::getMinCurrent ( ) const
```

get the value set minimum for the current in amps.

##### 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.

#### 15.5.3.7 getMinVoltage()

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

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

##### Returns

the minimum value set for the voltage in volts.

##### Note

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

#### 15.5.3.8 getName()

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

get the name of the element.

##### Returns

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



### 15.5.3.9 getPower()

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

get the value set for the power.

#### Returns

the value set for the power in watts.

### 15.5.3.10 getSamplingInterval()

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

get how frequently we are sampling the data.

#### Returns

the data sampling interval value in seconds.

### 15.5.3.11 isCharge()

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

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

#### Returns

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

### 15.5.3.12 isMaximumVoltageVsOCP()

```
bool AisConstantPowerElement::isMaximumVoltageVsOCP ( ) const
```

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

#### 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

### 15.5.3.13 isMinimumVoltageVsOCP()

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

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

#### 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

### 15.5.3.14 setCharge()

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

set whether the experiment is to simulate charge or discharge.

#### Parameters

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

### 15.5.3.15 setMaxCapacity()

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

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

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

#### Parameters

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

**15.5.3.16 setMaxCurrent()**

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

set a maximum current to stop the experiment.

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

**Parameters**

<i>maxCurrent</i>	the maximum current value in amps at which the experiment will stop.
-------------------	--

**15.5.3.17 setMaxDuration()**

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

set the maximum duration for the experiment.

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

**Parameters**

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

**15.5.3.18 setMaximumVoltageVsOCP()**

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

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

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

**Parameters**

<i>vsOCP</i>	true to set the specified maximum voltage to reference the open-circuit voltage and false to set against the reference terminal.
--------------	--

### 15.5.3.19 setMaxVoltage()

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

set a maximum voltage to stop the experiment.

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

#### Parameters

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

### 15.5.3.20 setMinCurrent()

```
void AisConstantPowerElement::setMinCurrent (
    double maxCurrent )
```

set a minimum current to stop the experiment.

The 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

<i>maxCurrent</i>	the minimum current value in amps at which the experiment will stop.
-------------------	--

### 15.5.3.21 setMinimumVoltageVsOCP()

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

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

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

#### Parameters

<i>vsOCP</i>	true to set the specified minimum voltage to reference the open-circuit voltage and false to set against the reference terminal.
--------------	--

### 15.5.3.22 setMinVoltage()

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

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

#### Parameters

<i>minVoltage</i>	the value for the voltage in volts.
-------------------	-------------------------------------

#### Note

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

### 15.5.3.23 setPower()

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

set the value for the power.

#### Parameters

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

### 15.5.3.24 setSamplingInterval()

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

set how frequently we are sampling the data.

#### Parameters

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

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

- AisConstantPowerElement.h

## 15.6 AisConstantResistanceElement Class Reference

This element/experiment simulates a constant resistance load.

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

Inherits AisAbstractElement.

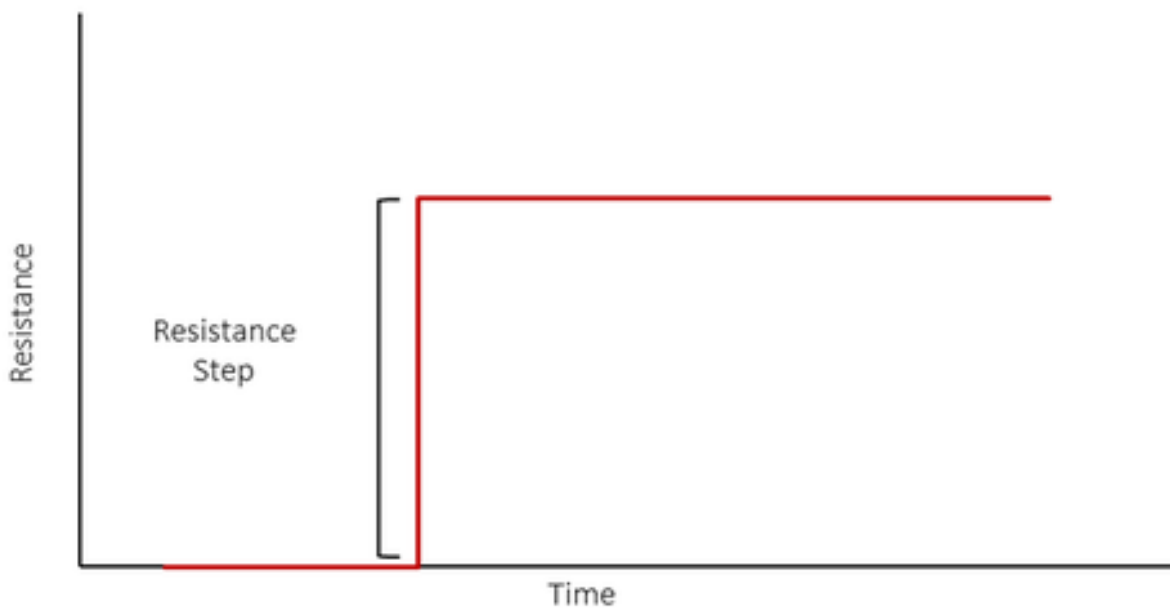
### Public Member Functions

- [AisConstantResistanceElement](#) (double resistance, double duration, double samplingInterval)  
*the constant resistance element constructor.*
- [AisConstantResistanceElement](#) (const [AisConstantResistanceElement](#) &)  
*copy constructor for the [AisConstantResistanceElement](#) object.*
- [AisConstantResistanceElement](#) & **operator=** (const [AisConstantResistanceElement](#) &)  
*overload equal to operator for the [AisConstantResistanceElement](#) object.*
- QString [getName](#) () const override  
*get the name of the element.*
- QStringList [getCategory](#) () const override  
*get a list of applicable categories of the element.*
- double [getResistance](#) () const  
*get the value set for the resistance as a load.*
- void [setResistance](#) (double resistance)  
*set the value for the resistance as a load*
- double [getSamplingInterval](#) () const  
*get how frequently we are sampling the data.*
- void [setSamplingInterval](#) (double samplingInterval)  
*set how frequently we are sampling the data.*
- double [getMinVoltage](#) () const  
*get the value set minimum for the voltage in volts.*
- void [setMinVoltage](#) (double minVoltage)  
*set a minimum voltage to stop the experiment.*
- 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.*

### 15.6.1 Detailed Description

This element/experiment simulates a constant resistance load.



### 15.6.2 Constructor & Destructor Documentation

#### 15.6.2.1 AisConstantResistanceElement()

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

the constant resistance element constructor.

**Parameters**

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

### 15.6.3 Member Function Documentation

#### 15.6.3.1 getCategory()

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

get a list of applicable categories of the element.

**Returns**

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

#### 15.6.3.2 getMaxCapacity()

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

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

**Returns**

the value set for the maximum capacity in Coulomb.

**Note**

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

#### 15.6.3.3 getMaxCurrent()

```
double AisConstantResistanceElement::getMaxCurrent ( ) const
```

get the value set maximum for the current in amps.

**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.



#### 15.6.3.4 getMaxDuration()

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

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

##### Returns

the maximum duration for the experiment in seconds.

#### 15.6.3.5 getMaxVoltage()

```
double AisConstantResistanceElement::getMaxVoltage ( ) const
```

get the value set maximum for the voltage in volts.

##### 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

#### 15.6.3.6 getMinCurrent()

```
double AisConstantResistanceElement::getMinCurrent ( ) const
```

get the value set minimum for the current in amps.

##### 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.

### 15.6.3.7 getMinVoltage()

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

get the value set minimum for the voltage in volts.

#### Returns

the value set for the minimum voltage in volts.

#### Note

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

### 15.6.3.8 getName()

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

get the name of the element.

#### Returns

The name of the element: "Constant Resistance".

### 15.6.3.9 getResistance()

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

get the value set for the resistance as a load.

#### Returns

the value in ohm of the load resistance.

### 15.6.3.10 getSamplingInterval()

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

get how frequently we are sampling the data.

#### Returns

the data sampling interval value in seconds.

### 15.6.3.11 isMaximumVoltageVsOCP()

```
bool AisConstantResistanceElement::isMaximumVoltageVsOCP ( ) const
```

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

#### 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

### 15.6.3.12 isMinimumVoltageVsOCP()

```
bool AisConstantResistanceElement::isMinimumVoltageVsOCP ( ) const
```

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

#### 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

### 15.6.3.13 setMaxCapacity()

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

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

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

#### Parameters

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

### 15.6.3.14 setMaxCurrent()

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

set a maximum current to stop the experiment.

There 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 above that value.

#### Parameters

<i>maxCurrent</i>	the maximum current value in amps at which the experiment will stop.
-------------------	--

### 15.6.3.15 setMaxDuration()

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

set the maximum duration for the experiment.

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

#### Parameters

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

### 15.6.3.16 setMaximumVoltageVsOCP()

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

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

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

#### Parameters

<i>vsOCP</i>	true to set the specified maximum voltage to reference the open-circuit voltage and false to set against the reference terminal.
--------------	--

**15.6.3.17 setMaxVoltage()**

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

set a maximum voltage to stop the experiment.

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

**Parameters**

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

**15.6.3.18 setMinCurrent()**

```
void AisConstantResistanceElement::setMinCurrent (
    double maxCurrent )
```

set a minimum current to stop the experiment.

The 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**

<i>maxCurrent</i>	the minimum current value in amps at which the experiment will stop.
-------------------	--

**15.6.3.19 setMinimumVoltageVsOCP()**

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

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

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

**Parameters**

<i>vsOCP</i>	true to set the specified minimum voltage to reference the open-circuit voltage and false to set against the reference terminal.
--------------	--

### 15.6.3.20 setMinVoltage()

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

set a minimum voltage to stop the experiment.

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

#### Parameters

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

### 15.6.3.21 setResistance()

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

set the value for the resistance as a load

#### Parameters

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

### 15.6.3.22 setSamplingInterval()

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

set how frequently we are sampling the data.

#### Parameters

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

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

- AisConstantResistanceElement.h

## 15.7 AisCyclicVoltammetryElement Class Reference

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

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

Inherits AisAbstractElement.

### Public Member Functions

- [AisCyclicVoltammetryElement](#) (double startVoltage, double firstVoltageLimit, double secondVoltageLimit, double endVoltage, double dEdt, double samplingInterval)  
*constructor of the cyclic voltammetry element.*
- [AisCyclicVoltammetryElement](#) (const [AisCyclicVoltammetryElement](#) &)  
*copy constructor for the [AisCyclicVoltammetryElement](#) object.*
- [AisCyclicVoltammetryElement](#) & **operator=** (const [AisCyclicVoltammetryElement](#) &)  
*overload equal to operator for the [AisCyclicVoltammetryElement](#) object.*
- QString [getName](#) () const override  
*get the name of the element.*
- QStringList [getCategory](#) () const override  
*get a list of applicable categories of the element.*
- double [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.*
- double `getNumberOfCycles` ()  
*get the value set for the number of cycles*
- void `setNumberOfCycles` (int cycles)  
*set the number of cycles to oscillate between the first voltage-limit and the second voltage-limit.*
- double `getEndVoltage` () const  
*get the value set for the ending potential value.*
- void `setEndVoltage` (double endVoltage)  
*set the ending potential value.*
- bool `isEndVoltageVsOCP` () const  
*tells whether the end voltage is set with respect to the open circuit voltage or the reference terminal.*
- void `setEndVoltageVsOCP` (bool endVoltageVsOCP)  
*set whether to reference the end voltage against the open-circuit voltage or the reference terminal.*
- double `getdEdt` () const  
*get the value set for the constant scan rate  $dE/dt$ .*
- void `setdEdt` (double dEdt)  
*set the value for the constant scan rate  $dE/dt$ .*
- double `getSamplingInterval` () const  
*get how frequently we are sampling the data.*
- void `setSamplingInterval` (double samplInterval)  
*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` ()  
*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.*

### 15.7.1 Detailed Description

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

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



## 15.7.2 Constructor & Destructor Documentation

### 15.7.2.1 AisCyclicVoltammetryElement()

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

constructor of the cyclic voltammetry element.

#### Parameters

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

## 15.7.3 Member Function Documentation

### 15.7.3.1 getAlphaFactor()

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

Get the value set for the alpha factor.

#### 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.

### 15.7.3.2 getApproxMaxCurrent()

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

get the value set for the expected maximum current.

#### Returns

the value set for the expected maximum current in Amps.

#### Note

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

### 15.7.3.3 getCategory()

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

get a list of applicable categories of the element.

#### Returns

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

### 15.7.3.4 getdEdt()

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

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

#### Returns

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

### 15.7.3.5 getEndVoltage()

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

get the value set for the ending potential value.

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

#### Returns

the value set for the ending voltage in volts.

### 15.7.3.6 getFirstVoltageLimit()

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

get the value set for the first voltage-limit.

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

#### Returns

the first voltage-limit value in volts.

### 15.7.3.7 getName()

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

get the name of the element.

#### Returns

The name of the element: "Cyclic Voltammetry".

### 15.7.3.8 getNumberOfCycles()

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

get the value set for the number of cycles

#### Returns

the number of cycles set.

### 15.7.3.9 getQuietTime()

```
double AisCyclicVoltammetryElement::getQuietTime ( ) const
```

Gets the quiet time duration.

#### Returns

The quiet time duration in seconds.

#### 15.7.3.10 getQuietTimeSamplingInterval()

```
double AisCyclicVoltammetryElement::getQuietTimeSamplingInterval ( ) const
```

gets the potential sampling interval.

##### Returns

samplingInterval The quiet time sampling interval to set in seconds.

#### 15.7.3.11 getSamplingInterval()

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

get how frequently we are sampling the data.

##### Returns

the data sampling interval value in seconds.

#### 15.7.3.12 getSecondVoltageLimit()

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

get the value set for the second voltage-limit

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

##### Returns

the second voltage-limit value in volts.

#### 15.7.3.13 getStartVoltage()

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

get the value set for the start voltage

##### Returns

the value of the start voltage in volts

#### 15.7.3.14 isAutoRange()

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

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

##### Returns

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

#### 15.7.3.15 isEndVoltageVsOCP()

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

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

##### Returns

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

##### Note

if no value was set, the default is false

#### 15.7.3.16 isFirstVoltageLimitVsOCP()

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

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

##### Returns

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

##### Note

if no value was set, the default is false.

### 15.7.3.17 isSecondVoltageLimitVsOCP()

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

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

#### Returns

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

#### Note

if no value was set, the default is false.

### 15.7.3.18 isStartVoltageVsOCP()

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

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

#### Returns

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

### 15.7.3.19 setAlphaFactor()

```
void AisCyclicVoltammetryElement::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.

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

#### Parameters

<i>alphaFactor</i>	the value for the alphaFactor ranges from 0 to 100.
--------------------	---

### 15.7.3.20 setApproxMaxCurrent()

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

set maximum current expected, for manual current range selection.

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

#### Parameters

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

### 15.7.3.21 setAutoRange()

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

set to auto-select the current range.

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

### 15.7.3.22 setdEdt()

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

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

#### Parameters

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

### 15.7.3.23 setEndVoltage()

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

set the ending potential value.

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

#### Parameters

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

### 15.7.3.24 setEndVoltageVsOCP()

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

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

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

#### Parameters

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

### 15.7.3.25 setFirstVoltageLimit()

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

set the first voltage-limit

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

#### Parameters

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

### 15.7.3.26 setFirstVoltageLimitVsOCP()

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

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

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

#### Parameters

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



### 15.7.3.27 setNumberOfCycles()

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

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

#### Parameters

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

### 15.7.3.28 setQuietTime()

```
void AisCyclicVoltammetryElement::setQuietTime (
    double quietTime )
```

Sets the quiet time duration.

#### Parameters

<i>quietTime</i>	The quiet time duration to set in seconds.
------------------	--

### 15.7.3.29 setQuietTimeSamplingInterval()

```
void AisCyclicVoltammetryElement::setQuietTimeSamplingInterval (
    double quietTimeSamplingInterval )
```

Sets the quiet time sampling interval.

#### Parameters

<i>quietTimeSamplingInterval</i>	The quiet time sampling interval to set in seconds.
----------------------------------	---

### 15.7.3.30 setSamplingInterval()

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

set how frequently we are sampling the data.

## Parameters

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

**15.7.3.31 setSecondVoltageLimit()**

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

set the second voltage-limit

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

## Parameters

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

**15.7.3.32 setSecondVoltageLimitVsOCP()**

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

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

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

## Parameters

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

**15.7.3.33 setStartVoltage()**

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

set the value for the start voltage.

## Parameters

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

**15.7.3.34 setStartVoltageVsOCP()**

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

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

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

## Parameters

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

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

- AisCyclicVoltammetryElement.h

**15.8 AisDataManipulator Class Reference**

The [AisDataManipulator](#) 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>
```

**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.*

### 15.8.1 Detailed Description

The [AisDataManipulator](#) 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).

The Pulse Voltammetry example shows how to use this class with each pulse element.

### 15.8.2 Member Function Documentation

#### 15.8.2.1 [getBaseCurrent\(\)](#)

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

Get the base current.

##### Returns

The base current.

#### 15.8.2.2 [getBaseVoltage\(\)](#)

```
double AisDataManipulator::getBaseVoltage ( ) const
```

Get the base voltage.

##### Returns

The base voltage.

### 15.8.2.3 getFrequency()

```
double AisDataManipulator::getFrequency ( ) const
```

Get the pulse frequency.

#### Returns

The pulse frequency.

### 15.8.2.4 getPulseCurrent()

```
double AisDataManipulator::getPulseCurrent ( ) const
```

Get the pulse current.

#### Returns

The pulse current.

### 15.8.2.5 getPulsePeriod()

```
double AisDataManipulator::getPulsePeriod ( ) const
```

Get the pulse period.

#### Returns

The pulse period.

### 15.8.2.6 getPulseVoltage()

```
double AisDataManipulator::getPulseVoltage ( ) const
```

Get the pulse voltage.

#### Returns

The pulse voltage.

### 15.8.2.7 getPulseWidth()

```
double AisDataManipulator::getPulseWidth ( ) const
```

Get the pulse width.

#### Returns

The pulse width.

### 15.8.2.8 isPulseCompleted()

```
bool AisDataManipulator::isPulseCompleted ( ) const
```

Check if the pulse is completed.

#### Returns

True if the pulse is completed, false otherwise.

### 15.8.2.9 loadPrimaryData()

```
void AisDataManipulator::loadPrimaryData (
    const AisDCData & data )
```

Load primary data from [AisDCData](#) object.

#### Parameters

<i>data</i>	The <a href="#">AisDCData</a> object containing primary data.
-------------	---

### 15.8.2.10 setPulseType() [1/2]

```
AisErrorCode AisDataManipulator::setPulseType (
    AisPulseType type,
    double frequency )
```

Set pulse type with frequency.

#### Note

This function is usefull only for SquarewavePulse.

## Parameters

<i>type</i>	The type of pulse.
<i>frequency</i>	The frequency of the pulse.

## Returns

[AisErrorCode::Success](#) if pulse setting was successful. If not successful, possible returned errors are:

- [AisErrorCode::FailedRequest](#)

**15.8.2.11 setPulseType() [2/2]**

```
AisErrorCode AisDataManipulator::setPulseType (
    AisPulseType type,
    double pulseWidth,
    double pulsePeriod )
```

Set pulse type with pulse width and pulse period.

## Note

This function is usefull only for DifferentialPulse and NormalPulse.

## Parameters

<i>type</i>	The type of pulse.
<i>pulseWidth</i>	The width of the pulse.
<i>pulsePeriod</i>	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](#)

**15.9 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>
```

Inherits [AisAbstractElement](#).

## Public Member Functions

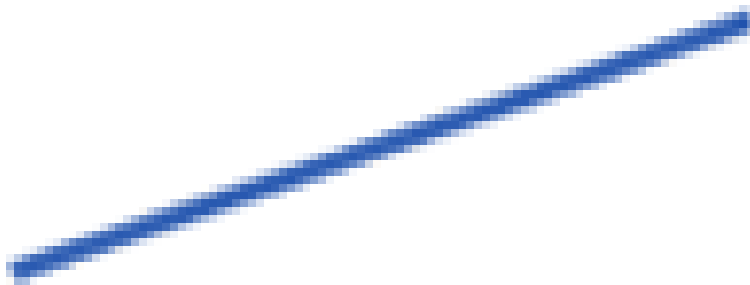
- [AisDCCurrentSweepElement](#) (double startCurrent, double endCurrent, double scanRate, double samplingInterval)
  - the DC current sweep element.*
- **AisDCCurrentSweepElement** (const [AisDCCurrentSweepElement](#) &)
  - copy constructor for the [AisDCCurrentSweepElement](#) object.*
- [AisDCCurrentSweepElement](#) & **operator=** (const [AisDCCurrentSweepElement](#) &)
  - overload equal to operator for the [AisDCCurrentSweepElement](#) object.*
- QString [getName](#) () const override
  - get the name of the element.*
- QStringList [getCategory](#) () const override
  - get a list of applicable categories of the element.*
- double [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](#) ()
  - 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.*



### 15.9.1 Detailed Description

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

# DC Current Linear Sweep



### 15.9.2 Constructor & Destructor Documentation

#### 15.9.2.1 AisDCCurrentSweepElement()

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

the DC current sweep element.

**Parameters**

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

**15.9.3 Member Function Documentation****15.9.3.1 getAlphaFactor()**

```
double AisDCCurrentSweepElement::getAlphaFactor ( )
```

Get the value set for the alpha factor.

**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.

**15.9.3.2 getCategory()**

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

get a list of applicable categories of the element.

**Returns**

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

**15.9.3.3 getEndingCurrent()**

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

get the value set for the ending current.

**Returns**

the value for the ending current in Amps.

#### 15.9.3.4 getMaxVoltage()

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

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

##### Returns

the value set for the maximum voltage.

##### Note

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

#### 15.9.3.5 getMinVoltage()

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

get the value set minimum for the voltage in volts.

##### Returns

the value set for the minimum voltage in volts.

##### Note

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

#### 15.9.3.6 getName()

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

get the name of the element.

##### Returns

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

### 15.9.3.7 getQuietTime()

```
double AisDCCurrentSweepElement::getQuietTime ( ) const
```

Gets the quiet time duration.

#### Returns

The quiet time duration in seconds.

### 15.9.3.8 getQuietTimeSamplingInterval()

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

gets the quiet time sampling interval.

#### Returns

samplingInterval The quiet time sampling interval to set in seconds.

### 15.9.3.9 getSamplingInterval()

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

get how frequently we are sampling the data.

#### Returns

the data sampling interval value in seconds.

### 15.9.3.10 getScanRate()

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

get the value set for the scan rate.

#### Returns

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

#### See also

[setScanRate](#)

### 15.9.3.11 getStartingCurrent()

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

get the value set for the starting current.

#### Returns

the value set for the constant current in Amps.

### 15.9.3.12 setAlphaFactor()

```
void AisDCCurrentSweepElement::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.

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

#### Parameters

<i>alphaFactor</i>	the value for the alphaFactor ranges from 0 to 100.
--------------------	---

### 15.9.3.13 setEndingCurrent()

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

set the value for the ending current.

#### Parameters

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

### 15.9.3.14 setMaxVoltage()

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

set a maximum voltage to stop the experiment.

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

## Parameters

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

**15.9.3.15 setMinVoltage()**

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

set a minimum voltage to stop the experiment.

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

## Parameters

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

**15.9.3.16 setQuietTime()**

```
void AisDCCurrentSweepElement::setQuietTime (
    double quietTime )
```

Sets the quiet time duration.

## Parameters

<i>quietTime</i>	The quiet time duration to set in seconds.
------------------	--

**15.9.3.17 setQuietTimeSamplingInterval()**

```
void AisDCCurrentSweepElement::setQuietTimeSamplingInterval (
    double quietTimeSamplingInterval )
```

Sets the quiet time sampling interval.

## Parameters

<i>quietTimeSamplingInterval</i>	The quiet time sampling interval to set in seconds.
----------------------------------	---

### 15.9.3.18 setSamplingInterval()

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

set how frequently we are sampling the data.

#### Parameters

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

### 15.9.3.19 setScanRate()

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

set the value for the current scan rate.

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

#### Parameters

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

### 15.9.3.20 setStartingCurrent()

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

set the value for the starting current.

#### Parameters

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

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

- AisDCCurrentSweepElement.h

## 15.10 AisDCData Struct Reference

a structure containing DC data information.



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

## Public Attributes

- double **timestamp**  
*the time at which the DC data arrived.*
- double **workingElectrodeVoltage**  
*the measured working electrode voltage in volts.*
- double **counterElectrodeVoltage**  
*the measured counter electrode voltage in volts.*
- double **current**  
*the measured electric current value in Amps*
- double **temperature**  
*the measured temperature in Celsius.*

### 15.10.1 Detailed Description

a structure containing DC data information.

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

- AisDataPoints.h

## 15.11 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>
```

Inherits AisAbstractElement.

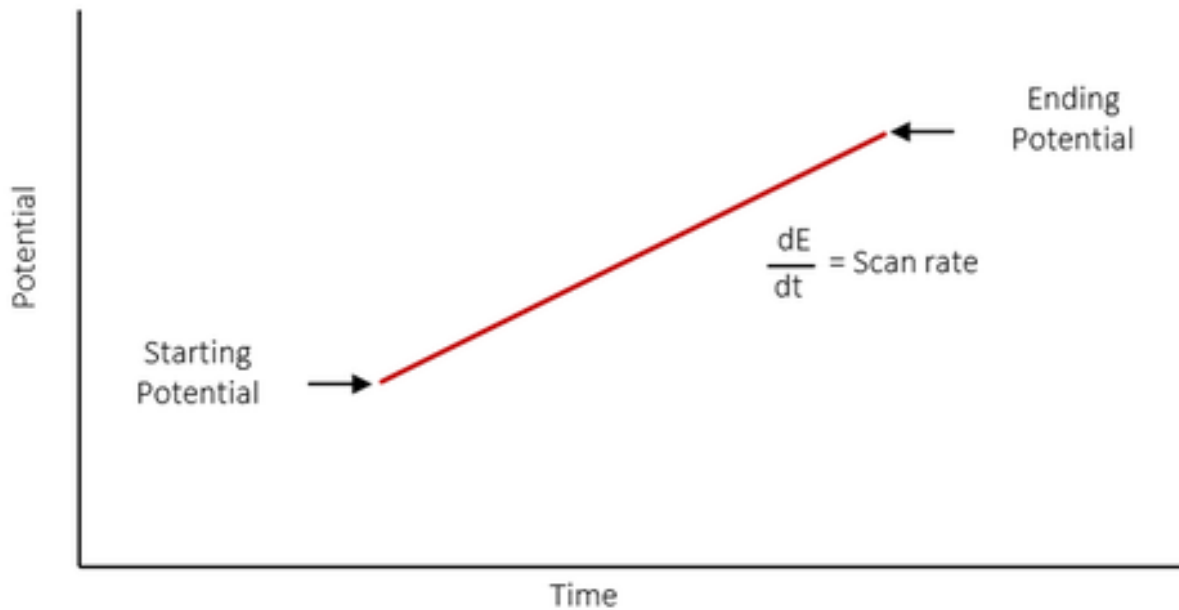
## Public Member Functions

- [AisDCPotentialSweepElement](#) (double startPotential, double endPotential, double scanRate, double samplingInterval)  
*the potential sweep element constructor.*
- **AisDCPotentialSweepElement** (const [AisDCPotentialSweepElement](#) &)  
*copy constructor for the [AisDCPotentialSweepElement](#) object.*
- [AisDCPotentialSweepElement](#) & **operator=** (const [AisDCPotentialSweepElement](#) &)  
*overload equal to operator for the [AisDCPotentialSweepElement](#) object.*
- QString [getName](#) () const override  
*get the name of the element.*
- QStringList [getCategory](#) () const override  
*get a list of applicable categories of the element.*
- double [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** ()  
*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.*

### 15.11.1 Detailed Description

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



### 15.11.2 Constructor & Destructor Documentation

#### 15.11.2.1 AisDCPotentialSweepElement()

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

the potential sweep element constructor.

#### Parameters

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

### 15.11.3 Member Function Documentation

#### 15.11.3.1 `getAlphaFactor()`

```
double AisDCPotentialSweepElement::getAlphaFactor ( )
```

Get the value set for the alpha factor.

##### 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.

#### 15.11.3.2 `getApproxMaxCurrent()`

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

get the value set for the expected maximum current.

##### Returns

the value set for the expected maximum current in Amps.

##### Note

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

#### 15.11.3.3 `getCategory()`

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

get a list of applicable categories of the element.

##### Returns

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

#### 15.11.3.4 getEndingPot()

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

get the value set for the ending potential value.

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

##### Returns

the value set for the ending voltage in volts.

#### 15.11.3.5 getMaxAbsoluteCurrent()

```
double AisDCPotentialSweepElement::getMaxAbsoluteCurrent ( ) const
```

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

##### 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

#### 15.11.3.6 getMinAbsoluteCurrent()

```
double AisDCPotentialSweepElement::getMinAbsoluteCurrent ( ) const
```

get the value set minimum for the Current in amps.

##### 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

### 15.11.3.7 getName()

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

get the name of the element.

#### Returns

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

### 15.11.3.8 getQuietTime()

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

Gets the quiet time duration.

#### Returns

The quiet time duration in seconds.

### 15.11.3.9 getQuietTimeSamplingInterval()

```
double AisDCPotentialSweepElement::getQuietTimeSamplingInterval ( ) const
```

gets the quiet time sampling interval.

#### Returns

samplingInterval The quiet time sampling interval to set in seconds.

### 15.11.3.10 getSamplingInterval()

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

get how frequently we are sampling the data.

#### Returns

the data sampling interval value in seconds.

#### 15.11.3.11 getScanRate()

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

get the value set for the voltage scan rate.

##### Returns

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

##### See also

[setScanRate](#)

#### 15.11.3.12 getStartingPot()

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

get the value set for the starting potential.

##### Returns

the value of the starting potential in volts.

#### 15.11.3.13 isAutoRange()

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

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

##### Returns

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

#### 15.11.3.14 isEndVoltageVsOCP()

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

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

##### Returns

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

##### See also

[setEndVoltageVsOCP](#)

### 15.11.3.15 isStartVoltageVsOCP()

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

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

#### Returns

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

#### See also

[setStartVoltageVsOCP](#)

### 15.11.3.16 setAlphaFactor()

```
void AisDCPotentialSweepElement::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.

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

#### Parameters

<i>alphaFactor</i>	the value for the alphaFactor ranges from 0 to 100.
--------------------	---

### 15.11.3.17 setApproxMaxCurrent()

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

set maximum current expected, for manual current range selection.

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

#### Parameters

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



**15.11.3.18 setAutoRange()**

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

set to auto-select the current range.

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

**15.11.3.19 setEndingPot()**

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

set the ending potential value.

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

**Parameters**

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

**15.11.3.20 setEndVoltageVsOCP()**

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

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

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

**Parameters**

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

**Note**

by default, this is set to false.

**15.11.3.21 setMaxAbsoluteCurrent()**

```
void AisDCPotentialSweepElement::setMaxAbsoluteCurrent (
    double maxCurrent )
```

set a maximum Current to stop the experiment.

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

#### Parameters

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

### 15.11.3.22 setMinAbsoluteCurrent()

```
void AisDCPotentialSweepElement::setMinAbsoluteCurrent (
    double minCurrent )
```

set a minimum Current to stop the experiment.

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

#### Parameters

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

### 15.11.3.23 setQuietTime()

```
void AisDCPotentialSweepElement::setQuietTime (
    double quietTime )
```

Sets the quiet time duration.

#### Parameters

<i>quietTime</i>	The quiet time duration to set in seconds.
------------------	--

### 15.11.3.24 setQuietTimeSamplingInterval()

```
void AisDCPotentialSweepElement::setQuietTimeSamplingInterval (
    double quietTimeSamplingInterval )
```

Sets the quiet time sampling interval.

## Parameters

<i>quietTimeSamplingInterval</i>	The quiet time sampling interval to set in seconds.
----------------------------------	---

**15.11.3.25 setSamplingInterval()**

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

set how frequently we are sampling the data.

## Parameters

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

**15.11.3.26 setScanRate()**

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

set the value for the voltage scan rate.

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

## Parameters

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

**15.11.3.27 setStartingPot()**

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

set the value for the starting potential.

## Parameters

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

### 15.11.3.28 setStartVoltageVsOCP()

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

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

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

#### Parameters

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

#### Note

by default, this is set to false.

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

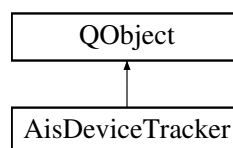
- AisDCPotentialSweepElement.h

## 15.12 AisDeviceTracker Class Reference

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

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

Inheritance diagram for AisDeviceTracker:



### Signals

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

## Public Member Functions

- [AisErrorCode](#) [connectToDeviceOnComPort](#) (const QString &comPort)  
*establish a connection with a device connected on a USB port.*
- const [AisInstrumentHandler](#) & [getInstrumentHandler](#) (const QString &deviceName) const  
*get an instrument handler to control a specific device.*
- const std::list< QString > [getConnectedDevices](#) () const  
*get a list of all the connected devices.*
- int [connectAllPluggedInDevices](#) ()  
*connect all devices physically plugged to the computer.*
- [AisErrorCode](#) [updateFirmwareOnComPort](#) (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](#) ()

### 15.12.1 Detailed Description

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

### 15.12.2 Member Function Documentation

#### 15.12.2.1 [connectAllPluggedInDevices\(\)](#)

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

connect all devices physically plugged to the computer.

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

#### Returns

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

#### Note

emits [newDeviceConnected\(\)](#) signal with the device name for each successful connection.

#### 15.12.2.2 connectToDeviceOnComPort()

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

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

## Parameters

<i>comPort</i>	the communication port to connect through.
----------------	--

## Returns

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

- [AisErrorCode::Unknown](#)
- [AisErrorCode::FirmwareNotSupported](#)
- [AisErrorCode::ConnectionFailed](#)

## Note

emits [newDeviceConnected\(\)](#) signal with the device name if establishing the connection was successful.

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

### 15.12.2.3 deviceDisconnected

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

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

## Parameters

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

### 15.12.2.4 getConnectedDevices()

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

get a list of all the connected devices.

## Returns

a list of all the connected devices.

### 15.12.2.5 getInstrumentHandler()

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

get an instrument handler to control a specific device.

**Parameters**

<i>deviceName</i>	the name of the connected device to get the instrument handler for.
-------------------	---

**Returns**

the instrument handler that controls the specified device.

**Note**

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

**See also**

[AisInstrumentHandler](#)  
[connectToDeviceOnComPort\(\)](#)  
[getConnectedDevices\(\)](#)

**15.12.2.6 newDeviceConnected**

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

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

**Parameters**

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

**Note**

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

**15.12.2.7 saveLogToFile()**

```
void AisDeviceTracker::saveLogToFile (
    bool save )
```

Allow to collect device error message in file for debugging purpose.

**Note**

by default it will be true.



## Parameters

<i>save</i>	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](#)**15.12.2.8 setLogFilePath()**

```
void AisDeviceTracker::setLogFilePath (
    const QString & path )
```

This will help to change the log file directory.

## Note

by default it will be Document/Admiral Instrument/API

## Parameters

<i>path</i>	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](#)**15.12.2.9 updateFirmwareOnAllAvailableDevices()**

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

request firmware update for all available devices.

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

**Returns**

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

**Note**

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

**See also**

[updateFirmwareOnComPort](#)

**15.12.2.10 updateFirmwareOnComPort()**

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

update firmware on connected device at USB port.

**Parameters**

<i>comPort</i>	the communication port to connect through.
----------------	--

**Returns**

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

- [AisErrorCode::FirmwareUptodate](#)
- [AisErrorCode::ConnectionFailed](#)

**Note**

emits firmwareUpdateNotification() signal to provide firmware update progress.

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

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

- AisDeviceTracker.h

**15.13 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>
```

Inherits AisAbstractElement.

## Public Member Functions

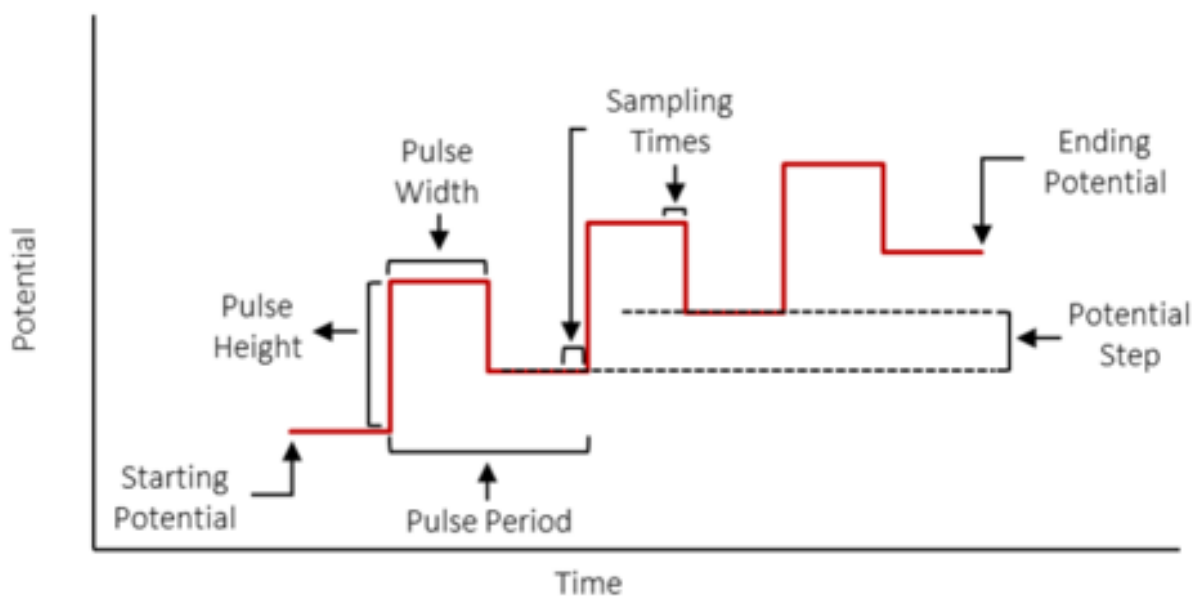
- [AisDiffPulseVoltammetryElement](#) (double startVoltage, double endVoltage, double voltageStep, double pulseHeight, double pulseWidth, double pulsePeriod)  
*the differential pulse element constructor.*
- [AisDiffPulseVoltammetryElement](#) (const [AisDiffPulseVoltammetryElement](#) &)  
*copy constructor for the [AisDiffPulseVoltammetryElement](#) object.*
- [AisDiffPulseVoltammetryElement](#) & **operator=** (const [AisDiffPulseVoltammetryElement](#) &)  
*overload equal to operator for the [AisDiffPulseVoltammetryElement](#) object.*
- QString [getName](#) () const override  
*get the name of the element.*
- QStringList [getCategory](#) () const override  
*get a list of applicable categories of the element.*
- double [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](#) ()  
*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.*

### 15.13.1 Detailed Description

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

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



Advanced control of data output for pulse experiments can be performed using the class

See also

[AisDataManipulator](#)

## 15.13.2 Constructor & Destructor Documentation

### 15.13.2.1 AisDiffPulseVoltammetryElement()

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

the differential pulse element constructor.

#### Parameters

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

## 15.13.3 Member Function Documentation

### 15.13.3.1 getAlphaFactor()

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

Get the value set for the alpha factor.

#### 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.

### 15.13.3.2 getApproxMaxCurrent()

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

get the value set for the expected maximum current.

#### Returns

the value set for the expected maximum current in Amps.

#### Note

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

### 15.13.3.3 getCategory()

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

get a list of applicable categories of the element.

#### Returns

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

### 15.13.3.4 getEndVoltage()

```
double AisDiffPulseVoltammetryElement::getEndVoltage ( ) const
```

get the value set for the ending potential value.

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

#### Returns

the value set for the ending voltage in volts.

### 15.13.3.5 getName()

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

get the name of the element.

#### Returns

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

#### 15.13.3.6 `getPulseHeight()`

```
double AisDiffPulseVoltammetryElement::getPulseHeight ( ) const
```

get the value set for the pulse height.

##### Returns

the value set for the pulse height in volts.

##### See also

[setPulseHeight](#)

#### 15.13.3.7 `getPulsePeriod()`

```
double AisDiffPulseVoltammetryElement::getPulsePeriod ( ) const
```

get the value set for the pulse period.

##### Returns

the value set for the pulse period in seconds.

##### See also

[setPulsePeriod](#)

#### 15.13.3.8 `getPulseWidth()`

```
double AisDiffPulseVoltammetryElement::getPulseWidth ( ) const
```

get the value set for the pulse width.

##### Returns

the value set for the pulse width in seconds.

##### See also

[setPulseWidth](#)

### 15.13.3.9 getQuietTime()

```
double AisDiffPulseVoltammetryElement::getQuietTime ( ) const
```

Gets the quiet time duration.

#### Returns

The quiet time duration in seconds.

### 15.13.3.10 getQuietTimeSamplingInterval()

```
double AisDiffPulseVoltammetryElement::getQuietTimeSamplingInterval ( ) const
```

gets the quiet time sampling interval.

#### Returns

samplingInterval The quiet time sampling interval to set in seconds.

### 15.13.3.11 getStartVoltage()

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

get the value set for the start voltage.

#### Returns

the value of the start voltage in volts.

### 15.13.3.12 getVStep()

```
double AisDiffPulseVoltammetryElement::getVStep ( ) const
```

get the value set for the potential step.

#### Returns

the value set for the potential step in volts.

#### See also

[setVStep](#)



### 15.13.3.13 isAutoRange()

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

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

#### Returns

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

### 15.13.3.14 isEndVoltageVsOCP()

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

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

#### Returns

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

#### See also

[setEndVoltageVsOCP](#)

### 15.13.3.15 isStartVoltageVsOCP()

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

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

#### Returns

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

#### See also

[setStartVoltageVsOCP](#)

### 15.13.3.16 setAlphaFactor()

```
void AisDiffPulseVoltammetryElement::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.

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

## Parameters

<i>alphaFactor</i>	the value for the alphaFactor ranges from 0 to 100.
--------------------	---

**15.13.3.17 setApproxMaxCurrent()**

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

set maximum current expected, for manual current range selection.

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

## Parameters

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

**15.13.3.18 setAutoRange()**

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

set to auto-select the current range.

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

**15.13.3.19 setEndVoltage()**

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

set the ending potential value.

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

## Parameters

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

**15.13.3.20 setEndVoltageVsOCP()**

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

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

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

#### Parameters

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

#### Note

by default, this is set to false.

### 15.13.3.21 setPulseHeight()

```
void AisDiffPulseVoltammetryElement::setPulseHeight (
    double pulseHeight )
```

set the value for the pulse height.

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

#### Parameters

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

### 15.13.3.22 setPulsePeriod()

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

set the value for the pulse period.

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

#### Parameters

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

### 15.13.3.23 setPulseWidth()

```
void AisDiffPulseVoltammetryElement::setPulseWidth (
    double pulseWidth )
```

set the value for the pulse width.

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

#### Parameters

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

#### See also

[setPulseHeight](#)

### 15.13.3.24 setQuietTime()

```
void AisDiffPulseVoltammetryElement::setQuietTime (
    double quietTime )
```

Sets the quiet time duration.

#### Parameters

<i>quietTime</i>	The quiet time duration to set in seconds.
------------------	--

### 15.13.3.25 setQuietTimeSamplingInterval()

```
void AisDiffPulseVoltammetryElement::setQuietTimeSamplingInterval (
    double quietTimeSamplingInterval )
```

Sets the quiet time sampling interval.

#### Parameters

<i>quietTimeSamplingInterval</i>	The quiet time sampling interval to set in seconds.
----------------------------------	---

### 15.13.3.26 setStartVoltage()

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

set the value for the start voltage.

#### Parameters

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

#### 15.13.3.27 setStartVoltageVsOCP()

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

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

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

#### Parameters

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

#### Note

by default, this is set to false.

#### 15.13.3.28 setVStep()

```
void AisDiffPulseVoltammetryElement::setVStep (
    double vStep )
```

set the value for the potential step.

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

#### Parameters

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

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

- AisDiffPulseVoltammetryElement.h

## 15.14 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>
```

Inherits AisAbstractElement.

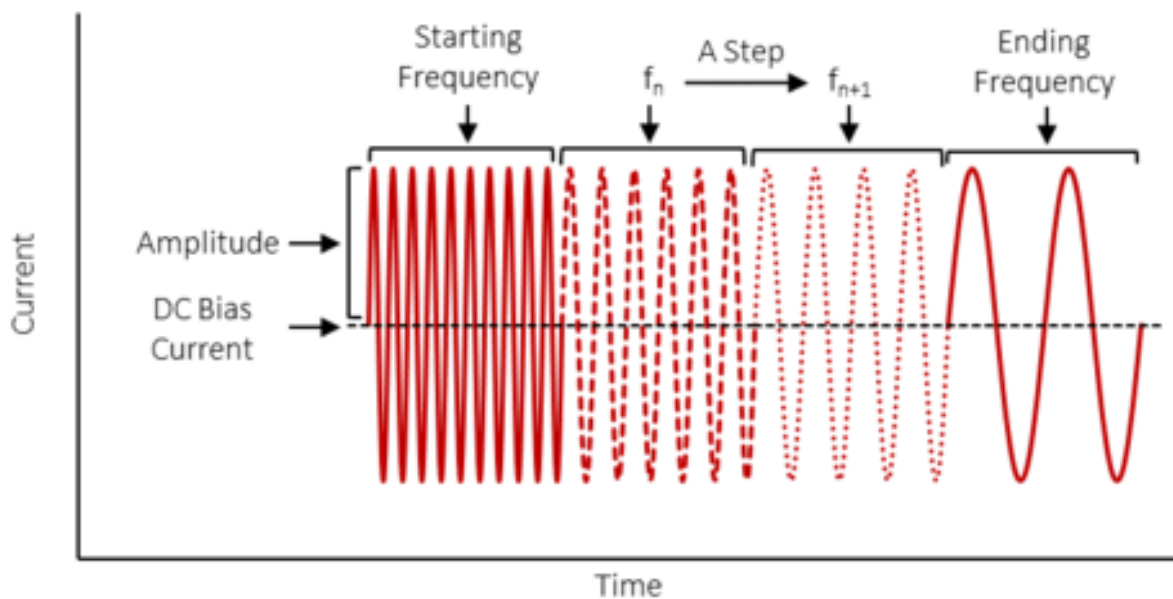
### Public Member Functions

- [AisEISGalvanostaticElement](#) (double startFrequency, double endFrequency, double stepsPerDecade, double currentBias, double currentAmplitude)  
*the EIS galvanostatic element constructor.*
- [AisEISGalvanostaticElement](#) (const [AisEISGalvanostaticElement](#) &)  
*copy constructor for the [AisEISGalvanostaticElement](#) object.*
- [AisEISGalvanostaticElement](#) & **operator=** (const [AisEISGalvanostaticElement](#) &)  
*overload equal to operator for the [AisEISGalvanostaticElement](#) object.*
- 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.*
- QString [getName](#) () const override  
*get the name of the element.*
- QStringList [getCategory](#) () const override  
*get a list of applicable categories of the element.*
- double [getStartFreq](#) () const  
*get the value set for the current starting frequency*
- void [setStartFreq](#) (double startFreq)  
*set the value for the current starting frequency.*
- double [getEndFreq](#) () const  
*the value set for the current ending frequency.*
- void [setEndFreq](#) (double endFreq)  
*set the value for the current end frequency.*
- double [getStepsPerDecade](#) () const  
*get the value set for the current frequency steps per decade.*
- void [setStepsPerDecade](#) (double stepsPerDecade)  
*set the number of the current frequency steps per decade.*
- double [getBiasCurrent](#) () const  
*get the value set for the DC bias (DC offset).*
- void [setBiasCurrent](#) (double biasCurrent)  
*set the value for the DC bias (DC offset).*
- double [getAmplitude](#) () const  
*the value to set for the AC current amplitude.*
- void [setAmplitude](#) (double amplitude)  
*set the value for the AC current amplitude.*
- void [setMinimumCycles](#) (int numberOfCycle)  
*set the minimum number of period of applied sinusoidal current at each frequency.*
- int [getMinimumCycles](#) () const  
*the value to set for minimum number of cycle Sampled.*

### 15.14.1 Detailed Description

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

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



### 15.14.2 Constructor & Destructor Documentation

#### 15.14.2.1 AisEISGalvanostaticElement()

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

the EIS galvanostatic element constructor.

#### Parameters

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

### 15.14.3 Member Function Documentation

#### 15.14.3.1 getAmplitude()

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

the value to set for the AC current amplitude.

##### Returns

the value set for the AC current amplitude in Amps.

#### 15.14.3.2 getBiasCurrent()

```
double AisEISGalvanostaticElement::getBiasCurrent ( ) const
```

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

##### Returns

the value set for the DC bias in Amps.

#### 15.14.3.3 getCategory()

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

get a list of applicable categories of the element.

##### Returns

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

#### 15.14.3.4 getEndFreq()

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

the value set for the current ending frequency.

##### Returns

the value set for the current end frequency in Hz



### 15.14.3.5 getMinimumCycles()

```
int AisEISGalvanostaticElement::getMinimumCycles ( ) const
```

the value to set for minimum number of cycle Sampled.

#### Returns

the value set for the minimum number of cycles.

### 15.14.3.6 getName()

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

get the name of the element.

#### Returns

The name of the element: "Galvanostatic EIS".

### 15.14.3.7 getQuietTime()

```
double AisEISGalvanostaticElement::getQuietTime ( ) const
```

Gets the quiet time duration.

#### Returns

The quiet time duration in seconds.

### 15.14.3.8 getQuietTimeSamplingInterval()

```
double AisEISGalvanostaticElement::getQuietTimeSamplingInterval ( ) const
```

gets the quiet time sampling interval.

#### Returns

samplingInterval The quiet time sampling interval to set in seconds.

#### 15.14.3.9 `getStartFreq()`

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

get the value set for the current starting frequency

##### Returns

the value set for the current start frequency in Hz?

#### 15.14.3.10 `getStepsPerDecade()`

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

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

##### Returns

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

#### 15.14.3.11 `setAmplitude()`

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

set the value for the AC current amplitude.

##### Parameters

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

#### 15.14.3.12 `setBiasCurrent()`

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

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

##### Parameters

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

### 15.14.3.13 setEndFreq()

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

set the value for the current end frequency.

#### Parameters

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

### 15.14.3.14 setMinimumCycles()

```
void AisEISGalvanostaticElement::setMinimumCycles (
    int numberOfCycle )
```

set the minimum number of period of applied sinusoidal current at each frequency.

#### Parameters

<i>numberOfCycle</i>	set number of cycles Sampled.
----------------------	-------------------------------

### 15.14.3.15 setQuietTime()

```
void AisEISGalvanostaticElement::setQuietTime (
    double quietTime )
```

Sets the quiet time duration.

#### Parameters

<i>quietTime</i>	The quiet time duration to set in seconds.
------------------	--

### 15.14.3.16 setQuietTimeSamplingInterval()

```
void AisEISGalvanostaticElement::setQuietTimeSamplingInterval (
    double quietTimeSamplingInterval )
```

Sets the quiet time sampling interval.

## Parameters

<i>quietTimeSamplingInterval</i>	The quiet time sampling interval to set in seconds.
----------------------------------	---

**15.14.3.17 setStartFreq()**

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

set the value for the current starting frequency.

## Parameters

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

**15.14.3.18 setStepsPerDecade()**

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

set the number of the current frequency steps per decade.

## Parameters

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

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

- AisEISGalvanostaticElement.h

**15.15 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>
```

Inherits AisAbstractElement.

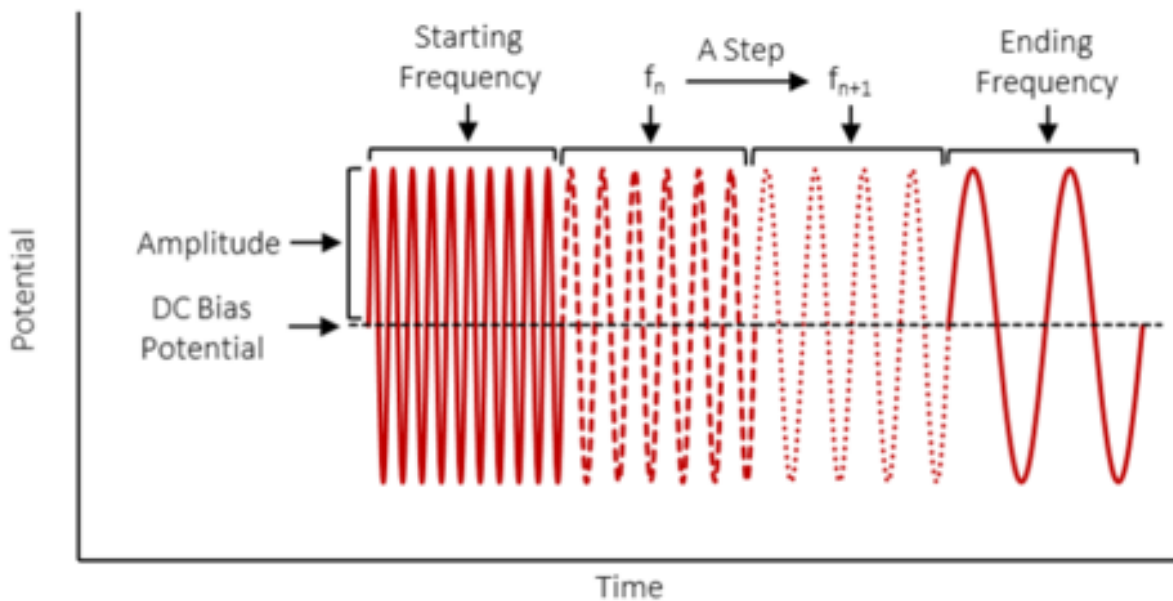
## Public Member Functions

- [AisEISPotentiostaticElement](#) (double startFrequency, double endFrequency, double stepsPerDecade, double voltageBias, double voltageAmplitude)  
*the EIS potentiostatic element*
- [AisEISPotentiostaticElement](#) (const [AisEISPotentiostaticElement](#) &)  
*copy constructor for the [AisEISPotentiostaticElement](#) object.*
- [AisEISPotentiostaticElement](#) & [operator=](#) (const [AisEISPotentiostaticElement](#) &)  
*overload equal to operator for the [AisEISPotentiostaticElement](#) object.*
- QString [getName](#) () const override  
*get the name of the element.*
- QStringList [getCategory](#) () const override  
*get a list of applicable categories of the element.*
- double [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.*
- void [setMinimumCycles](#) (int numberOfCycle)  
*set the minimum number of period of applied sinusoidal current at each frequency.*
- int [getMinimumCycles](#) () const  
*the value to set for minimum number of cycle Sampled.*

### 15.15.1 Detailed Description

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

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



### 15.15.2 Constructor & Destructor Documentation

#### 15.15.2.1 AisEISPotentiostaticElement()

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

the EIS potentiostatic element

#### Parameters

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

## 15.15.3 Member Function Documentation

### 15.15.3.1 getAmplitude()

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

the value to set for the AC voltage amplitude.

#### Returns

the value set for the AC voltage amplitude in volts.

### 15.15.3.2 getBiasVoltage()

```
double AisEISPotentiostaticElement::getBiasVoltage ( ) const
```

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

#### Returns

the value set for the DC bias in volts.

### 15.15.3.3 getCategory()

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

get a list of applicable categories of the element.

#### Returns

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

### 15.15.3.4 getEndFreq()

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

the value set for the voltage ending frequency.

#### Returns

the value set for the voltage end frequency in Hz

#### 15.15.3.5 getMinimumCycles()

```
int AisEISPotentiostaticElement::getMinimumCycles ( ) const
```

the value to settled for minimum number of cycle Sampled.

##### Returns

the value set for the minimum number of cycles.

#### 15.15.3.6 getName()

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

get the name of the element.

##### Returns

The name of the element: "Potentiostatic EIS".

#### 15.15.3.7 getQuietTime()

```
double AisEISPotentiostaticElement::getQuietTime ( ) const
```

Gets the quiet time duration.

##### Returns

The quiet time duration in seconds.

#### 15.15.3.8 getQuietTimeSamplingInterval()

```
double AisEISPotentiostaticElement::getQuietTimeSamplingInterval ( ) const
```

gets the quiet time sampling interval.

##### Returns

samplingInterval The quiet time sampling interval to set in seconds.



### 15.15.3.9 getStartFreq()

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

get the value set for the voltage starting frequency

#### Returns

the value set for the start frequency in Hz

### 15.15.3.10 getStepsPerDecade()

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

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

#### Returns

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

### 15.15.3.11 isBiasVoltageVsOCP()

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

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

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

#### Returns

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

### 15.15.3.12 setAmplitude()

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

set the value for the AC voltage amplitude.

## Parameters

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

**15.15.3.13 setBiasVoltage()**

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

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

## Parameters

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

**15.15.3.14 setBiasVoltageVsOCP()**

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

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

## Parameters

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

**15.15.3.15 setEndFreq()**

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

set the value for the voltage end frequency.

## Parameters

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

**15.15.3.16 setMinimumCycles()**

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

set the minimum number of period of applied sinusoidal current at each frequency.

**Parameters**

<i>numberOfCycle</i>	set number of cycles Sampled.
----------------------	-------------------------------

**15.15.3.17 setQuietTime()**

```
void AisEISPotentiostaticElement::setQuietTime (
    double quietTime )
```

Sets the quiet time duration.

**Parameters**

<i>quietTime</i>	The quiet time duration to set in seconds.
------------------	--

**15.15.3.18 setQuietTimeSamplingInterval()**

```
void AisEISPotentiostaticElement::setQuietTimeSamplingInterval (
    double quietTimeSamplingInterval )
```

Sets the quiet time sampling interval.

**Parameters**

<i>quietTimeSamplingInterval</i>	The quiet time sampling interval to set in seconds.
----------------------------------	---

**15.15.3.19 setStartFreq()**

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

set the value for the voltage starting frequency.

## Parameters

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

**15.15.3.20 setStepsPerDecade()**

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

set the number of the voltage frequency steps per decade.

## Parameters

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

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

- AisEISPotentiostaticElement.h

**15.16 AisErrorCode Class Reference**

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

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

**Public Types**

- enum `ErrorCode` : `uint8_t` {  
`Unknown` = 255 , `Success` = 0 , `ConnectionFailed` = 1 , `FirmwareNotSupported` = 2 ,  
`FirmwareFileNotFound` = 3 , `FirmwareUptodate` = 4 , `InvalidChannel` = 10 , `BusyChannel` = 11 ,  
`DeviceNotFound` = 13 , `ManualExperimentNotRunning` = 51 , `ExperimentNotUploaded` = 52 ,  
`ExperimentIsEmpty` = 53 ,  
`InvalidParameters` = 54 , `ChannelNotBusy` = 55 , `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 , `FailedRequest` = 254 }

## Public Member Functions

- **AisErrorCode** ([ErrorCode](#) error)
- **AisErrorCode** ([ErrorCode](#) error, [QString](#) message)
- [QString](#) [message](#) () const  
a function to get a message explaining the error.
- int [value](#) () const  
a function to get the error code.
- **operator ErrorCode** () const

### 15.16.1 Detailed Description

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

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

See also

[message](#).

### 15.16.2 Member Enumeration Documentation

#### 15.16.2.1 ErrorCode

```
enum AisErrorCode::ErrorCode : uint8_t
```

Enumerator

Unknown	indicates that the command failed for an unknown reason.
Success	indicates success.
ConnectionFailed	indicates failure connecting the plugged in device when calling <a href="#">AisDeviceTracker::connectToDeviceOnComPort</a> .
FirmwareNotSupported	indicates failure connecting the plugged in device when calling <a href="#">AisDeviceTracker::connectToDeviceOnComPort</a> 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.
ManualExperimentNotRunning	indicates that the given command applies when there is a manual experiment running on the channel but there is none.
ExperimentNotUploaded	indicates that the given command applies when an experiment has already been uploaded to the channel but there is none.
ExperimentIsEmpty	indicates that the given experiment has no elements. It need to contain at least one.
InvalidParameters	indicates that a given parameter is invalid. For example, it is out of the allowed range.

## Enumerator

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.

### 15.16.3 Member Function Documentation

#### 15.16.3.1 message()

```
QString AIS::message ( ) const
```

a function to get a message explaining the error.

#### Returns

a message that explains the error.

### 15.16.3.2 value()

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

a function to get the error code.

#### Returns

the error code

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

- AisErrorCode.h

## 15.17 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>
```

### Public Member Functions

- **AisExperiment** ()  
*this is the default constructor for the custom experiment.*
- **AisExperiment** (const **AisExperiment** &exp)  
*this is the copy constructor for the custom experiment.*
- void **operator=** (const **AisExperiment** &exp)  
*the assignment operator for the custom experiment.*
- QString **getExperimentName** () const  
*get the name of the custom experiment.*
- QString **getDescription** () const  
*get a brief description of the custom experiment.*
- QStringList **getCategory** () const  
*get the category for the custom experiment.*
- void **setExperimentName** (QString name)  
*set a name for the custom experiment.*
- void **setDescription** (QString description)  
*set a description for the experiment.*
- bool **appendElement** (AisAbstractElement &element, uint repeat=1)  
*append an element to the custom experiment.*
- bool **appendSubExperiment** (const **AisExperiment** &subExp, uint repeat=1)  
*append a sub experiment to this/(the calling) custom experiment.*

### Friends

- class **AisInstrumentHandler**

### 15.17.1 Detailed Description

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.

#### Note

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

### 15.17.2 Constructor & Destructor Documentation

#### 15.17.2.1 AisExperiment()

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

this is the copy constructor for the custom experiment.

#### Parameters

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

### 15.17.3 Member Function Documentation

#### 15.17.3.1 appendElement()

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

append an element to the custom experiment.

#### Parameters

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



**Returns**

true if appending the element was successful and false otherwise.

**Note**

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

**15.17.3.2 appendSubExperiment()**

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

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

**Parameters**

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

**Returns**

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

**15.17.3.3 getCategory()**

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

get the category for the custom experiment.

**Returns**

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

#### 15.17.3.4 getDescription()

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

get a brief description of the custom experiment.

##### Returns

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

#### 15.17.3.5 getExperimentName()

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

get the name of the custom experiment.

##### Returns

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

#### 15.17.3.6 operator=()

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

the assignment operator for the custom experiment.

##### Parameters

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

#### 15.17.3.7 setDescription()

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

set a description for the experiment.

##### Parameters

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

### 15.17.3.8 setExperimentName()

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

set a name for the custom experiment.

#### Parameters

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

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

- AisExperiment.h

## 15.18 AisExperimentNode Struct Reference

a structure containing some information regarding the running element.

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

### Public Attributes

- QString **stepName**  
*This is the name of the current element running.*
- int **stepNumber**  
*this number is the order of the element within the custom experiment.*
- int **substepNumber**  
*this number is the order of the step within the element.*
- int **cycle**  
*this number is cycle within the element.*

### 15.18.1 Detailed Description

a structure containing some information regarding the running element.

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

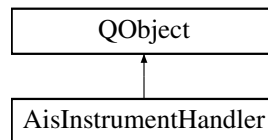
- AisDataPoints.h

## 15.19 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>
```

Inheritance diagram for AisInstrumentHandler:



### Signals

- void [deviceDisconnected](#) ()  
a signal that is emitted if the device associated with this handler has been disconnected.
- void [groundFloatStateChanged](#) (bool grounded)  
a signal that is emitted when the floating ground connection state has changed.
- void [experimentNewElementStarting](#) (uint8\_t channel, const [AisExperimentNode](#) &stepInfo)  
a signal that is emitted whenever a new elemental experiment has started.
- void [activeDCDataReady](#) (uint8\_t channel, const [AisDCData](#) &DCData)  
a signal that is emitted whenever new DC data for an active experiment are ready.
- void [idleDCDataReady](#) (uint8\_t channel, const [AisDCData](#) &DCData)  
a signal that is emitted whenever new DC data are ready when the device is in an idle state.
- void [recoveryDCDataReady](#) (uint8\_t channel, const [AisDCData](#) &DCData)  
a signal that is emitted whenever new DC recovery data are ready.
- void [activeACDataReady](#) (uint8\_t channel, const [AisACData](#) &ACData)  
a signal that is emitted whenever new AC data for an active experiment are ready.
- void [recoveryACDataReady](#) (uint8\_t channel, const [AisACData](#) &ACData)  
a signal that is emitted whenever new AC recovery data are ready.
- void [experimentStopped](#) (uint8\_t channel)  
a signal that is emitted whenever an experiment was stopped manually or has completed.
- void [experimentPaused](#) (uint8\_t channel)  
a signal that is emitted whenever an experiment was paused.
- void [experimentResumed](#) (uint8\_t channel)  
a signal that is emitted whenever an experiment was resumed.
- void [recoverDataErased](#) (bool successful)  
a signal that is emitted whenever data erase process is completed.
- 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) const  
*upload an already created custom experiment to a specific channel on the device.*
- [AisErrorCode uploadExperimentToChannel](#) (uint8\_t channel, const [AisExperiment](#) &experiment) const  
*upload an already created custom experiment to a specific channel on the device.*
- [AisErrorCode startUploadedExperiment](#) (uint8\_t channel) const  
*start the previously uploaded experiment on the specific channel.*
- [AisErrorCode 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) const  
*set IR compensation.*
- [AisErrorCode setCompRange](#) (uint8\_t channel, const [AisCompRange](#) &compRange) const  
*set a compensation range with stability factor and bandwidth index.*
- int8\_t [setLinkedChannels](#) (std::vector< uint8\_t > channels) const  
*connect several channels together in parallel mode.*
- int8\_t [setBipolarLinkedChannels](#) (std::vector< uint8\_t > channels) const  
*connect two channels together in bipolar mode.*
- bool [hasBipolarMode](#) (uint8\_t channel) const  
*tells whether the given channel is bipolar mode*
- std::vector< uint8\_t > [getLinkedChannels](#) (uint8\_t channel) const  
*get a list of channels linked to the given channel.*
- bool [isChannelBusy](#) (uint8\_t channel) const  
*tells whether the given channel is busy or not.*
- bool [isChannelPaused](#) (uint8\_t channel) const  
*tells whether the given channel has a paused experiment or not.*
- std::vector< uint8\_t > [getFreeChannels](#) () const  
*get a list of the currently free channels.*
- int [getNumberOfChannels](#) () const  
*get the number of all the channels on this device.*
- [AisErrorCode eraseRecoverData](#) () const  
*delete the recover data from device.*
- [AisErrorCode startManualExperiment](#) (uint8\_t channel) const  
*start a manual experiment.*
- [AisErrorCode setManualModeSamplingInterval](#) (uint8\_t channel, double value) const  
*set an interval for sampling the data.*
- [AisErrorCode setManualModeOCP](#) (uint8\_t channel) const  
*set open-circuit potential mode.*
- [AisErrorCode setManualModeConstantVoltage](#) (uint8\_t channel, double value) const

- set constant voltage for the manual experiment.*
- [AisErrorCode setManualModeConstantVoltage](#) (uint8\_t channel, double value, int currentRangeIndex) const  
*set constant voltage for the manual experiment and also set a manual current range.*
- [AisErrorCode 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.*
- `std::vector< std::pair< double, double > >` [getManualModeVoltageRangeList](#) (uint8\_t channel) const  
*get a list of the applicable voltage ranges to the given channel specific to your device.*

### 15.19.1 Detailed Description

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

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

### 15.19.2 Member Function Documentation

#### 15.19.2.1 activeACDataReady

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

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

#### Parameters

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

### 15.19.2.2 activeDCDataReady

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

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

#### Parameters

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

### 15.19.2.3 deviceDisconnected

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

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

### 15.19.2.4 deviceError

```
void AisInstrumentHandler::deviceError (
    uint8_t channel,
    const QString & error ) [signal]
```

a signal that is emitted whenever device send any critical error.

#### Parameters

<i>channel</i>	the channel number at which error rise.
<i>error</i>	information about error message.

#### Note

stop experiment command will automatically send on channel.

### 15.19.2.5 eraseRecoverData()

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

delete the recover data from device.

**Returns**

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

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

**15.19.2.6 experimentNewElementStarting**

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

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

**Parameters**

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

**See also**

[AisExperimentNode](#)

**15.19.2.7 experimentPaused**

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

a signal that is emitted whenever an experiment was paused.

**Parameters**

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

**15.19.2.8 experimentResumed**

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

a signal that is emitted whenever an experiment was resumed.



## Parameters

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

**15.19.2.9 experimentStopped**

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

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

## Parameters

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

**15.19.2.10 getExperimentUTCStartTime()**

```
double AisInstrumentHandler::getExperimentUTCStartTime (
    uint8_t channel ) const
```

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

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

## Parameters

<i>channel</i>	the channel for which to get the start time of the experiment.
----------------	--

## Returns

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

**15.19.2.11 getFreeChannels()**

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

get a list of the currently free channels.

## Returns

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

### 15.19.2.12 getLinkedChannels()

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

get a list of channels linked to the given channel.

#### Parameters

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

#### Returns

a list of channels linked to the channel parameter.

### 15.19.2.13 getManualModeCurrentRangeList()

```
std::vector< std::pair< double, double > > AisInstrumentHandler::getManualModeCurrentRange↵
List (
    uint8_t channel ) const
```

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

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

#### Parameters

<i>channel</i>	a valid channel number for which to check the current range.
----------------	--

#### Returns

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

### 15.19.2.14 getManualModeVoltageRangeList()

```
std::vector< std::pair< double, double > > AisInstrumentHandler::getManualModeVoltageRange↵
List (
    uint8_t channel ) const
```

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

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**

<i>channel</i>	a valid channel number for which to check the current range.
----------------	--

**Returns**

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

**15.19.2.15 getNumberOfChannels()**

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

get the number of all the channels on this device.

**Returns**

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

**15.19.2.16 groundFloatStateChanged**

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

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

**Parameters**

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

**15.19.2.17 hasBipolarMode()**

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

tells whether the given channel is bipolar mode

**Parameters**

<i>channel</i>	the channel number to check if it is bipolar mode
----------------	---

**Returns**

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

**15.19.2.18 idleDCDataReady**

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

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

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

**Parameters**

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

**15.19.2.19 isChannelBusy()**

```
bool AisInstrumentHandler::isChannelBusy (
    uint8_t channel ) const
```

tells whether the given channel is busy or not.

**Parameters**

<i>channel</i>	the channel number to check if it is busy or not.
----------------	---

**Returns**

true only if given a valid channel number that has either a running or a paused experiment.

**15.19.2.20 isChannelPaused()**

```
bool AisInstrumentHandler::isChannelPaused (
    uint8_t channel ) const
```

tells whether the given channel has a paused experiment or not.

**Parameters**

<i>channel</i>	the channel number to check if it has a paused experiment.
----------------	--

**Returns**

true only if given a valid channel number that has an experiment that has been paused.

**15.19.2.21 pauseExperiment()**

```
AisErrorCode AisInstrumentHandler::pauseExperiment (
    uint8_t channel ) const
```

pause a running experiment on the channel.

**Parameters**

<i>channel</i>	the channel number to pause the experiment on.
----------------	--

**Returns**

true if an experiment was successfully paused on the channel and false otherwise. If not successful, possible returned errors are:

- [AisErrorCode::FailedToPauseExperiment](#)
- [AisErrorCode::DeviceNotFound](#)
- [AisErrorCode::InvalidChannel](#)
- [AisErrorCode::ChannelNotBusy](#)

This will return [AisErrorCode::Success](#) only if there is currently a running experiment on a valid channel on a connected device.

**15.19.2.22 recoverDataErased**

```
void AisInstrumentHandler::recoverDataErased (
    bool successful ) [signal]
```

a signal that is emitted whenever data erase process is completed.

**Parameters**

<i>successful</i>	is true on erased correctly, and false on data is not erased.
-------------------	---

### 15.19.2.23 recoveryACDataReady

```
void AisInstrumentHandler::recoveryACDataReady (
    uint8_t channel,
    const AisACData & ACData ) [signal]
```

a signal that is emitted whenever new AC recovery data are ready.

#### Parameters

<i>channel</i>	the channel number from which the AC data are recovered from.
<i>ACData</i>	the AC data that just arrived.

### 15.19.2.24 recoveryDCDataReady

```
void AisInstrumentHandler::recoveryDCDataReady (
    uint8_t channel,
    const AisDCData & DCData ) [signal]
```

a signal that is emitted whenever new DC recovery data are ready.

#### Parameters

<i>channel</i>	the channel number from which the DC data are recovered from.
<i>DCData</i>	the DC data that just arrived.

### 15.19.2.25 resumeExperiment()

```
AisErrorCode AisInstrumentHandler::resumeExperiment (
    uint8_t channel ) const
```

resume a paused experiment on the channel.

#### Parameters

<i>channel</i>	the channel number to resume the experiment on.
----------------	---

#### Returns

[AisErrorCode::Success](#) if an experiment was successfully resumed on the channel. If not successful, possible returned errors are:

- [AisErrorCode::FailedToResumeExperiment](#)
- [AisErrorCode::DeviceNotFound](#)
- [AisErrorCode::InvalidChannel](#)

- [AisErrorCode::ChannelNotBusy](#)

This will return [AisErrorCode::Success](#) only if there is currently a paused experiment on a valid channel on a connected device.

#### 15.19.2.26 setBipolarLinkedChannels()

```
int8_t AisInstrumentHandler::setBipolarLinkedChannels (
    std::vector< uint8_t > channels ) const
```

connect two channels together in bipolar mode.

You may combine two channels to expand the voltage range to include negative voltages. Note that this is only applicable to the cyclor model. For 4 channel Cyclor models, you can combine channels 1 and 2 or channels 3 and 4. You cannot use any other channel combinations.

##### Parameters

<i>channels</i>	a list of two channels to be operate 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 cyclor model.

#### 15.19.2.27 setCompRange()

```
AisErrorCode AisInstrumentHandler::setCompRange (
    uint8_t channel,
    const AisCompRange & compRange ) const
```

set a compensation range with stability factor and bandwidth index.

##### Parameters

<i>channel</i>	the channel for which to set the compensation range.
<i>compRange</i>	an object of type compRange that is initialized with a stability factor (0-10) and a bandwidth index (0-10).

##### Returns

[AisErrorCode::Success](#) if setting the IR compensation was successful. If not successful, possible returned errors are:

- [AisErrorCode::DeviceNotFound](#)
- [AisErrorCode::InvalidChannel](#)
- [AisErrorCode::InvalidParameters](#)

See also

[AisCompRange](#)

### 15.19.2.28 setIRComp()

```
AisErrorCode AisInstrumentHandler::setIRComp (
    uint8_t channel,
    double uncompensatedResistance,
    double compensationLevel ) const
```

set IR compensation.

#### Parameters

<i>channel</i>	the channel for which to set the IR compensation.
<i>uncompensatedResistance</i>	the value of the uncompensated resistance in Ohms.
<i>compensationLevel</i>	the compensation percentage (0%-100%). This is unit-less.

#### Returns

[AisErrorCode::Success](#) if setting the IR compensation was successful. If not successful, possible returned errors are:

- [AisErrorCode::DeviceNotFound](#)
- [AisErrorCode::InvalidChannel](#)
- [AisErrorCode::InvalidParameters](#)

### 15.19.2.29 setLinkedChannels()

```
int8_t AisInstrumentHandler::setLinkedChannels (
    std::vector< uint8_t > channels ) const
```

connect several channels together in parallel mode.

You may connect a list of channels so you can get a higher combined output current of all channels. Note that this is only applicable to the cycler model.

#### Parameters

<i>channels</i>	a list of channels to be linked.
-----------------	----------------------------------



**Returns**

the master channel out of the given list of channels. The master channel is your interface to upload an experiment to and then control it.

**Note**

this functionality is only applicable to the cyclor model.

**15.19.2.30 setManualModeConstantCurrent()**

```
AisErrorCode AisInstrumentHandler::setManualModeConstantCurrent (
    uint8_t channel,
    double value ) const
```

set constant current for the manual experiment.

**Parameters**

<i>channel</i>	a valid channel number to set a constant voltage for.
<i>value</i>	the value to set the constant current in Amps.

**Returns**

[AisErrorCode::Success](#) if setting the constant current was successful. If not successful, possible returned errors are:

- [AisErrorCode::ManualExperimentNotRunning](#)
- [AisErrorCode::DeviceNotFound](#)
- [AisErrorCode::InvalidChannel](#)
- [AisErrorCode::DeviceCommunicationFailed](#)
- [AisErrorCode::FailedToSetManualModeConstantCurrent](#)

**15.19.2.31 setManualModeConstantVoltage() [1/2]**

```
AisErrorCode AisInstrumentHandler::setManualModeConstantVoltage (
    uint8_t channel,
    double value ) const
```

set constant voltage for the manual experiment.

**Parameters**

<i>channel</i>	a valid channel number to set a constant voltage for.
<i>value</i>	the value to set the constant voltage in volts.

## Returns

[AisErrorCode::Success](#) if setting the constant voltage was successful. If not successful, possible returned errors are:

- [AisErrorCode::FailedToSetManualModeConstantVoltage](#)
- [AisErrorCode::ManualExperimentNotRunning](#)
- [AisErrorCode::DeviceNotFound](#)
- [AisErrorCode::InvalidChannel](#)

### 15.19.2.32 setManualModeConstantVoltage() [2/2]

```
AisErrorCode AisInstrumentHandler::setManualModeConstantVoltage (
    uint8_t channel,
    double value,
    int currentRangeIndex ) const
```

set constant voltage for the manual experiment and also set a manual current range.

## Parameters

<i>channel</i>	a valid channel number to set a constant voltage for.
<i>value</i>	the value to set the constant voltage in volts.
<i>currentRangeIndex</i>	the index of the desired current range.

## Returns

[AisErrorCode::Success](#) if setting the constant voltage was successful. You can get a list of the available ranges for your model by calling [getManualModeCurrentRangeList](#). If not successful, possible returned errors are:

- [AisErrorCode::FailedToSetManualModeConstantVoltage](#)
- [AisErrorCode::FailedToSetManualModeCurrentRange](#)
- [AisErrorCode::ManualExperimentNotRunning](#)
- [AisErrorCode::DeviceNotFound](#)
- [AisErrorCode::InvalidChannel](#)

### 15.19.2.33 setManualModeCurrentAutorange()

```
AisErrorCode AisInstrumentHandler::setManualModeCurrentAutorange (
    uint8_t channel ) const
```

enable current autoranging for the manual experiment.

## Parameters

<i>channel</i>	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](#)

**15.19.2.34 setManualModeCurrentRange()**

```
AisErrorCode AisInstrumentHandler::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.

## Parameters

<i>channel</i>	a valid channel number to set the current range for.
<i>currentRangeIndex</i>	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](#)

**15.19.2.35 setManualModeOCP()**

```
AisErrorCode AisInstrumentHandler::setManualModeOCP (
    uint8_t channel ) const
```

set open-circuit potential mode.

To apply the set potential or current, leave the open circuit potential mode off. This operation is reversed automatically when calling either [setManualModeConstantVoltage\(\)](#) or [setManualModeConstantCurrent\(\)](#)

## Parameters

<i>channel</i>	a valid channel number to set open circuit mode on.
----------------	---

## Returns

[AisErrorCode::Success](#) if turning on the open circuit mode was successful. If not successful, possible returned errors are:

- [AisErrorCode::ManualExperimentNotRunning](#)
- [AisErrorCode::DeviceNotFound](#)
- [AisErrorCode::InvalidChannel](#)
- [AisErrorCode::DeviceCommunicationFailed](#)

**15.19.2.36 setManualModeSamplingInterval()**

```
AisErrorCode AisInstrumentHandler::setManualModeSamplingInterval (
    uint8_t channel,
    double value ) const
```

set an interval for sampling the data.

## Parameters

<i>channel</i>	the channel to set the sampling interval for.
<i>value</i>	the value for the sampling interval in seconds.

## Returns

[AisErrorCode::Success](#) if the operation was set successfully. If not successful, possible returned errors are:

- [AisErrorCode::DeviceNotFound](#)
- [AisErrorCode::Unknown](#)
- [AisErrorCode::InvalidChannel](#)

**15.19.2.37 setManualModeVoltageAutorange()**

```
AisErrorCode AisInstrumentHandler::setManualModeVoltageAutorange (
    uint8_t channel ) const
```

enable voltage autoranging for the manual experiment.

## Parameters

<i>channel</i>	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](#)

**15.19.2.38 setManualModeVoltageRange()**

```
AisErrorCode AisInstrumentHandler::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.

## Parameters

<i>channel</i>	a valid channel number to set the voltage range for.
<i>voltageRangeIndex</i>	the index of the desired voltage range.

## Returns

[AisErrorCode::Success](#) if setting the voltage range 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](#)

**15.19.2.39 skipExperimentStep()**

```
AisErrorCode AisInstrumentHandler::skipExperimentStep (
    uint8_t channel ) const
```

skip the current experiment step and proceed to the next.

When running an element that has several steps like going from CC to CV, then skipping the step goes to the next step within the element. When having several elements in the custom experiment and the current element has one step or we are at the last step within the element, then skipping the step results in going to the next element. If this is the final step of the final element, the experiment will stop.

## Parameters

<i>channel</i>	a valid channel number with an experiment to skip the step.
----------------	---

## Returns

[AisErrorCode::Success](#) the experiment step was successfully skipped If not successful, possible returned errors are:

- [AisErrorCode::DeviceNotFound](#)
- [AisErrorCode::InvalidChannel](#)
- [AisErrorCode::ChannelNotBusy](#)
- [AisErrorCode::DeviceCommunicationFailed](#)

**15.19.2.40 startIdleSampling()**

```
AisErrorCode AisInstrumentHandler::startIdleSampling (  
    uint8_t channel ) const
```

start idle sampling when an experiment is neither uploaded nor running on the specified channel.

## Parameters

<i>channel</i>	the channel number to start collecting idle data on.
----------------	--

## 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](#)

**15.19.2.41 startManualExperiment()**

```
AisErrorCode AisInstrumentHandler::startManualExperiment (  
    uint8_t channel ) const
```

start a manual experiment.

With manual experiments, users can turn on any connected channel and toggle between open circuit mode and voltage or current setpoints that can be changed in real-time and run for indefinite periods.

## Parameters

<i>channel</i>	a valid channel number to run the manual experiment on.
----------------	---

## Returns

[AisErrorCode::Success](#) if the manual experiment was successfully started. If not successful, possible returned errors are:

- [AisErrorCode::FailedManualModeStartExperiment](#)
- [AisErrorCode::DeviceNotFound](#)
- [AisErrorCode::InvalidChannel](#)
- [AisErrorCode::BusyChannel](#)

**15.19.2.42 startUploadedExperiment()**

```
AisErrorCode AisInstrumentHandler::startUploadedExperiment (
    uint8_t channel ) const
```

start the previously uploaded experiment on the specific channel.

## Parameters

<i>channel</i>	the channel number to start the experiment on.
----------------	--

## Returns

[AisErrorCode::Success](#) if the experiment was successfully started on the channel. If not successful, possible returned errors are:

- [AisErrorCode::DeviceCommunicationFailed](#)
- [AisErrorCode::ExperimentNotUploaded](#)
- [AisErrorCode::DeviceNotFound](#)
- [AisErrorCode::InvalidChannel](#)
- [AisErrorCode::BusyChannel](#)

## See also

[uploadExperimentToChannel](#)  
[isChannelBusy](#)

**15.19.2.43 stopExperiment()**

```
AisErrorCode AisInstrumentHandler::stopExperiment (
    uint8_t channel ) const
```

stop a running or a paused experiment on the channel.

## Parameters

<i>channel</i>	the channel number to stop the experiment on.
----------------	---

## Returns

[AisErrorCode::Success](#) if an experiment was successfully stopped on the channel. If not successful, possible returned errors are:

- [AisErrorCode::FailedToStopExperiment](#)
- [AisErrorCode::DeviceNotFound](#)
- [AisErrorCode::InvalidChannel](#)

This will only return [AisErrorCode::Success](#) if there is currently a running or a paused experiment on a valid channel on a connected device.

**15.19.2.44 uploadExperimentToChannel()** [1/2]

```
AisErrorCode AisInstrumentHandler::uploadExperimentToChannel (
    uint8_t channel,
    const AisExperiment & experiment ) const
```

upload an already created custom experiment to a specific channel on the device.

Any running experiment is run on a specific device on a specific channel. This function uploads an experiment to a channel so that you may start, pause, resume and stop the experiment. All of these four control functionalities and others require a channel number to control the experiment. Therefore, if we have several channels, we need to keep track of which experiment is on which channel.

## Parameters

<i>channel</i>	the channel number to upload the experiment to.
<i>experiment</i>	the custom experiment to be uploaded to the channel.

## Returns

[AisErrorCode::Success](#) if the experiment was successfully uploaded to the channel. If not successful, possible returned errors are:

- [AisErrorCode::FailedToUploadExperiment](#)
- [AisErrorCode::ExperimentIsEmpty](#)
- [AisErrorCode::DeviceNotFound](#)
- [AisErrorCode::InvalidChannel](#)
- [AisErrorCode::BusyChannel](#)
- [AisErrorCode::InvalidParameters](#)

This returns [AisErrorCode::Success](#) only when given a valid channel number that is not busy on a connected device.

## See also

[isChannelBusy](#)



### 15.19.2.45 uploadExperimentToChannel() [2/2]

```
AisErrorCode AisInstrumentHandler::uploadExperimentToChannel (
    uint8_t channel,
    std::shared_ptr< AisExperiment > experiment ) const
```

upload an already created custom experiment to a specific channel on the device.

Any running experiment is run on a specific device on a specific channel. This function uploads an experiment to a channel so that you may start, pause, resume and stop the experiment. All of these four control functionalities and others require a channel number to control the experiment. Therefore, if we have several channels, we need to keep track of which experiment is on which channel.

#### Parameters

<i>channel</i>	the channel number to upload the experiment to.
<i>experiment</i>	the custom experiment to be uploaded to the channel.

#### Returns

[AisErrorCode::Success](#) if the experiment was successfully uploaded to the channel. If not successful, possible returned errors are:

- [AisErrorCode::FailedToUploadExperiment](#)
- [AisErrorCode::ExperimentIsEmpty](#)
- [AisErrorCode::DeviceNotFound](#)
- [AisErrorCode::InvalidChannel](#)
- [AisErrorCode::BusyChannel](#)
- [AisErrorCode::InvalidParameters](#)

This returns [AisErrorCode::Success](#) only when given a valid channel number that is not busy on a connected device.

#### See also

[isChannelBusy](#)

The documentation for this class was generated from the following file:

- AisInstrumentHandler.h

## 15.20 AisMottSchottkyElement Class Reference

This class performs Mott-Schottky analysis on the working electrode for a specified range of potentials.

```
#include <AisMottSchottkyElement.h>
```

Inherits [AisAbstractElement](#).

## Public Member Functions

- [AisMottSchottkyElement](#) (double startingPotential, double endingPotential, double voltageStep, double startFrequency, double endFrequency, double stepsPerDecade, double amplitude, double 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.*
- double [getMinCycles](#) () const  
*Get the minimum number of cycles per frequency step.*
- void [setMinCycles](#) (double 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 `getQuietTimeSamplInterval` () const  
*Get the sampling interval during the quiet time.*
- void `setQuietTimeSamplInterval` (double quietTimeSamplInterval)  
*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 `getStepQuietSamplInterval` () const  
*Get the sampling interval during the quiet time after each potential step.*
- void `setStepQuietSamplInterval` (double stepQuietTimeSamplInterval)  
*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).*

### 15.20.1 Detailed Description

This class performs Mott-Schottky analysis on the working electrode for a specified range of potentials.

The Mott-Schottky experiment is used to analyze the electrochemical properties of a working electrode by sweeping the potential from a starting potential to an ending potential in discrete steps. At each potential step, Electrochemical Impedance Spectroscopy (EIS) is performed with varying frequencies to gather impedance data.

The experiment can be configured with various parameters, including the starting and ending potentials, voltage step size, starting and ending frequencies, steps per frequency decade, AC excitation amplitude, and the minimum number of cycles per frequency.

#### Note

The potential sweep is done in a staircase manner, with each step consisting of a quiet time followed by an EIS measurement.

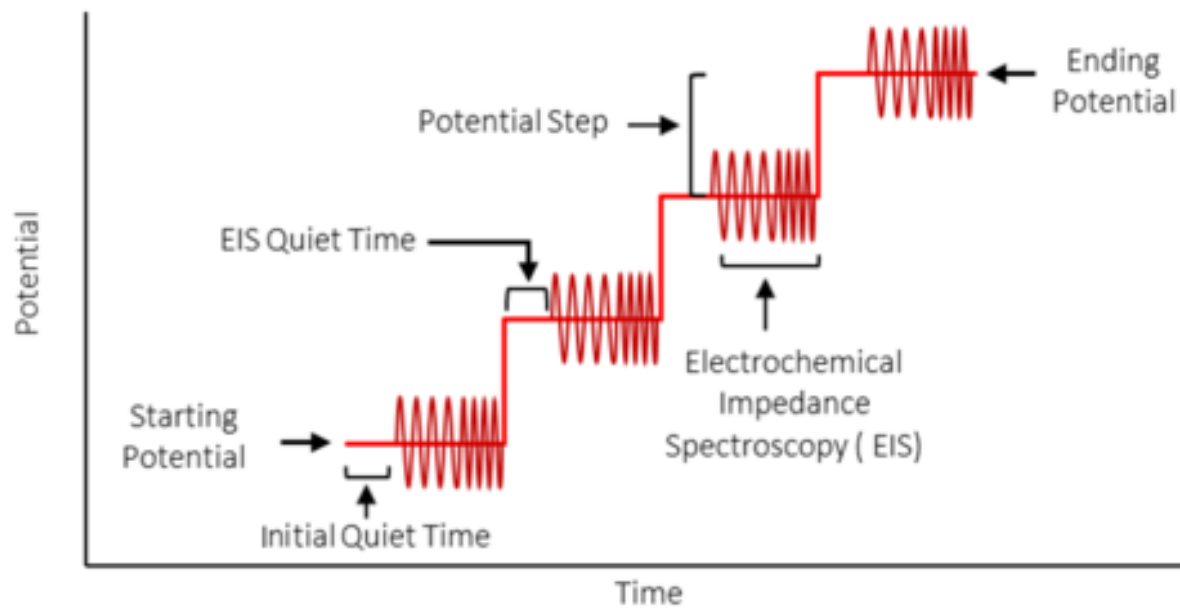


Figure 15.2 MottSchottky

## 15.20.2 Constructor & Destructor Documentation

### 15.20.2.1 AisMottSchottkyElement() [1/2]

```
AisMottSchottkyElement::AisMottSchottkyElement (
    double startingPotential,
    double endingPotential,
    double voltageStep,
    double startFrequency,
    double endFrequency,
    double stepsPerDecade,
    double amplitude,
    double minCycles ) [explicit]
```

Constructor for the Mott-Schottky experiment element.

#### Parameters

<i>startingPotential</i>	The starting potential (voltage) for the experiment.
<i>endingPotential</i>	The ending potential (voltage) for the experiment.
<i>voltageStep</i>	The voltage step size between each potential during the experiment.
<i>startFrequency</i>	The starting frequency for the EIS measurement.
<i>endFrequency</i>	The ending frequency for the EIS measurement.
<i>stepsPerDecade</i>	The number of frequency steps per decade.
<i>amplitude</i>	The amplitude of the AC signal used in the EIS measurement.
<i>minCycles</i>	The minimum number of cycles per frequency step during the EIS measurement.

### 15.20.2.2 AisMottSchottkyElement() [2/2]

```
AisMottSchottkyElement::AisMottSchottkyElement (
    const AisMottSchottkyElement & other ) [explicit]
```

Copy constructor for the [AisMottSchottkyElement](#) object.

#### Parameters

<i>other</i>	The object to copy from.
--------------	--------------------------

## 15.20.3 Member Function Documentation

### 15.20.3.1 getAmplitude()

```
double AisMottSchottkyElement::getAmplitude ( ) const
```

Get the amplitude of the AC signal used in the EIS measurement.

#### Returns

The AC amplitude in volts.

### 15.20.3.2 getCategory()

```
QStringList AisMottSchottkyElement::getCategory ( ) const [override]
```

Get a list of applicable categories of the experiment element.

#### Returns

A list of categories where the experiment is applicable, such as "Advanced Experiments".

### 15.20.3.3 getEndFrequency()

```
double AisMottSchottkyElement::getEndFrequency ( ) const
```

Get the ending frequency for the EIS measurement.

#### Returns

The ending frequency in Hz.

#### 15.20.3.4 getEndingPotential()

```
double AisMottSchottkyElement::getEndingPotential ( ) const
```

Get the ending potential for the experiment.

##### Returns

The ending potential in volts.

#### 15.20.3.5 getMinCycles()

```
double AisMottSchottkyElement::getMinCycles ( ) const
```

Get the minimum number of cycles per frequency step.

##### Returns

The minimum number of cycles.

#### 15.20.3.6 getName()

```
QString AisMottSchottkyElement::getName ( ) const [override]
```

Get the name of the experiment element.

##### Returns

The name of the element, "Mott-Schottky".

#### 15.20.3.7 getQuietTime()

```
double AisMottSchottkyElement::getQuietTime ( ) const
```

Get the quiet time before starting the EIS measurement.

##### Returns

The quiet time in seconds.

### 15.20.3.8 getQuietTimeSampInterval()

```
double AisMottSchottkyElement::getQuietTimeSampInterval ( ) const
```

Get the sampling interval during the quiet time.

#### Returns

The sampling interval in seconds.

### 15.20.3.9 getStartFrequency()

```
double AisMottSchottkyElement::getStartFrequency ( ) const
```

Get the starting frequency for the EIS measurement.

#### Returns

The starting frequency in Hz.

### 15.20.3.10 getStartingPotential()

```
double AisMottSchottkyElement::getStartingPotential ( ) const
```

Get the starting potential for the experiment.

#### Returns

The starting potential in volts.

### 15.20.3.11 getStepQuietSampInterval()

```
double AisMottSchottkyElement::getStepQuietSampInterval ( ) const
```

Get the sampling interval during the quiet time after each potential step.

#### Returns

The sampling interval in seconds.

#### 15.20.3.12 getStepQuietTime()

```
double AisMottSchottkyElement::getStepQuietTime ( ) const
```

Get the quiet time after each potential step before starting the EIS measurement.

##### Returns

The quiet time after each potential step in seconds.

#### 15.20.3.13 getStepsPerDecade()

```
double AisMottSchottkyElement::getStepsPerDecade ( ) const
```

Get the number of frequency steps per decade.

##### Returns

The number of steps per decade.

#### 15.20.3.14 getVoltageStep()

```
double AisMottSchottkyElement::getVoltageStep ( ) const
```

Get the voltage step size between each potential.

##### Returns

The voltage step size in volts.

#### 15.20.3.15 isEndVoltageVsOCP()

```
bool AisMottSchottkyElement::isEndVoltageVsOCP ( ) const
```

Check if the ending voltage is measured versus the open circuit potential (OCP).

##### Returns

True if the ending voltage is measured versus OCP, false otherwise.



### 15.20.3.16 isStartVoltageVsOCP()

```
bool AisMottSchottkyElement::isStartVoltageVsOCP ( ) const
```

Check if the starting voltage is measured versus the open circuit potential (OCP).

#### Returns

True if the starting voltage is measured versus OCP, false otherwise.

### 15.20.3.17 operator=()

```
AisMottSchottkyElement & AisMottSchottkyElement::operator= (
    const AisMottSchottkyElement & other )
```

Assignment operator for the [AisMottSchottkyElement](#) object.

#### Parameters

<i>other</i>	The object to assign from.
--------------	----------------------------

#### Returns

A reference to the assigned object.

### 15.20.3.18 setAmplitude()

```
void AisMottSchottkyElement::setAmplitude (
    double amplitude )
```

Set the amplitude of the AC signal used in the EIS measurement.

#### Parameters

<i>amplitude</i>	The AC amplitude in volts.
------------------	----------------------------

### 15.20.3.19 setEndFrequency()

```
void AisMottSchottkyElement::setEndFrequency (
    double endFrequency )
```

Set the ending frequency for the EIS measurement.

## Parameters

<i>endFrequency</i>	The ending frequency in Hz.
---------------------	-----------------------------

**15.20.3.20 setEndingPotential()**

```
void AisMottSchottkyElement::setEndingPotential (
    double endingPotential )
```

Set the ending potential for the experiment.

## Parameters

<i>endingPotential</i>	The ending potential in volts.
------------------------	--------------------------------

**15.20.3.21 setEndVoltageVsOCP()**

```
void AisMottSchottkyElement::setEndVoltageVsOCP (
    bool endVsOCP )
```

Set whether the ending voltage is measured versus the open circuit potential (OCP).

## Parameters

<i>endVsOCP</i>	True if the ending voltage is measured versus OCP, false otherwise.
-----------------	---

**15.20.3.22 setMinCycles()**

```
void AisMottSchottkyElement::setMinCycles (
    double minCycles )
```

Set the minimum number of cycles per frequency step.

## Parameters

<i>minCycles</i>	The minimum number of cycles.
------------------	-------------------------------

**15.20.3.23 setQuietTime()**

```
void AisMottSchottkyElement::setQuietTime (
    double quietTime )
```

Set the quiet time before starting the EIS measurement.

**Parameters**

<i>quietTime</i>	The quiet time in seconds.
------------------	----------------------------

**15.20.3.24 setQuietTimeSampInterval()**

```
void AisMottSchottkyElement::setQuietTimeSampInterval (
    double quietTimeSampInterval )
```

Set the sampling interval during the quiet time.

**Parameters**

<i>quietTimeSampInterval</i>	The sampling interval in seconds.
------------------------------	-----------------------------------

**15.20.3.25 setStartFrequency()**

```
void AisMottSchottkyElement::setStartFrequency (
    double startFrequency )
```

Set the starting frequency for the EIS measurement.

**Parameters**

<i>startFrequency</i>	The starting frequency in Hz.
-----------------------	-------------------------------

**15.20.3.26 setStartingPotential()**

```
void AisMottSchottkyElement::setStartingPotential (
    double startingPotential )
```

Set the starting potential for the experiment.

## Parameters

<i>startingPotential</i>	The starting potential in volts.
--------------------------	----------------------------------

**15.20.3.27 setStartVoltageVsOCP()**

```
void AisMottSchottkyElement::setStartVoltageVsOCP (
    bool startVsOCP )
```

Set whether the starting voltage is measured versus the open circuit potential (OCP).

## Parameters

<i>startVsOCP</i>	True if the starting voltage is measured versus OCP, false otherwise.
-------------------	---

**15.20.3.28 setStepQuietSampInterval()**

```
void AisMottSchottkyElement::setStepQuietSampInterval (
    double stepQuietTimeSampInterval )
```

Set the sampling interval during the quiet time after each potential step.

## Parameters

<i>stepQuietTimeSampInterval</i>	The sampling interval in seconds.
----------------------------------	-----------------------------------

**15.20.3.29 setStepQuietTime()**

```
void AisMottSchottkyElement::setStepQuietTime (
    double stepQuietTime )
```

Set the quiet time after each potential step before starting the EIS measurement.

## Parameters

<i>stepQuietTime</i>	The quiet time after each potential step in seconds.
----------------------	--

**15.20.3.30 setStepsPerDecade()**

```
void AisMottSchottkyElement::setStepsPerDecade (
    double stepsPerDecade )
```

Set the number of frequency steps per decade.

**Parameters**

<i>stepsPerDecade</i>	The number of steps per decade.
-----------------------	---------------------------------

**15.20.3.31 setVoltageStep()**

```
void AisMottSchottkyElement::setVoltageStep (
    double voltageStep )
```

Set the voltage step size between each potential.

**Parameters**

<i>voltageStep</i>	The voltage step size in volts.
--------------------	---------------------------------

The documentation for this class was generated from the following file:

- AisMottSchottkyElement.h

**15.21 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>
```

Inherits AisAbstractElement.

**Public Member Functions**

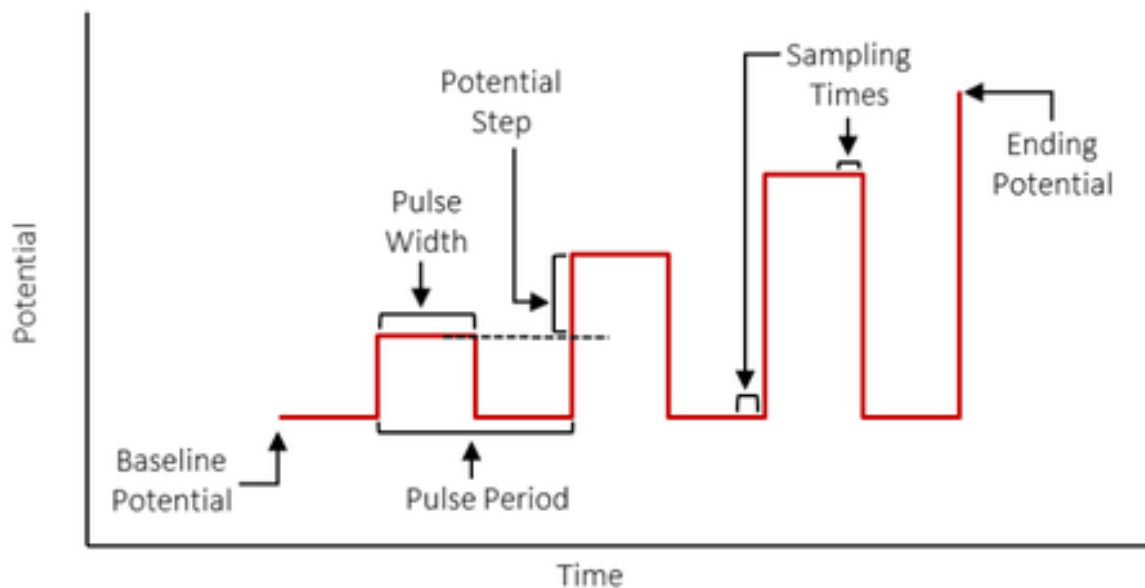
- [AisNormalPulseVoltammetryElement](#) (double startVoltage, double endVoltage, double voltageStep, double pulseWidth, double pulsePeriod)  
*the normal-pulse-voltammetry element constructor*
- **AisNormalPulseVoltammetryElement** (const [AisNormalPulseVoltammetryElement](#) &)  
*copy constructor for the [AisNormalPulseVoltammetryElement](#) object.*
- [AisNormalPulseVoltammetryElement](#) & **operator=** (const [AisNormalPulseVoltammetryElement](#) &)  
*overload equal to operator for the [AisNormalPulseVoltammetryElement](#) object.*
- QString [getName](#) () const override

- get the name of the element.*
- QStringList [getCategory](#) () const override
  - get a list of applicable categories of the element.*
- double [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](#) ()
  - 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.*

### 15.21.1 Detailed Description

This experiment holds the working electrode at a **baseline potential** during the **quiet time**, then applies a train of pulses, which increase in amplitude until the **final potential** is reached.

The **potential step** is the magnitude of this incremental increase. The **pulse width** is the amount of time between the rising and falling edge of a pulse. The **pulse period** is the amount of time between the beginning of one pulse and the beginning of the next.



Advanced control of data output for pulse experiments can be performed using the class

See also

[AisDataManipulator](#)

### 15.21.2 Constructor & Destructor Documentation

#### 15.21.2.1 AisNormalPulseVoltammetryElement()

```
AisNormalPulseVoltammetryElement::AisNormalPulseVoltammetryElement (
    double startVoltage,
    double endVoltage,
    double voltageStep,
    double pulseWidth,
    double pulsePeriod ) [explicit]
```

the normal-pulse-voltammetry element constructor

**Parameters**

<i>startVoltage</i>	the value of the starting potential in volts
<i>endVoltage</i>	the value of the ending potential in volts
<i>voltageStep</i>	the value set for the voltage step in volts.
<i>pulseWidth</i>	the value for the pulse width in seconds.
<i>pulsePeriod</i>	the value for the pulse period in seconds.

**15.21.3 Member Function Documentation****15.21.3.1 getAlphaFactor()**

```
double AisNormalPulseVoltammetryElement::getAlphaFactor ( )
```

Get the value set for the alpha factor.

**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.

**15.21.3.2 getApproxMaxCurrent()**

```
double AisNormalPulseVoltammetryElement::getApproxMaxCurrent ( ) const
```

get the value set for the expected maximum current.

**Returns**

the value set for the expected maximum current in Amps.

**Note**

if nothing was manually set, the device will auto-select the current range and the return value will be positive infinity.



### 15.21.3.3 getCategory()

```
QStringList AisNormalPulseVoltammetryElement::getCategory ( ) const [override]
```

get a list of applicable categories of the element.

#### Returns

A list of applicable categories: ("Potentiostatic Control", "Basic Voltammetry", "Pulse Voltammetry").

### 15.21.3.4 getEndVoltage()

```
double AisNormalPulseVoltammetryElement::getEndVoltage ( ) const
```

get the value set for the ending potential value.

This is the value of the voltage at which the experiment will stop.

#### Returns

the value set for the ending voltage in volts.

### 15.21.3.5 getName()

```
QString AisNormalPulseVoltammetryElement::getName ( ) const [override]
```

get the name of the element.

#### Returns

The name of the element: "Normal Pulse Potential Voltammetry".

### 15.21.3.6 getPulsePeriod()

```
double AisNormalPulseVoltammetryElement::getPulsePeriod ( ) const
```

get the value set for the pulse period.

#### Returns

the value for the pulse period in seconds.

#### See also

[setPulsePeriod](#)

### 15.21.3.7 `getPulseWidth()`

```
double AisNormalPulseVoltammetryElement::getPulseWidth ( ) const
```

get the value set for the pulse width

#### Returns

the value of the pulse width in seconds.

#### See also

[setPulseWidth](#)

### 15.21.3.8 `getQuietTime()`

```
double AisNormalPulseVoltammetryElement::getQuietTime ( ) const
```

Gets the quiet time duration.

#### Returns

The quiet time duration in seconds.

### 15.21.3.9 `getQuietTimeSamplingInterval()`

```
double AisNormalPulseVoltammetryElement::getQuietTimeSamplingInterval ( ) const
```

gets the quiet time sampling interval.

#### Returns

quiet time The quiet time sampling interval to set in seconds.

### 15.21.3.10 `getStartVoltage()`

```
double AisNormalPulseVoltammetryElement::getStartVoltage ( ) const
```

get the value set for the start voltage.

#### Returns

the value of the start voltage in volts.

### 15.21.3.11 getVStep()

```
double AisNormalPulseVoltammetryElement::getVStep ( ) const
```

get the value set for the voltage step.

#### Returns

the value set for the voltage step in volts.

#### See also

[setVStep](#)

### 15.21.3.12 isAutoRange()

```
bool AisNormalPulseVoltammetryElement::isAutoRange ( ) const
```

tells whether the current range is set to auto-select or not.

#### Returns

true if the current range is set to auto-select and false if a rage has been selected.

### 15.21.3.13 isEndVoltageVsOCP()

```
bool AisNormalPulseVoltammetryElement::isEndVoltageVsOCP ( ) const
```

tells whether the end voltage is set with respect to the open circuit voltage or the reference terminal.

#### Returns

true if the end voltage is set with respect to the open-circuit voltage and false if set against the reference terminal.

#### Note

if nothing is set, the default is false.

**15.21.3.14 isStartVoltageVsOCP()**

```
bool AisNormalPulseVoltammetryElement::isStartVoltageVsOCP ( ) const
```

tells whether the start voltage is set against the open-circuit voltage or the reference terminal.

**Returns**

true if the start voltage is set against the open-circuit voltage and false if it is set against the reference terminal.

**Note**

if nothing is set, the default is false.

**See also**

[setStartVoltageVsOCP](#)

**15.21.3.15 setAlphaFactor()**

```
void AisNormalPulseVoltammetryElement::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.

The is an **optional** parameter. If nothing is set, the device will use the default value of 75.

**Parameters**

<i>alphaFactor</i>	the value for the alphaFactor ranges from 0 to 100.
--------------------	---

**15.21.3.16 setApproxMaxCurrent()**

```
void AisNormalPulseVoltammetryElement::setApproxMaxCurrent (
    double approxMaxCurrent )
```

set maximum current expected, for manual current range selection.

The is an **optional** parameter. If nothing is set, the device will auto-select the current range.

**Parameters**

<i>approxMaxCurrent</i>	the value for the maximum current expected in Amps.
-------------------------	---

### 15.21.3.17 setAutoRange()

```
void AisNormalPulseVoltammetryElement::setAutoRange ( )
```

set to auto-select the current range.

This option is set by default. There is no need to call this function to auto-select if the range was not manually set.

### 15.21.3.18 setEndVoltage()

```
void AisNormalPulseVoltammetryElement::setEndVoltage (
    double endVoltage )
```

set the ending potential value.

This is the value of the voltage at which the experiment will stop.

#### Parameters

<i>endVoltage</i>	the value to set for the ending potential in volts.
-------------------	---

### 15.21.3.19 setEndVoltageVsOCP()

```
void AisNormalPulseVoltammetryElement::setEndVoltageVsOCP (
    bool endVoltageVsOcp )
```

set whether to reference the end voltage against the open-circuit voltage or the reference terminal.

The reference terminal is for you to connect to any reference point you like. Connect it to the working electrode to reference ground.

#### Parameters

<i>endVoltageVsOcp</i>	true to set the end voltage to be referenced against the open-circuit voltage and false if set against the reference terminal.
------------------------	--

#### Note

by default, this is set to false.

### 15.21.3.20 setPulsePeriod()

```
void AisNormalPulseVoltammetryElement::setPulsePeriod (
    double pulsePeriod )
```

set the value for the pulse period.

The pulse period is the time spent between the starts of two consecutive pulses.

#### Parameters

<i>pulsePeriod</i>	the value to set for the pulse period in seconds.
--------------------	---

### 15.21.3.21 setPulseWidth()

```
void AisNormalPulseVoltammetryElement::setPulseWidth (
    double pulseWidth )
```

set the value in seconds for pulse width.

The pulse width is the value in seconds for the time spent at the same voltage set for the pulse height.

#### Parameters

<i>pulseWidth</i>	the value to set for the pulse width in seconds.
-------------------	--

### 15.21.3.22 setQuietTime()

```
void AisNormalPulseVoltammetryElement::setQuietTime (
    double quietTime )
```

Sets the quiet time duration.

#### Parameters

<i>quietTime</i>	The quiet time duration to set in seconds.
------------------	--

### 15.21.3.23 setQuietTimeSamplingInterval()

```
void AisNormalPulseVoltammetryElement::setQuietTimeSamplingInterval (
    double quietTimeSamplingInterval )
```

Sets the quiet time sampling interval.

## Parameters

<i>quietTimeSamplingInterval</i>	The quiet time sampling interval to set in seconds.
----------------------------------	---

**15.21.3.24 setStartVoltage()**

```
void AisNormalPulseVoltammetryElement::setStartVoltage (
    double startVoltage )
```

set the value for the start voltage.

## Parameters

<i>startVoltage</i>	the value of the start voltage in volts
---------------------	---

**15.21.3.25 setStartVoltageVsOCP()**

```
void AisNormalPulseVoltammetryElement::setStartVoltageVsOCP (
    bool startVoltageVsOCP )
```

set whether to reference the start voltage against the open-circuit voltage or the reference terminal.

The reference terminal is for you to connect to any reference point you like. Connect it to the working electrode to reference ground.

## Parameters

<i>startVoltageVsOCP</i>	true to if the start voltage is set to reference the open-circuit voltage and false if set against the reference terminal.
--------------------------	--

## Note

by default, this is set to false.

**15.21.3.26 setVStep()**

```
void AisNormalPulseVoltammetryElement::setVStep (
    double vStep )
```

set the value for the voltage step.

The voltage step is the voltage difference between the heights of two consecutive pulses.

## Parameters

<code>vStep</code>	the value for the voltage step in volts.
--------------------	--

The documentation for this class was generated from the following file:

- `AisNormalPulseVoltammetryElement.h`

## 15.22 AisOpenCircuitElement Class Reference

This experiment observes the **open circuit potential** of the working electrode for a specific period of time.

```
#include <AisOpenCircuitElement.h>
```

Inherits `AisAbstractElement`.

### Public Member Functions

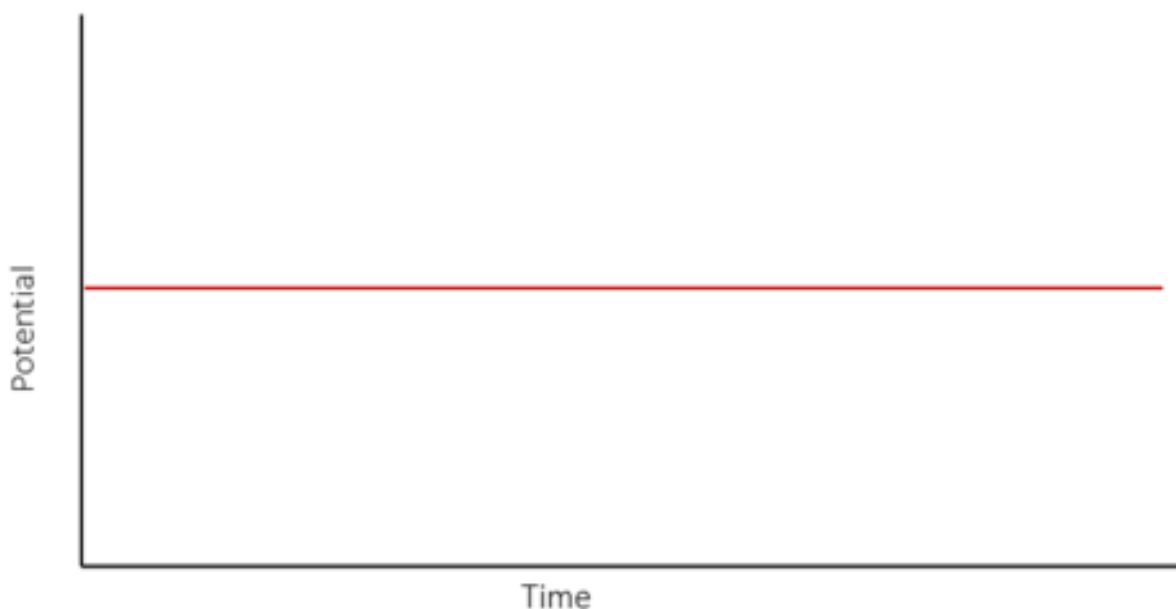
- `AisOpenCircuitElement` (double duration, double samplingInterval)  
*the open-circuit element constructor.*
- `AisOpenCircuitElement` (const `AisOpenCircuitElement` &)  
*copy constructor for the `AisOpenCircuitElement` object.*
- `AisOpenCircuitElement` & `operator=` (const `AisOpenCircuitElement` &)  
*overload equal to operator for the `AisOpenCircuitElement` object.*
- `QString getName ()` const override  
*get the name of the element.*
- `QStringList getCategory ()` const override  
*get a list of applicable categories of the element.*
- `double getSamplingInterval ()` const  
*get how frequently we are sampling the data.*
- `void setSamplingInterval` (double samplingInterval)  
*set how frequently we are sampling the data.*
- `double getMaxDuration ()` const  
*get the value set for the duration of the experiment.*
- `void setMaxDuration` (double maxDuration)  
*set the value set for the duration of the experiment.*
- `double getMaxVoltage ()` const  
*get the value set for the maximum voltage. The experiment will end when it reaches this value.*
- `void setMaxVoltage` (double maxVoltage)  
*set a maximum voltage to stop the experiment.*
- `double getMinVoltage ()` const  
*get the value set minimum for the voltage in volts.*
- `void setMinVoltage` (double minVoltage)  
*set a minimum voltage to stop the experiment.*
- `double getMindVdt ()` const  
*get the value set for the minimum voltage rate of change with respect to time (minimum dV/dt).*
- `void setMindVdt` (double mindVdt)



- set the minimum value for the voltage rate of change with respect to time (minimum  $dV/dt$ ).*
- bool `isAutoVoltageRange ()` const  
*tells whether the voltage range is set to auto-select or not.*
- void `setAutoVoltageRange ()`  
*set to auto-select the voltage range.*
- double `getApproxMaxVoltage ()` const  
*get the value set for the expected maximum voltage.*
- void `setApproxMaxVoltage (double approxMaxVoltage)`  
*set maximum voltage expected, for manual voltage range selection.*

### 15.22.1 Detailed Description

This experiment observes the **open circuit potential** of the working electrode for a specific period of time.



### 15.22.2 Constructor & Destructor Documentation

#### 15.22.2.1 AisOpenCircuitElement()

```
AisOpenCircuitElement::AisOpenCircuitElement (
    double duration,
    double samplingInterval ) [explicit]
```

the open-circuit element constructor.

**Parameters**

<i>duration</i>	the maximum duration for the experiment in seconds.
<i>samplingInterval</i>	the data sampling interval value in seconds.

**15.22.3 Member Function Documentation****15.22.3.1 getApproxMaxVoltage()**

```
double AisOpenCircuitElement::getApproxMaxVoltage ( ) const
```

get the value set for the expected maximum voltage.

**Returns**

the value set for the expected maximum Voltage in volt.

**Note**

if nothing was manually set, the device will auto-select the voltage range and the return value will be positive infinity.

**15.22.3.2 getCategory()**

```
QStringList AisOpenCircuitElement::getCategory ( ) const [override]
```

get a list of applicable categories of the element.

**Returns**

A list of applicable categories: ("Basic Experiments").

**15.22.3.3 getMaxDuration()**

```
double AisOpenCircuitElement::getMaxDuration ( ) const
```

get the value set for the duration of the experiment.

**Returns**

the value set for the duration of the experiment in seconds.

#### 15.22.3.4 getMaxVoltage()

```
double AisOpenCircuitElement::getMaxVoltage ( ) const
```

get the value set for the maximum voltage. The experiment will end when it reaches this value.

##### Returns

the value set for the maximum voltage.

##### Note

this is an optional parameter. If no value has been set, the default value is positive infinity.

#### 15.22.3.5 getMindVdt()

```
double AisOpenCircuitElement::getMindVdt ( ) const
```

get the value set for the minimum voltage rate of change with respect to time (minimum  $dV/dt$ ).

##### Returns

the value set for the minimum voltage rate of change with respect to time (minimum  $dV/dt$ ).

##### Note

this is an optional parameter. If no value has been set, the default value is zero

#### 15.22.3.6 getMinVoltage()

```
double AisOpenCircuitElement::getMinVoltage ( ) const
```

get the value set minimum for the voltage in volts.

##### Returns

the value set for the minimum voltage in volts.

##### Note

this is an optional parameter. If no value has been set, the default value is negative infinity.

### 15.22.3.7 getName()

```
QString AisOpenCircuitElement::getName ( ) const [override]
```

get the name of the element.

#### Returns

The name of the element: "Open Circuit Potential".

### 15.22.3.8 getSamplingInterval()

```
double AisOpenCircuitElement::getSamplingInterval ( ) const
```

get how frequently we are sampling the data.

#### Returns

the data sampling interval value in seconds.

### 15.22.3.9 isAutoVoltageRange()

```
bool AisOpenCircuitElement::isAutoVoltageRange ( ) const
```

tells whether the voltage range is set to auto-select or not.

#### Returns

true if the voltage range is set to auto-select and false if a range has been selected.

### 15.22.3.10 setApproxMaxVoltage()

```
void AisOpenCircuitElement::setApproxMaxVoltage (
    double approxMaxVoltage )
```

set maximum voltage expected, for manual voltage range selection.

The is an **optional** parameter. If nothing is set, the device will auto-select the voltage range.

#### Parameters

<i>approxMaxVoltage</i>	the value for the maximum current expected in V.
-------------------------	--

**15.22.3.11 setAutoVoltageRange()**

```
void AisOpenCircuitElement::setAutoVoltageRange ( )
```

set to auto-select the voltage range.

This option is set by default. There is no need to call this function to auto-select if the range was not manually set.

**15.22.3.12 setMaxDuration()**

```
void AisOpenCircuitElement::setMaxDuration (
    double maxDuration )
```

set the value set for the duration of the experiment.

**Parameters**

<i>maxDuration</i>	the value to set for the duration of the experiment in seconds.
--------------------	---

**15.22.3.13 setMaxVoltage()**

```
void AisOpenCircuitElement::setMaxVoltage (
    double maxVoltage )
```

set a maximum voltage to stop the experiment.

The is an **optional** condition. If nothing is set, then the experiment will not stop based on an upper-limit voltage value. If a maximum voltage is set, the experiment will continue to run as long as the measured voltage is below that value.

**Parameters**

<i>maxVoltage</i>	the maximum voltage value in volts at which the experiment will stop.
-------------------	---

**15.22.3.14 setMindVdt()**

```
void AisOpenCircuitElement::setMindVdt (
    double mindVdt )
```

set the minimum value for the voltage rate of change with respect to time (minimum dV/dt).

The is an **optional** condition. If nothing is set, then the experiment will not stop based on an lower-limit rate of change value. If a minimum value is set, the experiment will continue to run as long as the rage of change is above that value.

## Parameters

<i>mindVdt</i>	the minimum value for the voltage rate of change with respect to time (minimum dV/dt).
----------------	--

**15.22.3.15 setMinVoltage()**

```
void AisOpenCircuitElement::setMinVoltage (
    double minVoltage )
```

set a minimum voltage to stop the experiment.

The is an **optional** condition. If nothing is set, then the experiment will not stop based on an lower-limit voltage value. If a maximum voltage is set, the experiment will continue to run as long as the measured voltage is above that value.

## Parameters

<i>minVoltage</i>	the minimum voltage value in volts at which the experiment will stop.
-------------------	---

**15.22.3.16 setSamplingInterval()**

```
void AisOpenCircuitElement::setSamplingInterval (
    double samplingInterval )
```

set how frequently we are sampling the data.

## Parameters

<i>samplingInterval</i>	the data sampling interval value in seconds.
-------------------------	--

The documentation for this class was generated from the following file:

- AisOpenCircuitElement.h

**15.23 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>
```

Inherits AisAbstractElement.

## Public Member Functions

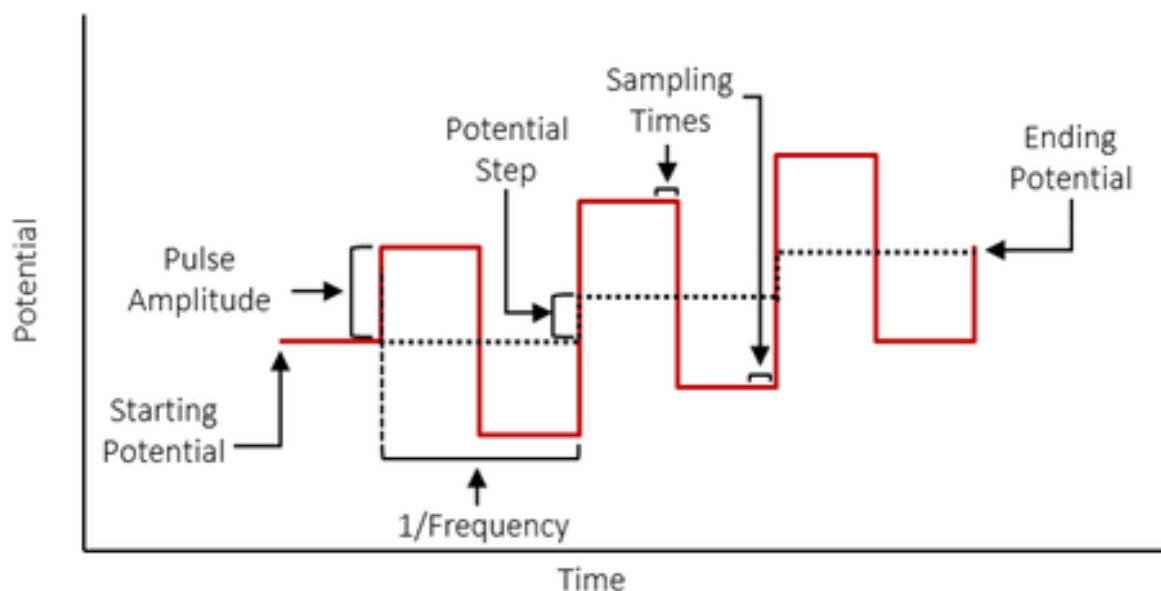
- [AisSquareWaveVoltammetryElement](#) (double startVoltage, double endVoltage, double voltageStep, double pulseAmp, double pulseFrequency)  
*the square wave element constructor*
- [AisSquareWaveVoltammetryElement](#) (const [AisSquareWaveVoltammetryElement](#) &)  
*copy constructor for the [AisSquareWaveVoltammetryElement](#) object.*
- [AisSquareWaveVoltammetryElement](#) & **operator=** (const [AisSquareWaveVoltammetryElement](#) &)  
*overload equal to operator for the [AisSquareWaveVoltammetryElement](#) object.*
- QString [getName](#) () const override  
*get the name of the element.*
- QStringList [getCategory](#) () const override  
*get a list of applicable categories of the element.*
- double [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 [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 ()`  
*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.*

### 15.23.1 Detailed Description

This experiment holds the working electrode at the **starting potential** during the **quiet time**. Then it applies a train of square pulses superimposed on a staircase waveform with a uniform **potential step** magnitude.

The potential continues to step until the **final potential** is reached. Each square pulse consists of a forward pulse and a reverse pulse of equal in **amplitude** but opposite in direction. **Frequency** is the inverse of the total duration of a square pulse. Current responses are sampled at two points, one at the end of the forward pulse (if) and another at the end of the reverse pulse (ir). The difference in current sampled at these two points is plotted against the potential of the corresponding staircase tread.



Advanced control of data output for pulse experiments can be performed using the class

See also

[AisDataManipulator](#)



## 15.23.2 Constructor & Destructor Documentation

### 15.23.2.1 AisSquareWaveVoltammetryElement()

```
AisSquareWaveVoltammetryElement::AisSquareWaveVoltammetryElement (
    double startVoltage,
    double endVoltage,
    double voltageStep,
    double pulseAmp,
    double pulseFrequency ) [explicit]
```

the square wave element constructor

#### Parameters

<i>startVoltage</i>	the value of the starting potential in volts
<i>endVoltage</i>	the value of the ending potential in volts
<i>voltageStep</i>	the value set for the voltage step in volts.
<i>pulseAmp</i>	the value for the pulse amplitude in volts.
<i>pulseFrequency</i>	the value for the pulse frequency in Hz.

## 15.23.3 Member Function Documentation

### 15.23.3.1 getAlphaFactor()

```
double AisSquareWaveVoltammetryElement::getAlphaFactor ( )
```

Get the value set for the alpha factor.

#### 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.

### 15.23.3.2 getApproxMaxCurrent()

```
double AisSquareWaveVoltammetryElement::getApproxMaxCurrent ( ) const
```

get the value set for the expected maximum current.

#### Returns

the value set for the expected maximum current in Amps.

#### Note

if nothing was manually set, the device will auto-select the current range and the return value will be positive infinity.

### 15.23.3.3 getCategory()

```
QStringList AisSquareWaveVoltammetryElement::getCategory ( ) const [override]
```

get a list of applicable categories of the element.

#### Returns

A list of applicable categories: ("Potentiostatic Control", "Pulse Voltammetry").

### 15.23.3.4 getEndVoltage()

```
double AisSquareWaveVoltammetryElement::getEndVoltage ( ) const
```

get the value set for the ending potential value.

This is the value of the voltage at which the experiment will stop.

#### Returns

the value set for the ending voltage in volts.

### 15.23.3.5 getName()

```
QString AisSquareWaveVoltammetryElement::getName ( ) const [override]
```

get the name of the element.

#### Returns

The name of the element: "Square Wave Potential Voltammetry".

### 15.23.3.6 getPulseAmp()

```
double AisSquareWaveVoltammetryElement::getPulseAmp ( ) const
```

get the value set for the pulse amplitude.

#### Returns

the value set for the pulse amplitude in volts.

#### See also

[setPulseAmp](#)

### 15.23.3.7 getPulseFreq()

```
double AisSquareWaveVoltammetryElement::getPulseFreq ( ) const
```

get the value set for the pulse frequency.

#### Returns

the value set for the frequency in Hz.

### 15.23.3.8 getQuietTime()

```
double AisSquareWaveVoltammetryElement::getQuietTime ( ) const
```

Gets the quiet time duration.

#### Returns

The quiet time duration in seconds.

### 15.23.3.9 getQuietTimeSamplingInterval()

```
double AisSquareWaveVoltammetryElement::getQuietTimeSamplingInterval ( ) const
```

gets the quiet time sampling interval.

#### Returns

samplingInterval The quiet time sampling interval to set in seconds.

#### 15.23.3.10 `getStartVoltage()`

```
double AisSquareWaveVoltammetryElement::getStartVoltage ( ) const
```

get the value set for the start voltage.

##### Returns

the value of the start voltage in volts.

#### 15.23.3.11 `getVStep()`

```
double AisSquareWaveVoltammetryElement::getVStep ( ) const
```

get the value set for the voltage step.

##### Returns

the value set for the voltage step in volts.

##### See also

[setVStep](#)

#### 15.23.3.12 `isAutoRange()`

```
bool AisSquareWaveVoltammetryElement::isAutoRange ( ) const
```

tells whether the current range is set to auto-select or not.

##### Returns

true if the current range is set to auto-select and false if a range has been selected.

#### 15.23.3.13 `isEndVoltageVsOCP()`

```
bool AisSquareWaveVoltammetryElement::isEndVoltageVsOCP ( ) const
```

tells whether the end voltage is set with respect to the open circuit voltage or the reference terminal.

##### Returns

true if the end voltage is set with respect to the open-circuit voltage and false if set against the reference terminal.

##### Note

if nothing is set, the default is false.

#### 15.23.3.14 isStartVoltageVsOCP()

```
bool AisSquareWaveVoltammetryElement::isStartVoltageVsOCP ( ) const
```

tells whether the start voltage is set against the open-circuit voltage or the reference terminal.

##### Returns

true if the start voltage is set against the open-circuit voltage and false if it is set against the reference terminal.

##### Note

if nothing is set, the default is false.

##### See also

[setStartVoltageVsOCP](#)

#### 15.23.3.15 setAlphaFactor()

```
void AisSquareWaveVoltammetryElement::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.

The is an **optional** parameter. If nothing is set, the device will use the default value of 75.

##### Parameters

<i>alphaFactor</i>	the value for the alphaFactor ranges from 0 to 100.
--------------------	---

#### 15.23.3.16 setApproxMaxCurrent()

```
void AisSquareWaveVoltammetryElement::setApproxMaxCurrent (
    double approxMaxCurrent )
```

set maximum current expected, for manual current range selection.

The is an **optional** parameter. If nothing is set, the device will auto-select the current range.

##### Parameters

<i>approxMaxCurrent</i>	the value for the maximum current expected in Amps.
-------------------------	---

### 15.23.3.17 setAutoRange()

```
void AisSquareWaveVoltammetryElement::setAutoRange ( )
```

set to auto-select the current range.

This option is set by default. There is no need to call this function to auto-select if the range was not manually set.

### 15.23.3.18 setEndVoltage()

```
void AisSquareWaveVoltammetryElement::setEndVoltage (
    double endVoltage )
```

set the ending potential value.

This is the value of the voltage at which the experiment will stop.

#### Parameters

<i>endVoltage</i>	the value to set for the ending potential in volts.
-------------------	---

### 15.23.3.19 setEndVoltageVsOCP()

```
void AisSquareWaveVoltammetryElement::setEndVoltageVsOCP (
    bool endVoltageVsOcp )
```

set whether to reference the end voltage against the open-circuit voltage or the reference terminal.

The reference terminal is for you to connect to any reference point you like. Connect it to the working electrode to reference ground.

#### Parameters

<i>endVoltageVsOcp</i>	true to set the end voltage to be referenced against the open-circuit voltage and false if set against the reference terminal.
------------------------	--

#### Note

by default, this is set to false.

### 15.23.3.20 setPulseAmp()

```
void AisSquareWaveVoltammetryElement::setPulseAmp (
    double pulseAmp )
```

set the value for the pulse amplitude.

The voltage pulse goes up in height by the given amplitude in addition to the starting potential (of the previous pulse). It then goes back down twice the amplitude to end up one amplitude below the starting potential (of the previous pulse).

#### Parameters

<i>pulseAmp</i>	the value to set for the pulse amplitude in volts.
-----------------	--

### 15.23.3.21 setPulseFreq()

```
void AisSquareWaveVoltammetryElement::setPulseFreq (
    double pulseFreq )
```

set the value for the pulse frequency.

#### Parameters

<i>pulseFreq</i>	the value to set for the pulse frequency in Hz.
------------------	---

### 15.23.3.22 setQuietTime()

```
void AisSquareWaveVoltammetryElement::setQuietTime (
    double quietTime )
```

Sets the quiet time duration.

#### Parameters

<i>quietTime</i>	The quiet time duration to set in seconds.
------------------	--

### 15.23.3.23 setQuietTimeSamplingInterval()

```
void AisSquareWaveVoltammetryElement::setQuietTimeSamplingInterval (
    double quietTimeSamplingInterval )
```

Sets the quiet time sampling interval.

## Parameters

<i>quietTimeSamplingInterval</i>	The quiet time sampling interval to set in seconds.
----------------------------------	---

**15.23.3.24 setStartVoltage()**

```
void AisSquareWaveVoltammetryElement::setStartVoltage (
    double startVoltage )
```

set the value for the start voltage.

## Parameters

<i>startVoltage</i>	the value of the start voltage in volts
---------------------	---

**15.23.3.25 setStartVoltageVsOCP()**

```
void AisSquareWaveVoltammetryElement::setStartVoltageVsOCP (
    bool startVoltageVsOcp )
```

set whether to reference the start voltage against the open-circuit voltage or the reference terminal.

The reference terminal is for you to connect to any reference point you like. Connect it to the working electrode to reference ground.

## Parameters

<i>startVoltageVsOcp</i>	true to if the start voltage is set to reference the open-circuit voltage and false if set against the reference terminal.
--------------------------	--

## Note

by default, this is set to false.

**15.23.3.26 setVStep()**

```
void AisSquareWaveVoltammetryElement::setVStep (
    double vStep )
```

set the value for the voltage step.

The voltage step is added to the value of the starting potential of the previous pulse to start the new pulse.



## Parameters

<code>vStep</code>	the value for the voltage step in volts.
--------------------	--

The documentation for this class was generated from the following file:

- `AisSquareWaveVoltammetryElement.h`

## 15.24 AisStaircasePotentialVoltammetryElement Class Reference

[AisStaircasePotentialVoltammetryElement](#) class represents an element for staircase potential voltammetry experiments. It inherits from [AisAbstractElement](#).

```
#include <AisStaircasePotentialVoltammetryElement.h>
```

Inherits [AisAbstractElement](#).

### Public Member Functions

- [AisStaircasePotentialVoltammetryElement](#) (double startVoltage, double firstVoltageLimit, double secondVoltageLimit, 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](#) &other)  
*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)

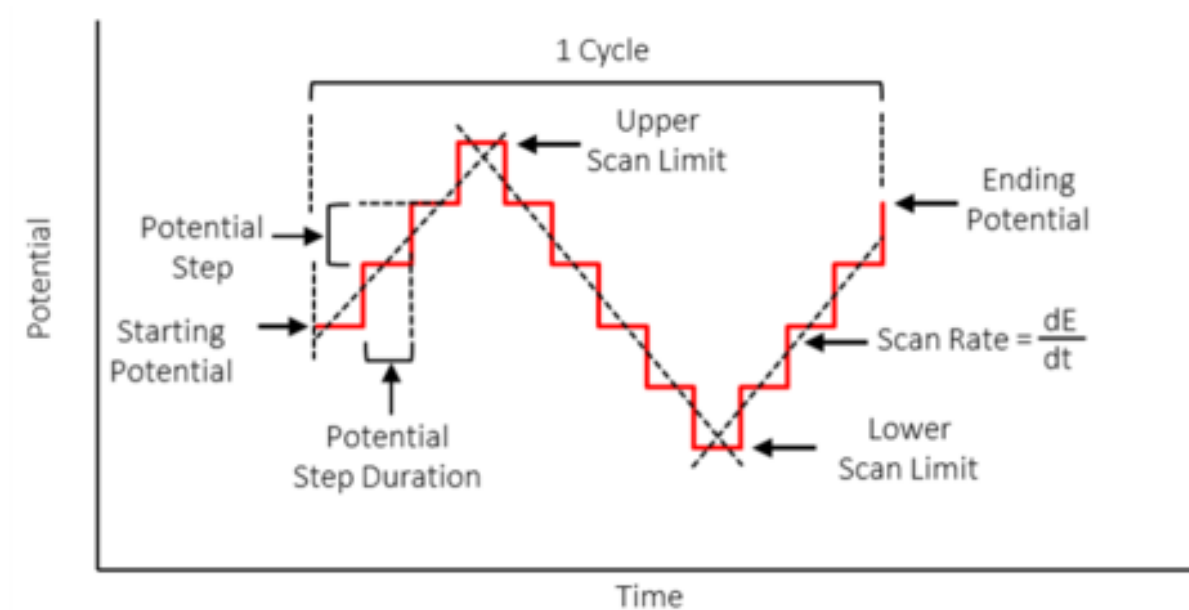
*Sets the approximate maximum current.*
- double `getNumberOfCycles` ()

*get the value set for the number of cycles*
- void `setNumberOfCycles` (int cycles)

*set the number of cycles to oscillate between the first voltage-limit and the second voltage-limit.*

### 15.24.1 Detailed Description

[AisStaircasePotentialVoltammetryElement](#) class represents an element for staircase potential voltammetry experiments. It inherits from [AisAbstractElement](#).



### 15.24.2 Constructor & Destructor Documentation

#### 15.24.2.1 AisStaircasePotentialVoltammetryElement() [1/2]

```
AisStaircasePotentialVoltammetryElement::AisStaircasePotentialVoltammetryElement (
    double startVoltage,
    double firstVoltageLimit,
    double secondVoltageLimit,
    double endVoltage,
    double stepSize,
    double stepDuration,
    double samplingInterval )
```

Constructs an [AisStaircasePotentialVoltammetryElement](#) with specified parameters.

#### Parameters

<i>startVoltage</i>	The starting voltage in volts.
<i>firstVoltageLimit</i>	The first voltage limit in volts.
<i>secondVoltageLimit</i>	The second voltage limit in volts.
<i>endVoltage</i>	The ending voltage in volts.
<i>stepSize</i>	The potential step size in volts.
<i>stepDuration</i>	The potential step duration in seconds.
<i>samplingInterval</i>	The potential sampling interval in seconds.

### 15.24.2.2 AisStaircasePotentialVoltammetryElement() [2/2]

```
AisStaircasePotentialVoltammetryElement::AisStaircasePotentialVoltammetryElement (
    const AisStaircasePotentialVoltammetryElement & other )
```

Copy constructor for [AisStaircasePotentialVoltammetryElement](#).

#### Parameters

<i>other</i>	The <a href="#">AisStaircasePotentialVoltammetryElement</a> to copy.
--------------	--

## 15.24.3 Member Function Documentation

### 15.24.3.1 getApproxMaxCurrent()

```
double AisStaircasePotentialVoltammetryElement::getApproxMaxCurrent ( ) const
```

Gets the approximate maximum current.

#### Returns

The approximate maximum current.

### 15.24.3.2 getCategory()

```
QStringList AisStaircasePotentialVoltammetryElement::getCategory ( ) const [override]
```

Gets the category of the element.

#### Returns

The category of the element.

### 15.24.3.3 getEndVoltage()

```
double AisStaircasePotentialVoltammetryElement::getEndVoltage ( ) const
```

Gets the ending voltage.

#### Returns

The ending voltage in volts.

#### 15.24.3.4 getFirstVoltageLimit()

```
double AisStaircasePotentialVoltammetryElement::getFirstVoltageLimit ( ) const
```

Gets the first voltage limit.

##### Returns

The first voltage limit in volts.

#### 15.24.3.5 getName()

```
QString AisStaircasePotentialVoltammetryElement::getName ( ) const [override]
```

Gets the name of the element.

##### Returns

The name of the element.

#### 15.24.3.6 getNumberOfCycles()

```
double AisStaircasePotentialVoltammetryElement::getNumberOfCycles ( )
```

get the value set for the number of cycles

##### Returns

the number of cycles set.

#### 15.24.3.7 getQuietTime()

```
double AisStaircasePotentialVoltammetryElement::getQuietTime ( ) const
```

Gets the quiet time duration.

##### Returns

The quiet time duration in seconds.

#### 15.24.3.8 getQuietTimeSamplingInterval()

```
double AisStaircasePotentialVoltammetryElement::getQuietTimeSamplingInterval ( ) const
```

gets the quiet time sampling interval.

##### Returns

samplingInterval The quiet time sampling interval to set in seconds.

#### 15.24.3.9 getSamplingInterval()

```
double AisStaircasePotentialVoltammetryElement::getSamplingInterval ( ) const
```

Gets the potential sampling interval.

##### Returns

The potential sampling interval in seconds.

#### 15.24.3.10 getSecondVoltageLimit()

```
double AisStaircasePotentialVoltammetryElement::getSecondVoltageLimit ( ) const
```

Gets the second voltage limit.

##### Returns

The second voltage limit in volts.

#### 15.24.3.11 getStartVoltage()

```
double AisStaircasePotentialVoltammetryElement::getStartVoltage ( ) const
```

Gets the starting voltage.

##### Returns

The starting voltage in volts.

**15.24.3.12 getStepDuration()**

```
double AisStaircasePotentialVoltammetryElement::getStepDuration ( ) const
```

Gets the potential step duration.

**Returns**

The potential step duration in seconds.

**15.24.3.13 getStepSize()**

```
double AisStaircasePotentialVoltammetryElement::getStepSize ( ) const
```

Gets the potential step size.

**Returns**

The potential step size in volts.

**15.24.3.14 isAutorange()**

```
bool AisStaircasePotentialVoltammetryElement::isAutorange ( ) const
```

Checks if the experiment should autorange the current.

**Returns**

True if autorange is enabled, false otherwise.

**15.24.3.15 isEndVoltageVsOCP()**

```
bool AisStaircasePotentialVoltammetryElement::isEndVoltageVsOCP ( ) const
```

Checks if the ending voltage is with respect to the open circuit mode.

**Returns**

True if the ending voltage is with respect to the open circuit mode, false otherwise.

#### 15.24.3.16 isFirstVoltageLimitVsOCP()

```
bool AisStaircasePotentialVoltammetryElement::isFirstVoltageLimitVsOCP ( ) const
```

Checks if the first voltage limit is with respect to the open circuit mode.

##### Returns

True if the first voltage limit is with respect to the open circuit mode, false otherwise.

#### 15.24.3.17 isSecondVoltageLimitVsOCP()

```
bool AisStaircasePotentialVoltammetryElement::isSecondVoltageLimitVsOCP ( ) const
```

Checks if the second voltage limit is with respect to the open circuit mode.

##### Returns

True if the second voltage limit is with respect to the open circuit mode, false otherwise.

#### 15.24.3.18 isStartVoltageVsOCP()

```
bool AisStaircasePotentialVoltammetryElement::isStartVoltageVsOCP ( ) const
```

Checks if the starting voltage is with respect to the open circuit mode.

##### Returns

True if the starting voltage is with respect to the open circuit mode, false otherwise.

#### 15.24.3.19 operator=()

```
AisStaircasePotentialVoltammetryElement & AisStaircasePotentialVoltammetryElement::operator= (
    const AisStaircasePotentialVoltammetryElement & other )
```

Assignment operator for [AisStaircasePotentialVoltammetryElement](#).

##### Parameters

<i>other</i>	The <a href="#">AisStaircasePotentialVoltammetryElement</a> to assign.
--------------	--



### Returns

Reference to this [AisStaircasePotentialVoltammetryElement](#).

#### 15.24.3.20 setApproxMaxCurrent()

```
void AisStaircasePotentialVoltammetryElement::setApproxMaxCurrent (
    double approxMaxCurrent )
```

Sets the approximate maximum current.

### Parameters

<i>approxMaxCurrent</i>	The approximate maximum current to set.
-------------------------	---

#### 15.24.3.21 setEndVoltage()

```
void AisStaircasePotentialVoltammetryElement::setEndVoltage (
    double endVoltage )
```

Sets the ending voltage.

### Parameters

<i>endVoltage</i>	The ending voltage to set in volts.
-------------------	-------------------------------------

#### 15.24.3.22 setEndVoltageVsOCP()

```
void AisStaircasePotentialVoltammetryElement::setEndVoltageVsOCP (
    bool endVsOCP )
```

Sets whether the ending voltage is with respect to the open circuit mode.

### Parameters

<i>endVsOCP</i>	True to set the ending voltage with respect to the open circuit mode, false otherwise.
-----------------	--

#### 15.24.3.23 setFirstVoltageLimit()

```
void AisStaircasePotentialVoltammetryElement::setFirstVoltageLimit (
```

```
double firstVoltageLimit )
```

Sets the first voltage limit.

#### Parameters

<i>firstVoltageLimit</i>	The first voltage limit to set in volts.
--------------------------	--

#### 15.24.3.24 setFirstVoltageLimitVsOCP()

```
void AisStaircasePotentialVoltammetryElement::setFirstVoltageLimitVsOCP (
    bool firstVoltageLimitVsOCP )
```

Sets whether the first voltage limit is with respect to the open circuit mode.

#### Parameters

<i>firstVoltageLimitVsOCP</i>	True to set the first voltage limit with respect to the open circuit mode, false otherwise.
-------------------------------	---

#### 15.24.3.25 setNumberOfCycles()

```
void AisStaircasePotentialVoltammetryElement::setNumberOfCycles (
    int cycles )
```

set the number of cycles to oscillate between the first voltage-limit and the second voltage-limit.

#### Parameters

<i>cycles</i>	the number of cycles to set
---------------	-----------------------------

#### 15.24.3.26 setQuietTime()

```
void AisStaircasePotentialVoltammetryElement::setQuietTime (
    double quietTime )
```

Sets the quiet time duration.

#### Parameters

<i>quietTime</i>	The quiet time duration to set in seconds.
------------------	--

### 15.24.3.27 setQuietTimeSamplingInterval()

```
void AisStaircasePotentialVoltammetryElement::setQuietTimeSamplingInterval (
    double quietTimeSamplingInterval )
```

Sets the quiet time sampling interval.

#### Parameters

<i>quietTimeSamplingInterval</i>	The quiet time sampling interval to set in seconds.
----------------------------------	---

### 15.24.3.28 setSamplingInterval()

```
void AisStaircasePotentialVoltammetryElement::setSamplingInterval (
    double samplingInterval )
```

Sets the potential sampling interval.

#### Parameters

<i>samplingInterval</i>	The potential sampling interval to set in seconds.
-------------------------	--

### 15.24.3.29 setSecondVoltageLimit()

```
void AisStaircasePotentialVoltammetryElement::setSecondVoltageLimit (
    double secondVoltageLimit )
```

Sets the second voltage limit.

#### Parameters

<i>secondVoltageLimit</i>	The second voltage limit to set in volts.
---------------------------	---

### 15.24.3.30 setSecondVoltageLimitVsOCP()

```
void AisStaircasePotentialVoltammetryElement::setSecondVoltageLimitVsOCP (
    bool secondVoltageLimitVsOCP )
```

Sets whether the second voltage limit is with respect to the open circuit mode.

## Parameters

<i>secondVoltageLimitVsOCP</i>	True to set the second voltage limit with respect to the open circuit mode, false otherwise.
--------------------------------	--

**15.24.3.31 setStartVoltage()**

```
void AisStaircasePotentialVoltammetryElement::setStartVoltage (
    double startVoltage )
```

Sets the starting voltage.

## Parameters

<i>startVoltage</i>	The starting voltage to set in volts.
---------------------	---------------------------------------

**15.24.3.32 setStartVoltageVsOCP()**

```
void AisStaircasePotentialVoltammetryElement::setStartVoltageVsOCP (
    bool startVsOCP )
```

Sets whether the starting voltage is with respect to the open circuit mode.

## Parameters

<i>startVsOCP</i>	True to set the starting voltage with respect to the open circuit mode, false otherwise.
-------------------	--

**15.24.3.33 setStepDuration()**

```
void AisStaircasePotentialVoltammetryElement::setStepDuration (
    double stepDuration )
```

Sets the potential step duration.

## Parameters

<i>stepDuration</i>	The potential step duration to set in seconds.
---------------------	--

**15.24.3.34 setStepSize()**

```
void AisStaircasePotentialVoltammetryElement::setStepSize (
    double stepSize )
```

Sets the potential step size.

**Parameters**

<i>stepSize</i>	The potential step size to set in volts.
-----------------	--

The documentation for this class was generated from the following file:

- AisStaircasePotentialVoltammetryElement.h

**15.25 AisSteppedCurrentElement Class Reference**

A class representing an experiment to apply the stepped current.

```
#include <AisSteppedCurrentElement.h>
```

Inherits AisAbstractElement.

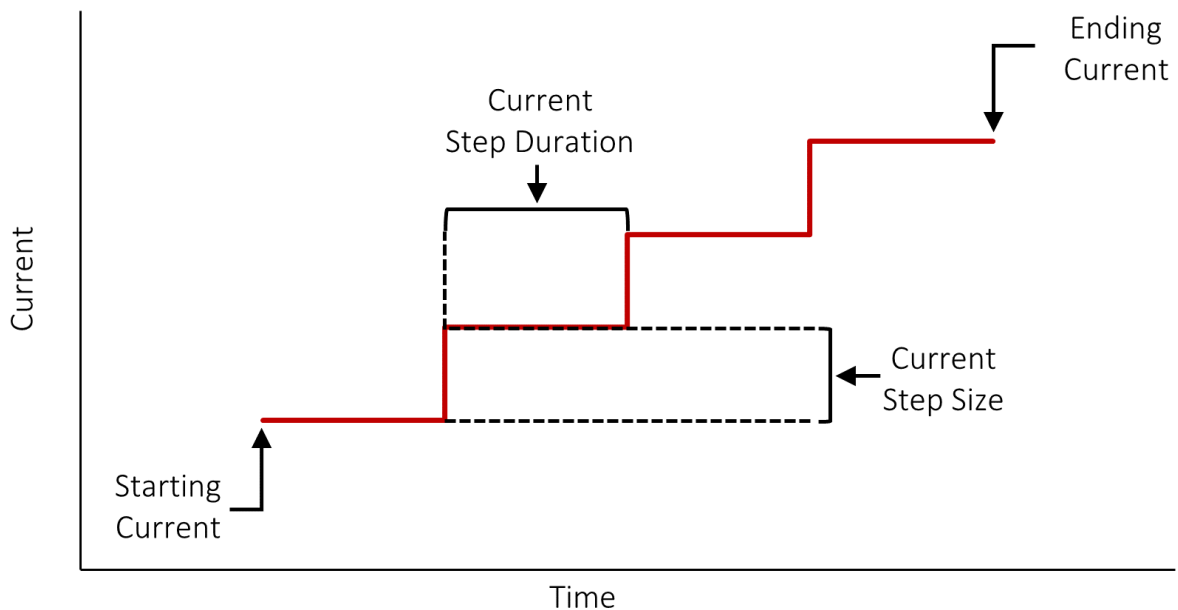
**Public Member Functions**

- [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.*
- double [getStepSize](#) () const  
*Gets the size of each current step in the stepped experiment.*
- double [getStartCurrent](#) () const  
*Gets the starting current value for the stepped experiment.*
- double [getStepDuration](#) () const

- Gets the duration of each current step in the stepped experiment.*
- void [setEndCurrent](#) (double iEnd)  
*Sets the ending current value for the stepped experiment.*
- void [setStepSize](#) (double iStep)  
*Sets the size of each current step in the stepped experiment.*
- void [setStartCurrent](#) (double iStart)  
*Sets the starting current value for 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.*

### 15.25.1 Detailed Description

A class representing an experiment to apply the stepped current.



### 15.25.2 Constructor & Destructor Documentation

### 15.25.2.1 AisSteppedCurrentElement()

```
AisSteppedCurrentElement::AisSteppedCurrentElement (
    double startCurrent,
    double endCurrent,
    double stepSize,
    double stepDuration,
    double samplingInterval ) [explicit]
```

Constructs a Stepped Current element.

This constructor initializes the Stepped Current element with the specified parameters.

#### Parameters

<i>startCurrent</i>	The initial current value in amperes.
<i>endCurrent</i>	The final current value in amperes.
<i>stepSize</i>	The size of each current step in amperes.
<i>stepDuration</i>	The duration of each current step in seconds.
<i>samplingInterval</i>	The data sampling interval value in seconds.

## 15.25.3 Member Function Documentation

### 15.25.3.1 getApproxMaxVoltage()

```
double AisSteppedCurrentElement::getApproxMaxVoltage ( ) const
```

get the value set for the expected maximum voltage.

#### Returns

the value set for the expected maximum Voltage in volt.

#### Note

if nothing was manually set, the device will auto-select the voltage range and the return value will be positive infinity.

### 15.25.3.2 getApproxMinVoltage()

```
double AisSteppedCurrentElement::getApproxMinVoltage ( ) const
```

get the value set for the expected minimum voltage.

#### Returns

the value set for the expected maximum Voltage in volt.

#### Note

if nothing was manually set, the device will auto-select the voltage range and the return value will be positive infinity.

### 15.25.3.3 getCategory()

```
QStringList AisSteppedCurrentElement::getCategory ( ) const [override]
```

get a list of applicable categories of the element.

#### Returns

A list of applicable categories: ("Galvanostatic Control").

### 15.25.3.4 getEndCurrent()

```
double AisSteppedCurrentElement::getEndCurrent ( ) const
```

Gets the ending current value for the stepped experiment.

#### Returns

The ending current value in amperes.

### 15.25.3.5 getName()

```
QString AisSteppedCurrentElement::getName ( ) const [override]
```

get the name of the element.

#### Returns

The name of the element: "SteppedCurrent".

### 15.25.3.6 getSamplingInterval()

```
double AisSteppedCurrentElement::getSamplingInterval ( ) const
```

get how frequently we are sampling the data.

#### Returns

the data sampling interval value in seconds.



### 15.25.3.7 getStartCurrent()

```
double AisSteppedCurrentElement::getStartCurrent ( ) const
```

Gets the starting current value for the stepped experiment.

#### Returns

The starting current value in amperes.

### 15.25.3.8 getStepDuration()

```
double AisSteppedCurrentElement::getStepDuration ( ) const
```

Gets the duration of each current step in the stepped experiment.

#### Returns

The duration of each current step in seconds.

### 15.25.3.9 getStepSize()

```
double AisSteppedCurrentElement::getStepSize ( ) const
```

Gets the size of each current step in the stepped experiment.

#### Returns

The size of each current step in amperes.

### 15.25.3.10 setApproxMaxVoltage()

```
void AisSteppedCurrentElement::setApproxMaxVoltage (
    double approxMaxVoltage )
```

set maximum voltage expected, for manual voltage range selection.

This is an **optional** parameter. If nothing is set, the device will auto-select the voltage range.

#### Parameters

<i>approxMaxVoltage</i>	the value for the maximum current expected in V.
-------------------------	--

#### 15.25.3.11 setApproxMinVoltage()

```
void AisSteppedCurrentElement::setApproxMinVoltage (
    double approxMinVoltage )
```

set minimum voltage expected, for manual voltage range selection.

The is an **optional** parameter. If nothing is set, the device will auto-select the voltage range.

##### Parameters

<i>approxMaxVoltage</i>	the value for the maximum current expected in V.
-------------------------	--

#### 15.25.3.12 setEndCurrent()

```
void AisSteppedCurrentElement::setEndCurrent (
    double iEnd )
```

Sets the ending current value for the stepped experiment.

##### Parameters

<i>iEnd</i>	The ending current value in amperes.
-------------	--------------------------------------

#### 15.25.3.13 setSamplingInterval()

```
void AisSteppedCurrentElement::setSamplingInterval (
    double samplingInterval )
```

set how frequently we are sampling the data.

##### Parameters

<i>samplingInterval</i>	the data sampling interval value in seconds.
-------------------------	--

#### 15.25.3.14 setStartCurrent()

```
void AisSteppedCurrentElement::setStartCurrent (
    double iStart )
```

Sets the starting current value for the stepped experiment.

## Parameters

<i>iStart</i>	The starting current value in amperes.
---------------	--

**15.25.3.15 setStepDuration()**

```
void AisSteppedCurrentElement::setStepDuration (
    double tStep )
```

Sets the duration of each current step in the stepped experiment.

## Parameters

<i>tStep</i>	The duration of each current step in seconds.
--------------	---

**15.25.3.16 setStepSize()**

```
void AisSteppedCurrentElement::setStepSize (
    double iStep )
```

Sets the size of each current step in the stepped experiment.

## Parameters

<i>iStep</i>	The size of each current step in amperes.
--------------	---

The documentation for this class was generated from the following file:

- AisSteppedCurrentElement.h

**15.26 AisSteppedVoltageElement Class Reference**

A class representing an experiment to apply the stepped voltage.

```
#include <AisSteppedVoltageElement.h>
```

Inherits AisAbstractElement.

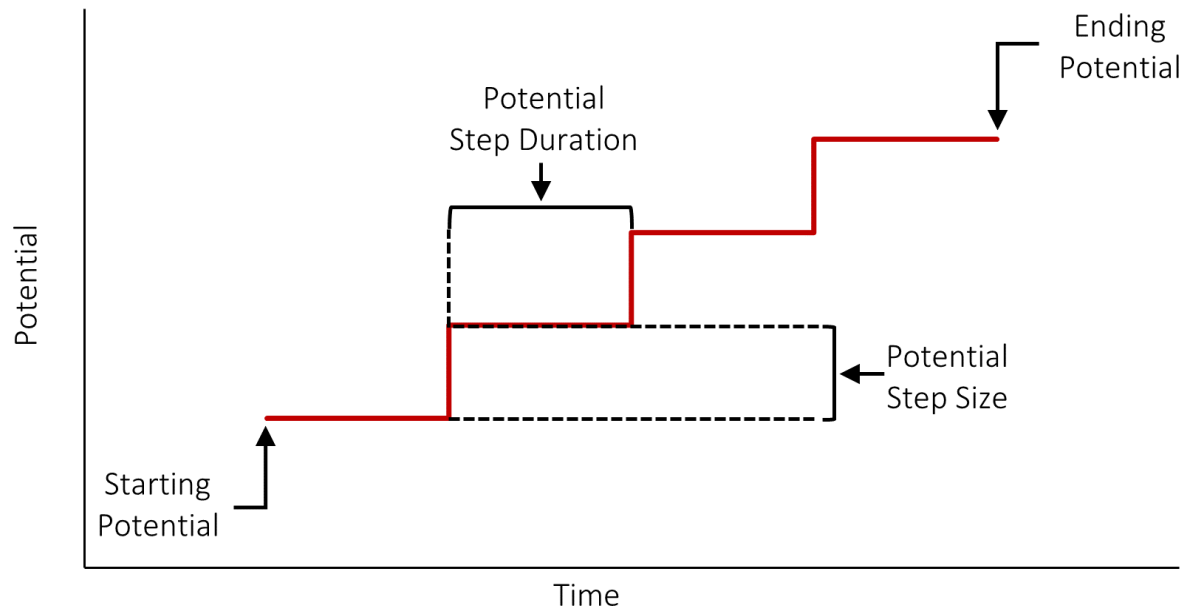
## Public Member Functions

- [AisSteppedVoltageElement](#) (double startVoltage, double endVoltage, double voltageStep, double voltageStepDuration, 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.*
- double [getEndVoltage](#) () const  
*Get the ending voltage for the experiment.*
- double [getStepSize](#) () const  
*Get the voltage step for each iteration.*
- double [getStepDuration](#) () const  
*Get the time step for each iteration.*
- double [getSamplingInterval](#) () const  
*Get the data sampling interval.*
- double [getApproxMaxCurrent](#) () const  
*Get the approximate maximum current.*
- bool [isStartVoltageVsOCP](#) () const  
*Check if the experiment starts with the open circuit potential.*
- bool [isEndVoltageVsOCP](#) () const  
*Check if the experiment ends with the open circuit potential.*
- bool [isAutoRange](#) () const  
*Check if current autoranging is enabled.*
- void [setStartVoltage](#) (double vStart)  
*Set the starting voltage for the experiment.*
- void [setEndVoltage](#) (double vEnd)  
*Set the ending voltage for the experiment.*
- void [setStepSize](#) (double stepsize)  
*Set the voltage step for each iteration.*
- void [setStepDuration](#) (double duration)  
*Set the time step for each iteration.*
- void [setSamplingInterval](#) (double samplingInterval)  
*Set the data sampling interval.*
- void [setApproxMaxCurrent](#) (double approxMaxCurrent)  
*Set the approximate maximum current.*
- void [setStartVoltageVsOCP](#) (bool startVsOCP)  
*Set whether the experiment starts with the open circuit potential.*
- void [setEndVoltageVsOCP](#) (bool endVsOCP)  
*Set whether the experiment ends with the open circuit potential.*
- void [setCurrentAutorange](#) ()  
*Enable current autoranging for the experiment.*

### 15.26.1 Detailed Description

A class representing an experiment to apply the stepped voltage.

This class inherits from `AisAbstractElement` and is designed for Stepped Voltage experiments.



### 15.26.2 Constructor & Destructor Documentation

#### 15.26.2.1 `AisSteppedVoltageElement()` [1/2]

```
AisSteppedVoltageElement::AisSteppedVoltageElement (
    double startVoltage,
    double endVoltage,
    double voltageStep,
    double voltageStepDuration,
    double samplingInterval ) [explicit]
```

Constructor for the [AisSteppedVoltageElement](#) element.

This constructor initializes the [AisSteppedVoltageElement](#) element with the specified parameters.

#### Parameters

<i>startVoltage</i>	The initial voltage value in volts.
<i>endVoltage</i>	The final voltage value in volts.
<i>voltageStep</i>	The size of each voltage step in volts.
<i>voltageStepDuration</i>	The duration of each voltage step in seconds.
<i>samplingInterval</i>	The data sampling interval value in seconds.

### 15.26.2.2 AisSteppedVoltageElement() [2/2]

```
AisSteppedVoltageElement::AisSteppedVoltageElement (  
    const AisSteppedVoltageElement & other ) [explicit]
```

Copy constructor for the [AisSteppedVoltageElement](#) object.

#### Parameters

<i>other</i>	The <a href="#">AisSteppedVoltageElement</a> object to be copied.
--------------	---

## 15.26.3 Member Function Documentation

### 15.26.3.1 getApproxMaxCurrent()

```
double AisSteppedVoltageElement::getApproxMaxCurrent ( ) const
```

Get the approximate maximum current.

#### Returns

The approximate maximum current in Amps.

### 15.26.3.2 getCategory()

```
QStringList AisSteppedVoltageElement::getCategory ( ) const [override]
```

Get a list of applicable categories of the element.

#### Returns

A list of applicable categories: ("Potentiostatic Control").

### 15.26.3.3 getEndVoltage()

```
double AisSteppedVoltageElement::getEndVoltage ( ) const
```

Get the ending voltage for the experiment.

#### Returns

The ending voltage in volts.

#### 15.26.3.4 getName()

```
QString AisSteppedVoltageElement::getName ( ) const [override]
```

Get the name of the element.

##### Returns

The name of the element: "Stepped Voltage".

#### 15.26.3.5 getSamplingInterval()

```
double AisSteppedVoltageElement::getSamplingInterval ( ) const
```

Get the data sampling interval.

##### Returns

The data sampling interval in seconds.

#### 15.26.3.6 getStartVoltage()

```
double AisSteppedVoltageElement::getStartVoltage ( ) const
```

Get the starting voltage for the experiment.

##### Returns

The starting voltage in volts.

#### 15.26.3.7 getStepDuration()

```
double AisSteppedVoltageElement::getStepDuration ( ) const
```

Get the time step for each iteration.

##### Returns

The time step in seconds.



### 15.26.3.8 getStepSize()

```
double AisSteppedVoltageElement::getStepSize ( ) const
```

Get the voltage step for each iteration.

#### Returns

The voltage step in volts.

### 15.26.3.9 isAutoRange()

```
bool AisSteppedVoltageElement::isAutoRange ( ) const
```

Check if current autoranging is enabled.

#### Returns

True if current autoranging is enabled, false otherwise.

### 15.26.3.10 isEndVoltageVsOCP()

```
bool AisSteppedVoltageElement::isEndVoltageVsOCP ( ) const
```

Check if the experiment ends with the open circuit potential.

#### Returns

True if the experiment ends with open circuit potential, false otherwise.

### 15.26.3.11 isStartVoltageVsOCP()

```
bool AisSteppedVoltageElement::isStartVoltageVsOCP ( ) const
```

Check if the experiment starts with the open circuit potential.

#### Returns

True if the experiment starts with open circuit potential, false otherwise.

### 15.26.3.12 operator=()

```
AisSteppedVoltageElement & AisSteppedVoltageElement::operator= (
    const AisSteppedVoltageElement & other )
```

Overloaded assignment operator for the [AisSteppedVoltageElement](#) object.

## Parameters

<i>other</i>	The <a href="#">AisSteppedVoltageElement</a> object to be assigned.
--------------	---

## Returns

A reference to the assigned [AisSteppedVoltageElement](#) object.

**15.26.3.13 setApproxMaxCurrent()**

```
void AisSteppedVoltageElement::setApproxMaxCurrent (
    double approxMaxCurrent )
```

Set the approximate maximum current.

## Parameters

<i>approxMaxCurrent</i>	The approximate maximum current in Amps.
-------------------------	--

**15.26.3.14 setEndVoltage()**

```
void AisSteppedVoltageElement::setEndVoltage (
    double vEnd )
```

Set the ending voltage for the experiment.

## Parameters

<i>vEnd</i>	The ending voltage in volts.
-------------	------------------------------

**15.26.3.15 setEndVoltageVsOCP()**

```
void AisSteppedVoltageElement::setEndVoltageVsOCP (
    bool endVsOCP )
```

Set whether the experiment ends with the open circuit potential.

## Parameters

<i>endVsOCP</i>	True to end with open circuit potential, false otherwise.
-----------------	---

### 15.26.3.16 setSamplingInterval()

```
void AisSteppedVoltageElement::setSamplingInterval (
    double samplingInterval )
```

Set the data sampling interval.

#### Parameters

<i>samplingInterval</i>	The data sampling interval in seconds.
-------------------------	--

### 15.26.3.17 setStartVoltage()

```
void AisSteppedVoltageElement::setStartVoltage (
    double vStart )
```

Set the starting voltage for the experiment.

#### Parameters

<i>vStart</i>	The starting voltage in volts.
---------------	--------------------------------

### 15.26.3.18 setStartVoltageVsOCP()

```
void AisSteppedVoltageElement::setStartVoltageVsOCP (
    bool startVsOCP )
```

Set whether the experiment starts with the open circuit potential.

#### Parameters

<i>startVsOCP</i>	True to start with open circuit potential, false otherwise.
-------------------	---

### 15.26.3.19 setStepDuration()

```
void AisSteppedVoltageElement::setStepDuration (
    double duration )
```

Set the time step for each iteration.

**Parameters**

<i>tStep</i>	The time step in seconds.
--------------	---------------------------

**15.26.3.20 setStepSize()**

```
void AisSteppedVoltageElement::setStepSize (
    double stepsize )
```

Set the voltage step for each iteration.

**Parameters**

<i>vStep</i>	The voltage step in volts.
--------------	----------------------------

The documentation for this class was generated from the following file:

- AisSteppedVoltageElement.h

# Chapter 16

## File Documentation

### 16.1 AisCompRange.h

```
1 #ifndef SQUIDSTATLIBRARY_AISCOMPRANGE_H
2 #define SQUIDSTATLIBRARY_AISCOMPRANGE_H
3
4 #include "AisSquidstatGlobal.h"
5 #include <QString>
6 #include <memory>
7
8 class AisCompRangePrivate;
9
10 class SQUIDSTATLIBRARY_EXPORT AisCompRange final {
11 public:
12     explicit AisCompRange(const QString& compRangeName, uint8_t bandwidthIndex, uint8_t stabilityFactor);
13
14     AisCompRange(const AisCompRange&);
15
16     uint8_t getBandwidthIndex() const;
17
18     void setBandwidthIndex(uint8_t index);
19
20     uint8_t getStabilityFactor() const;
21
22     void setStabilityFactor(uint8_t factor);
23
24     void setCompRangeName(const QString& compRangeName);
25
26     const QString& getCompRangeName() const;
27
28 private:
29     std::shared_ptr<AisCompRangePrivate> m_data;
30 };
31 #endif
```

### 16.2 AisDataManipulator.h

```
1 #ifndef AISDATAMANIPULATOR_H
2 #define AISDATAMANIPULATOR_H
3
4 #include "AisDataPoints.h"
5 #include "AisErrorCode.h"
6 #include "AisManipulatorType.h"
7 #include "AisSquidstatGlobal.h"
8 #include <iostream>
9 #include <string>
10 #include <memory>
11
12 class AisDataManipulatorPrivate;
13 class SQUIDSTATLIBRARY_EXPORT AisDataManipulator {
14 public:
15     AisDataManipulator();
16
17     AisErrorCode setPulseType(AisPulseType type, double pulseWidth, double pulsePeriod);
```

```

40
50     AisErrorCode setPulseType(AisPulseType type, double frequency);
51
56     double getPulseWidth() const;
57
62     double getPulsePeriod() const;
63
68     double getFrequency() const;
69
74     bool isPulseCompleted() const;
75
80     double getBaseCurrent() const;
81
86     double getPulseCurrent() const;
87
92     double getBaseVoltage() const;
93
98     double getPulseVoltage() const;
99
104     void loadPrimaryData(const AisDCData& data);
105
106 private:
107     std::shared_ptr<AisDataManipulatorPrivate> m_data;
108 };
109
110 #endif // AISDATAMANIPULATOR_H

```

## 16.3 AisDataPoints.h

```

1 #ifndef SQUIDSTATLIBRARY_AISDATAPOINTS_H
2 #define SQUIDSTATLIBRARY_AISDATAPOINTS_H
3
4 #include <qstring.h>
5
9 struct AisDCData {
10
14     double timestamp;
15
19     double workingElectrodeVoltage;
20
24     double counterElectrodeVoltage;
25
29     double current;
30
34     double temperature;
35 };
36
40 struct AisACData {
41
45     double timestamp;
46
50     double frequency;
51
55     double absoluteImpedance;
56
60     double realImpedance;
61
65     double imagImpedance;
66
70     double phaseAngle;
71
75     double totalHarmonicDistortion;
76
83     double numberOfCycles;
84
88     double workingElectrodeDCVoltage;
89
93     double DCCurrent;
94
98     double currentAmplitude;
99
103     double voltageAmplitude;
104 };
105
109 struct AisExperimentNode {
110
114     QString stepName;
115
119     int stepNumber;
120
124     int substepNumber;
125
126

```

```

130     int cycle;
131 };
132
133 #endif //SQUIDSTATLIBRARY_AISDATAPOINTS_H

```

## 16.4 AisDeviceTracker.h

```

1 #ifndef SQUIDSTATLIBRARY_AISDEVICETRACKER_H
2 #define SQUIDSTATLIBRARY_AISDEVICETRACKER_H
3
4 #include "AisErrorCode.h"
5 #include "AisSquidstatGlobal.h"
6 #include <QObject>
7 #include <memory>
8
9
10
11 class AisDeviceTrackerPrivate;
12 class AisInstrumentHandler;
13
14
15 class SQUIDSTATLIBRARY_EXPORT AisDeviceTracker final : public QObject
16 {
17     Q_OBJECT
18 public:
19     ~AisDeviceTracker() override;
20     static AisDeviceTracker *Instance();
21
22     AisErrorCode connectToDeviceOnComPort(const QString &comPort);
23
24     const AisInstrumentHandler &getInstrumentHandler(const QString &deviceName) const;
25
26     const std::list<QString> getConnectedDevices() const;
27
28     int connectAllPluggedInDevices();
29
30     AisErrorCode updateFirmwareOnComPort(const QString& comport) const;
31
32     int updateFirmwareOnAllAvailableDevices();
33
34     void saveLogToFile(bool save);
35
36     void setLogFilePath(const QString& path);
37
38 signals:
39     void newDeviceConnected(const QString &deviceName);
40
41     void deviceDisconnected(const QString &deviceName);
42
43     void firmwareUpdateNotification(const QString& message);
44
45 private:
46     AisDeviceTracker();
47     AisDeviceTracker(const AisDeviceTracker &);
48     void operator=(const AisDeviceTracker &);
49
50     std::unique_ptr<AisDeviceTrackerPrivate> m_data;
51 };
52
53 #endif

```

## 16.5 AisErrorCode.h

```

1
2 #ifndef AIS_ERROR_CODE_H
3 #define AIS_ERROR_CODE_H
4
5 #include "AisSquidstatGlobal.h"
6 #include <qstring.h>
7
8 class SQUIDSTATLIBRARY_EXPORT AisErrorCode {
9
10 public:
11     enum ErrorCode : uint8_t {
12         Unknown = 255,
13         Success = 0,
14         ConnectionFailed = 1,
15         FirmwareNotSupported = 2,

```

```

23     FirmwareFileNotFound = 3,
24     FirmwareUptodate = 4,
25
26     InvalidChannel = 10,
27     BusyChannel = 11,
28     DeviceNotFound = 13,
30     ManualExperimentNotRunning = 51,
31     ExperimentNotUploaded = 52,
32     ExperimentIsEmpty = 53,
33     InvalidParameters = 54,
34     ChannelNotBusy = 55,
35     ExperimentUploaded = 56,
37     DeviceCommunicationFailed = 100,
39     FailedToSetManualModeCurrentRange = 101,
40     FailedToSetManualModeConstantVoltage = 102,
41     FailedToPauseExperiment = 103,
42     FailedToResumeExperiment = 104,
43     FailedToStopExperiment = 105,
44     FailedToUploadExperiment = 106,
45     ExperimentAlreadyPaused = 107,
46     ExperimentAlreadyRun = 108,
47     FailedToSetManualModeVoltageRange = 109, /* !< indicates failure to set manual mode voltage range
    due to a possible communication failure with the device.*/
48     FailedToSetManualModeConstantCurrent = 110,
49     FailedToSetManualModeInOCP = 111,
50     FailedToSetManualModeSamplingInterval = 112,
51     FailedToSetIRComp = 113,
52     FailedToSetCompRange = 114,
54     FailedRequest = 254 /* !< indicates a failed request to the device. */
55 };
56
57 AisErrorCode();
58 AisErrorCode(ErrorCode error);
59 AisErrorCode(ErrorCode error, QString message);
60
61 QString message() const;
62
63 int value() const;
64
65 operator ErrorCodes() const;
66
67 private:
68     ErrorCodes code;
69     QString errorMessage;
70 };
71
72 #endif // ! AIS_ERROR_CODE_H

```

## 16.6 AisExperiment.h

```

1 #ifndef SQUIDSTATLIBRARY_AISCUSTOMEXPERIMENTRUNNER_H
2 #define SQUIDSTATLIBRARY_AISCUSTOMEXPERIMENTRUNNER_H
3
4 #include "AisSquidstatGlobal.h"
5 #include "experiments/builder_elements/AisAbstractElement.h"
6 #include <QString>
7
8 class CustomExperimentRunner;
9 class AisExperimentPrivate;
10
11 class SQUIDSTATLIBRARY_EXPORT AisExperiment final {
12 public:
13     explicit AisExperiment();
14
15     explicit AisExperiment(const AisExperiment& exp);
16
17     void operator=(const AisExperiment& exp);
18
19     ~AisExperiment();
20
21     QString getExperimentName() const;
22
23     QString getDescription() const;
24
25     QStringList getCategory() const;
26
27     void setExperimentName(QString name);
28
29     void setDescription(QString description);
30
31     bool appendElement(AisAbstractElement& element, uint repeat = 1);
32
33     bool appendSubExperiment(const AisExperiment& subExp, uint repeat = 1);

```



```

90
91 private:
92     friend class AisInstrumentHandler;
93     std::shared_ptr<AisExperimentPrivate> m_data;
94 };
95
96 #endif //SQUIDSTATLIBRARY_AISCUSTOMEXPERIMENTRUNNER_H

```

## 16.7 AisInstrumentHandler.h

```

1  #ifndef SQUIDSTATLIBRARY_AISINSTRUMENTHANDLER_H
2  #define SQUIDSTATLIBRARY_AISINSTRUMENTHANDLER_H
3
4  #include <ctime>
5
6  #include <QObject>
7
8  #include "AisCompRange.h"
9  #include "AisDataPoints.h"
10 #include "AisErrorCode.h"
11 #include "AisSquidstatGlobal.h"
12
13 class AisInstrumentHandlerPrivate;
14 class AisExperiment;
15
24 class SQUIDSTATLIBRARY_EXPORT AisInstrumentHandler final : public QObject {
25     Q_OBJECT
26     AisInstrumentHandlerPrivate* m_data;
27 public:
29     explicit AisInstrumentHandler(AisInstrumentHandlerPrivate* privateData);
31     ~AisInstrumentHandler();
32
55     AisErrorCode uploadExperimentToChannel(uint8_t channel, std::shared_ptr<AisExperiment> experiment)
66     const;
56
79     AisErrorCode uploadExperimentToChannel(uint8_t channel, const AisExperiment& experiment) const;
80
94     AisErrorCode startUploadedExperiment(uint8_t channel) const;
95
107    AisErrorCode startIdleSampling(uint8_t channel) const;
108
125    AisErrorCode skipExperimentStep(uint8_t channel) const;
126
139    AisErrorCode pauseExperiment(uint8_t channel) const;
140
153    AisErrorCode resumeExperiment(uint8_t channel) const;
154
166    AisErrorCode stopExperiment(uint8_t channel) const;
167
175    double getExperimentUTCStartTime(uint8_t channel) const;
176
189    AisErrorCode setIRComp(uint8_t channel, double uncompensatedResistance, double compensationLevel)
190    const;
190
202    AisErrorCode setCompRange(uint8_t channel, const AisCompRange& compRange) const;
203
214    int8_t setLinkedChannels(std::vector<uint8_t> channels) const;
215
227    int8_t setBipolarLinkedChannels(std::vector<uint8_t> channels) const;
228
229
235    bool hasBipolarMode(uint8_t channel) const;
236
242    std::vector<uint8_t> getLinkedChannels(uint8_t channel) const;
243
249    bool isChannelBusy(uint8_t channel) const;
250
256    bool isChannelPaused(uint8_t channel) const;
257
262    std::vector<uint8_t> getFreeChannels() const;
263
268    int getNumberOfChannels() const;
269
277    AisErrorCode eraseRecoverData() const;
278
292    AisErrorCode startManualExperiment(uint8_t channel) const;
293
304    AisErrorCode setManualModeSamplingInterval(uint8_t channel, double value) const;
305
319    AisErrorCode setManualModeOCP(uint8_t channel) const;
320
332    AisErrorCode setManualModeConstantVoltage(uint8_t channel, double value) const;
333

```

```

348     [[deprecated("This has been replaced by setManualModeCurrentRange().")]]
349     AisErrorCode setManualModeConstantVoltage(uint8_t channel, double value, int currentRangeIndex)
350     const;
351
352     AisErrorCode setManualModeCurrentRange(uint8_t channel, int currentRangeIndex) const;
353
354     AisErrorCode setManualModeCurrentAutorange(uint8_t channel) const;
355
356     AisErrorCode setManualModeVoltageRange(uint8_t channel, int voltageRangeIndex) const;
357
358     AisErrorCode setManualModeVoltageAutorange(uint8_t channel) const;
359
360     AisErrorCode setManualModeConstantCurrent(uint8_t channel, double value) const;
361
362     std::vector<std::pair<double, double>> getManualModeCurrentRangeList(uint8_t channel) const;
363
364     std::vector<std::pair<double, double>> getManualModeVoltageRangeList(uint8_t channel) const;
365
366 signals:
367     void deviceDisconnected();
368
369     void groundFloatStateChanged(bool grounded);
370
371     void experimentNewElementStarting(uint8_t channel, const AisExperimentNode& stepInfo);
372
373     void activeDCDataReady(uint8_t channel, const AisDCData& DCData);
374
375     void idleDCDataReady(uint8_t channel, const AisDCData& DCData);
376
377     void recoveryDCDataReady(uint8_t channel, const AisDCData& DCData);
378
379     void activeACDataReady(uint8_t channel, const AisACData& ACData);
380
381     void recoveryACDataReady(uint8_t channel, const AisACData& ACData);
382
383     void experimentStopped(uint8_t channel);
384
385     void experimentPaused(uint8_t channel);
386
387     void experimentResumed(uint8_t channel);
388
389     void recoverDataErased(bool successful);
390
391     void deviceError(uint8_t channel, const QString& error);
392
393 private slots:
394     void onActiveExperimentNodeBeginning(uint8_t channel, const AisExperimentNode&);
395     void onRecoveryExperimentNodeBeginning(uint8_t channel, const AisExperimentNode&);
396     void onDeviceDisconnected();
397
398 private:
399     void connectWithOperatorSignals();
400 };
401
402 #endif //SQUIDSTATLIBRARY_AISINSTRUMENTHANDLER_H

```

## 16.8 AisManipulatorType.h

```

1 #ifndef AISMANIPULATORTYPE_H
2 #define AISMANIPULATORTYPE_H
3
4 enum AisPulseType {
5     DifferentialPulse = 0,
6     NormalPulse = 1,
7     SquarewavePulse = 2
8 };
9
10 #endif // AISMANIPULATORTYPE_H

```

## 16.9 AisSquidstatGlobal.h

```

1 #pragma once
2
3 #include <QtGlobal>
4
5 /*

```

```

6  * For the API:
7  * - We should only use the SQUIDSTATLIBRARY_EXPORT macro
8  * - Any class that should be exposed to the user should be marked with SQUIDSTATLIBRARY_EXPORT
9  * For the SUI:
10 * - We should only use the SUI_SQUIDSTATLIBRARY_EXPORT macro
11 * - It will expose some internal members of the API that are shared among different libraries in the
    SUI
12 * - TODO: remove all the different libraries so dependencies are less complicated
13 */
14
15 #ifndef BUILD_STATIC
16     #if defined(SQUIDSTATLIBRARY_LIB)
17         #define SQUIDSTATLIBRARY_EXPORT Q_DECL_EXPORT
18     #if defined(SUI_BUILD)
19         #define SUI_SQUIDSTATLIBRARY_EXPORT Q_DECL_EXPORT
20     #else
21         #define SUI_SQUIDSTATLIBRARY_EXPORT
22     #endif
23 #else
24     #define SQUIDSTATLIBRARY_EXPORT Q_DECL_IMPORT
25     #if defined(SUI_BUILD)
26         #define SUI_SQUIDSTATLIBRARY_EXPORT Q_DECL_IMPORT
27     #else
28         #define SUI_SQUIDSTATLIBRARY_EXPORT
29     #endif
30 #endif
31 #else
32     #define SQUIDSTATLIBRARY_EXPORT
33     #define SUI_SQUIDSTATLIBRARY_EXPORT
34 #endif
35

```

## 16.10 AisAbstractElement.h

```

1  #ifndef SQUIDSTATLIBRARY_AISABSTRACTELEMENT_H
2  #define SQUIDSTATLIBRARY_AISABSTRACTELEMENT_H
3
4  #include "AisSquidstatGlobal.h"
5  #include <QString>
6  #include <memory>
7
8  class AbstractBuilderElement;
9
15 class SQUIDSTATLIBRARY_EXPORT AisAbstractElement {
16 public:
17     virtual ~AisAbstractElement();
18
24     virtual QString getName() const = 0;
25
31     virtual QStringList getCategory() const = 0;
32
33 protected:
35     std::shared_ptr<AbstractBuilderElement> m_data;
37     friend class AisExperiment;
38 };
39
40 #endif //SQUIDSTATLIBRARY_AISABSTRACTELEMENT_H

```

## 16.11 AisConstantCurrentElement.h

```

1  #pragma once
2
3  #include "AisAbstractElement.h"
4  #include "AisSquidstatGlobal.h"
5  #include <QString>
6
7  class ConstantCurrentAdvElement;
8
15 class SQUIDSTATLIBRARY_EXPORT AisConstantCurrentElement final : public AisAbstractElement {
16 public:
23     explicit AisConstantCurrentElement (
24         double current,
25         double samplingInterval,
26         double duration);
30     explicit AisConstantCurrentElement(const AisConstantCurrentElement&);
34     AisConstantCurrentElement& operator=(const AisConstantCurrentElement&);
35
36     ~AisConstantCurrentElement() override;
37

```

```

42     QString getName() const override;
43
44     QStringList getCategory() const override;
45
46     double getCurrent() const;
47
48     void setCurrent(double current);
49
50     double getSamplingInterval() const;
51
52     void setSamplingInterval(double samplingInterval);
53
54     double getMinSamplingVoltageDifference() const;
55
56     void setMinSamplingVoltageDifference(double minVoltageDifference);
57
58     double getMaxVoltage() const;
59
60     void setMaxVoltage(double maxVoltage);
61
62     double getMinVoltage() const;
63
64     void setMinVoltage(double minVoltage);
65
66     double getMaxDuration() const;
67
68     void setMaxDuration(double maxDuration);
69
70     double getMaxCapacity() const;
71
72     void setMaxCapacity(double maxCapacity);
73
74     bool isAutoRange() const;
75
76     void setAutoRange();
77
78     double getApproxMaxCurrent() const;
79
80     void setApproxMaxCurrent(double approxMaxCurrent);
81
82     bool isAutoVoltageRange() const;
83
84     void setAutoVoltageRange();
85
86     double getApproxMaxVoltage() const;
87
88     void setApproxMaxVoltage(double approxMaxVoltage);
89
90 private:
91     std::shared_ptr<ConstantCurrentAdvElement> m_dataDerived;
92 };

```

## 16.12 AisConstantPotElement.h

```

1  #pragma once
2
3  #include "AisAbstractElement.h"
4  #include "AisSquidstatGlobal.h"
5  #include <QString>
6
7  class ConstantPotAdvElement;
8
9  class SQUIDSTATLIBRARY_EXPORT AisConstantPotElement final : public AisAbstractElement {
10 public:
11     explicit AisConstantPotElement(
12         double voltage,
13         double samplingInterval,
14         double duration);
15     explicit AisConstantPotElement(const AisConstantPotElement&);
16
17     AisConstantPotElement& operator=(const AisConstantPotElement&);
18
19     ~AisConstantPotElement() override;
20
21     QString getName() const override;
22
23     QStringList getCategory() const override;
24
25     double getPotential() const;
26
27     void setPotential(double potential);
28

```

```

68     bool isVoltageVsOCP() const;
69
70     void setVoltageVsOCP(bool vsOCP);
71
72     double getSamplingInterval() const;
73
74     void setSamplingInterval(double samplingInterval);
75
76     double getMaxDuration() const;
77
78     void setMaxDuration(double maxDuration);
79
80     double getMaxCurrent() const;
81
82     void setMaxCurrent(double maxCurrent);
83
84     double getMinCurrent() const;
85
86     void setMinCurrent(double minCurrent);
87
88     double getMaxCapacity() const;
89
90     void setMaxCapacity(double maxCapacity);
91
92     double getMindIdt() const;
93
94     void setMindIdt(double mindIdt);
95
96     bool isAutoRange() const;
97
98     void setAutoRange();
99
100    double getApproxMaxCurrent() const;
101
102    void setApproxMaxCurrent(double approxMaxCurrent);
103
104    void setVoltageRange(int idx);
105
106 private:
107     std::shared_ptr<ConstantPotAdvElement> m_dataDerived;
108 };

```

## 16.13 AisConstantPowerElement.h

```

1  #pragma once
2
3  #include "AisAbstractElement.h"
4  #include "AisSquidstatGlobal.h"
5  #include <QString>
6
7  class ConstantPowerElement;
8
9  class SQUIDSTATLIBRARY_EXPORT AisConstantPowerElement final : public AisAbstractElement {
10 public:
11     explicit AisConstantPowerElement(
12         bool isCharge,
13         double power,
14         double duration,
15         double samplingInterval);
16
17     explicit AisConstantPowerElement(const AisConstantPowerElement&);
18
19     AisConstantPowerElement& operator=(const AisConstantPowerElement&);
20
21     ~AisConstantPowerElement() override;
22
23     QString getName() const override;
24
25     QStringList getCategory() const override;
26
27     bool isCharge() const;
28
29     void setCharge(bool isCharge);
30
31     double getPower() const;
32
33     void setPower(double power);
34
35     double getSamplingInterval() const;
36
37     void setSamplingInterval(double samplingInterval);
38
39     double getMaxVoltage() const;

```

```

97
106 void setMaxVoltage(double maxVoltage);
107
108
114 bool isMaximumVoltageVsOCP() const;
115
122 void setMaximumVoltageVsOCP(bool vsOCP);
123
124
131 double getMinVoltage() const;
132
139 void setMinVoltage(double minVoltage);
140
146 bool isMinimumVoltageVsOCP() const;
147
154 void setMinimumVoltageVsOCP(bool vsOCP);
155
161 double getMaxCurrent() const;
162
171 void setMaxCurrent(double maxCurrent);
172
178 double getMinCurrent() const;
179
188 void setMinCurrent(double maxCurrent);
189
195 double getMaxDuration() const;
196
203 void setMaxDuration(double maxDuration);
204
210 double getMaxCapacity() const;
211
220 void setMaxCapacity(double maxCapacity);
221
222 private:
223     std::shared_ptr<ConstantPowerElement> m_dataDerived;
224 };

```

## 16.14 AisConstantResistanceElement.h

```

1 #pragma once
2
3 #include "AisAbstractElement.h"
4 #include "AisSquidstatGlobal.h"
5 #include <QString>
6
7 class ConstantResistanceElement;
8
15 class SQUIDSTATLIBRARY_EXPORT AisConstantResistanceElement final : public AisAbstractElement {
16 public:
23     explicit AisConstantResistanceElement(
24         double resistance,
25         double duration,
26         double samplingInterval);
30     explicit AisConstantResistanceElement(const AisConstantResistanceElement&);
34     AisConstantResistanceElement& operator=(const AisConstantResistanceElement&);
35
36     ~AisConstantResistanceElement() override;
37
42     QString getName() const override;
43
48     QStringList getCategory() const override;
49
54     double getResistance() const;
55
60     void setResistance(double resistance);
61
66     double getSamplingInterval() const;
67
72     void setSamplingInterval(double samplingInterval);
73
79     double getMinVoltage() const;
80
89     void setMinVoltage(double minVoltage);
90
96     bool isMinimumVoltageVsOCP() const;
97
104     void setMinimumVoltageVsOCP(bool vsOCP);
105
111     double getMaxVoltage() const;
112
121     void setMaxVoltage(double maxVoltage);
122
128     bool isMaximumVoltageVsOCP() const;

```

```

129
136     void setMaximumVoltageVsOCP (bool vsOCP);
137
143     double getMaxCurrent () const;
144
153     void setMaxCurrent (double maxCurrent);
154
160     double getMinCurrent () const;
161
170     void setMinCurrent (double maxCurrent);
171
177     double getMaxDuration () const;
178
185     void setMaxDuration (double maxDuration);
186
192     double getMaxCapacity () const;
193
202     void setMaxCapacity (double maxCapacity);
203
204 private:
205     std::shared_ptr<ConstantResistanceElement> m_dataDerived;
206 };

```

## 16.15 AisCyclicVoltammetryElement.h

```

1 #pragma once
2
3 #include "AisAbstractElement.h"
4 #include "AisSquidstatGlobal.h"
5 #include <QString>
6
7 class CyclicVoltammetryElement;
8
22 class SQUIDSTATLIBRARY_EXPORT AisCyclicVoltammetryElement final : public AisAbstractElement {
23 public:
33     explicit AisCyclicVoltammetryElement (
34         double startVoltage,
35         double firstVoltageLimit,
36         double secondVoltageLimit,
37         double endVoltage,
38         double dEdt,
39         double samplingInterval);
40
44     explicit AisCyclicVoltammetryElement (const AisCyclicVoltammetryElement&);
45
49     AisCyclicVoltammetryElement& operator=(const AisCyclicVoltammetryElement&);
50
51     ~AisCyclicVoltammetryElement () override;
52
57     QString getName () const override;
58
63     QStringList getCategory () const override;
64
69     double getQuietTime () const;
70
75     void setQuietTime (double quietTime);
76
81     double getQuietTimeSamplingInterval () const;
82
87     void setQuietTimeSamplingInterval (double quietTimeSamplingInterval);
88
93     double getStartVoltage () const;
94
99     void setStartVoltage (double startVoltage);
100
105     bool isStartVoltageVsOCP () const;
106
113     void setStartVoltageVsOCP (bool startVoltageVsOCP);
114
122     double getFirstVoltageLimit () const;
123
131     void setFirstVoltageLimit (double v1);
132
138     bool isFirstVoltageLimitVsOCP () const;
139
146     void setFirstVoltageLimitVsOCP (bool firstVoltageLimitVsOCP);
147
155     double getSecondVoltageLimit () const;
156
164     void setSecondVoltageLimit (double v2);
165
171     bool isSecondVoltageLimitVsOCP () const;
172

```

```

179 void setSecondVoltageLimitVsOCP (bool secondVoltageLimitVsOCP);
180
185 double getNumberOfCycles();
186
191 void setNumberOfCycles(int cycles);
192
200 double getEndVoltage() const;
201
209 void setEndVoltage(double endVoltage);
210
216 bool isEndVoltageVsOCP() const;
217
224 void setEndVoltageVsOCP (bool endVoltageVsOCP);
225
230 double getdEdt() const;
231
236 void setdEdt (double dEdt);
237
242 double getSamplingInterval() const;
243
248 void setSamplingInterval(double sampInterval);
249
254 bool isAutoRange() const;
255
261 void setAutoRange();
262
268 double getApproxMaxCurrent() const;
269
277 void setApproxMaxCurrent(double approxMaxCurrent);
278
285 double getAlphaFactor();
286
294 void setAlphaFactor(double alphaFactor);
295
296 private:
297     std::shared_ptr<CyclicVoltammetryElement> m_dataDerived;
298 };

```

## 16.16 AisDCCurrentSweepElement.h

```

1 #pragma once
2
3 #include "AisAbstractElement.h"
4 #include "AisSquidstatGlobal.h"
5 #include <QString>
6
7 class DCCurrentSweepElement;
8
17 class SQUIDSTATLIBRARY_EXPORT AisDCCurrentSweepElement : public AisAbstractElement {
18 public:
26     explicit AisDCCurrentSweepElement(
27         double startCurrent,
28         double endCurrent,
29         double scanRate,
30         double samplingInterval
31     );
32
36     explicit AisDCCurrentSweepElement(const AisDCCurrentSweepElement&);
40     AisDCCurrentSweepElement& operator=(const AisDCCurrentSweepElement&);
41
42     ~AisDCCurrentSweepElement() override;
43
48     QString getName() const override;
49
54     QStringList getCategory() const override;
55
60     double getQuietTime() const;
61
66     void setQuietTime(double quietTime);
67
72     double getQuietTimeSamplingInterval() const;
73
78     void setQuietTimeSamplingInterval(double quietTimeSamplingInterval);
79
84     double getStartingCurrent() const;
85
90     void setStartingCurrent(double startingCurrent);
91
96     double getEndingCurrent() const;
97
102     void setEndingCurrent(double endingCurrent);
103
109     double getScanRate() const;

```



```

110
117     void setScanRate(double scanRate);
118
123     double getSamplingInterval() const;
124
129     void setSamplingInterval(double samplingInterval);
130
137     double getMaxVoltage() const;
138
147     void setMaxVoltage(double maxVoltage);
148
154     double getMinVoltage() const;
155
164     void setMinVoltage(double minVoltage);
165
172     double getAlphaFactor();
173
181     void setAlphaFactor(double alphaFactor);
182
183 private:
184     std::shared_ptr<DCCurrentSweepElement> m_dataDerived;
185 };

```

## 16.17 AisDCPotentialSweepElement.h

```

1 #pragma once
2
3 #include "AisAbstractElement.h"
4 #include "AisSquidstatGlobal.h"
5 #include <QString>
6
7 class DCPotentialSweepElement;
8
17 class SQUIDSTATLIBRARY_EXPORT AisDCPotentialSweepElement final : public AisAbstractElement {
18 public:
26     explicit AisDCPotentialSweepElement(
27         double startPotential,
28         double endPotential,
29         double scanRate,
30         double samplingInterval);
34     explicit AisDCPotentialSweepElement(const AisDCPotentialSweepElement&);
38     AisDCPotentialSweepElement& operator=(const AisDCPotentialSweepElement&);
39
40     ~AisDCPotentialSweepElement() override;
41
46     QString getName() const override;
47
52     QStringList getCategory() const override;
53
58     double getQuietTime() const;
59
64     void setQuietTime(double quietTime);
65
70     double getQuietTimeSamplingInterval() const;
71
76     void setQuietTimeSamplingInterval(double quietTimeSamplingInterval);
77
82     double getStartingPot() const;
83
88     void setStartingPot(double startingPotential);
89
95     bool isStartVoltageVsOCP() const;
96
104     void setStartVoltageVsOCP(bool startVoltageVsOCP);
105
112     double getEndingPot() const;
113
120     void setEndingPot(double endingPotential);
121
127     bool isEndVoltageVsOCP() const;
128
136     void setEndVoltageVsOCP(bool endVoltageVsOCP);
137
143     double getScanRate() const;
144
151     void setScanRate(double scanRate);
152
157     double getSamplingInterval() const;
158
163     void setSamplingInterval(double samplingInterval);
164
169     bool isAutoRange() const;
170

```

```

176     void setAutoRange();
177
183     double getApproxMaxCurrent() const;
184
192     void setApproxMaxCurrent(double approxMaxCurrent);
193
194
201     double getMaxAbsoluteCurrent() const;
202
211     void setMaxAbsoluteCurrent(double maxCurrent);
212
218     double getMinAbsoluteCurrent() const;
219
228     void setMinAbsoluteCurrent(double minCurrent);
229
236     double getAlphaFactor();
237
245     void setAlphaFactor(double alphaFactor);
246
247 private:
248     std::shared_ptr<DCPotentialSweepElement> m_dataDerived;
249 };

```

## 16.18 AisDiffPulseVoltammetryElement.h

```

1 #pragma once
2
3 #include "AisAbstractElement.h"
4 #include "AisSquidstatGlobal.h"
5 #include <QString>
6
7 class DiffPulseVoltammetryElement;
8
23 class SQUIDSTATLIBRARY_EXPORT AisDiffPulseVoltammetryElement final : public AisAbstractElement {
24 public:
34     explicit AisDiffPulseVoltammetryElement (
35         double startVoltage,
36         double endVoltage,
37         double voltageStep,
38         double pulseHeight,
39         double pulseWidth,
40         double pulsePeriod);
44     explicit AisDiffPulseVoltammetryElement(const AisDiffPulseVoltammetryElement&);
48     AisDiffPulseVoltammetryElement& operator=(const AisDiffPulseVoltammetryElement&);
49
50     ~AisDiffPulseVoltammetryElement() override;
51
56     QString getName() const override;
57
62     QStringList getCategory() const override;
63
68     double getQuietTime() const;
69
74     void setQuietTime(double quietTime);
75
80     double getQuietTimeSamplingInterval() const;
81
86     void setQuietTimeSamplingInterval(double quietTimeSamplingInterval);
87
92     double getStartVoltage() const;
93
98     void setStartVoltage(double startVoltage);
99
105     bool isStartVoltageVsOCP() const;
106
114     void setStartVoltageVsOCP(bool startVoltageVsOCP);
115
122     double getEndVoltage() const;
123
130     void setEndVoltage(double endVoltage);
131
137     bool isEndVoltageVsOCP() const;
138
146     void setEndVoltageVsOCP(bool endVoltageVsOCP);
147
153     double getVStep() const;
154
161     void setVStep(double vStep);
162
168     double getPulseHeight() const;
169
178     void setPulseHeight(double pulseHeight);
179

```

```

185     double getPulseWidth() const;
186
194     void setPulseWidth(double pulseWidth);
195
201     double getPulsePeriod() const;
202
209     void setPulsePeriod(double pulsePeriod);
210
215     bool isAutoRange() const;
216
222     void setAutoRange();
223
229     double getApproxMaxCurrent() const;
230
239     void setApproxMaxCurrent(double approxMaxCurrent);
240
247     double getAlphaFactor();
248
256     void setAlphaFactor(double alphaFactor);
257
258 private:
259     std::shared_ptr<DiffPulseVoltammetryElement> m_dataDerived;
260 };

```

## 16.19 AisEISGalvanostaticElement.h

```

1 #pragma once
2
3 #include "AisAbstractElement.h"
4 #include "AisSquidstatGlobal.h"
5 #include <QString>
6
7 class EISGalvanostaticElement;
8
20 class SQUIDSTATLIBRARY_EXPORT AisEISGalvanostaticElement final : public AisAbstractElement {
21 public:
30     explicit AisEISGalvanostaticElement(
31         double startFrequency,
32         double endFrequency,
33         double stepsPerDecade,
34         double currentBias,
35         double currentAmplitude);
39     explicit AisEISGalvanostaticElement(const AisEISGalvanostaticElement&);
43     AisEISGalvanostaticElement& operator=(const AisEISGalvanostaticElement&);
44
45     ~AisEISGalvanostaticElement() override;
46
51     double getQuietTime() const;
52
57     void setQuietTime(double quietTime);
58
63     double getQuietTimeSamplingInterval() const;
64
69     void setQuietTimeSamplingInterval(double quietTimeSamplingInterval);
70
75     QString getName() const override;
76
81     QStringList getCategory() const override;
82
83
88     double getStartFreq() const;
89
94     void setStartFreq(double startFreq);
95
100    double getEndFreq() const;
101
106    void setEndFreq(double endFreq);
107
112    double getStepsPerDecade() const;
113
118    void setStepsPerDecade(double stepsPerDecade);
119
124    double getBiasCurrent() const;
125
130    void setBiasCurrent(double biasCurrent);
131
136    double getAmplitude() const;
137
142    void setAmplitude(double amplitude);
143
148    void setMinimumCycles(int numberOfCycle);
149
154    int getMinimumCycles() const;

```

```

155
156 private:
157     std::shared_ptr<EISGalvanostaticElement> m_dataDerived;
158 };

```

## 16.20 AisEISPotentiostaticElement.h

```

1 #pragma once
2
3 #include "AisAbstractElement.h"
4 #include "AisSquidstatGlobal.h"
5 #include <QString>
6
7 class EISPotentiostaticElement;
8
20 class SQUIDSTATLIBRARY_EXPORT AisEISPotentiostaticElement final : public AisAbstractElement {
21 public:
30     explicit AisEISPotentiostaticElement (
31         double startFrequency,
32         double endFrequency,
33         double stepsPerDecade,
34         double voltageBias,
35         double voltageAmplitude);
39     explicit AisEISPotentiostaticElement(const AisEISPotentiostaticElement&);
43     AisEISPotentiostaticElement& operator=(const AisEISPotentiostaticElement&);
44
45     ~AisEISPotentiostaticElement() override;
46
51     QString getName() const override;
52
57     QStringList getCategory() const override;
58
63     double getQuietTime() const;
64
69     void setQuietTime(double quietTime);
70
75     double getQuietTimeSamplingInterval() const;
76
81     void setQuietTimeSamplingInterval(double quietTimeSamplingInterval);
82
87     double getStartFreq() const;
88
93     void setStartFreq(double startFreq);
94
99     double getEndFreq() const;
100
105     void setEndFreq(double endFreq);
106
111     double getStepsPerDecade() const;
112
117     void setStepsPerDecade(double stepsPerDecade);
118
123     double getBiasVoltage() const;
124
129     void setBiasVoltage(double biasVoltage);
130
137     bool isBiasVoltageVsOCP() const;
138
143     void setBiasVoltageVsOCP(bool biasVsOCP);
144
149     double getAmplitude() const;
150
155     void setAmplitude(double amplitude);
156
161     void setMinimumCycles(int numberOfCycle);
162
167     int getMinimumCycles() const;
168
169 private:
170     std::shared_ptr<EISPotentiostaticElement> m_dataDerived;
171 };

```

## 16.21 AisMottSchottkyElement.h

```

1 #pragma once
2
3 #include "AisAbstractElement.h"
4 #include "AisSquidstatGlobal.h"
5 #include <QString>

```

```

6
7 class MottSchottkyElement;
8
26 class SQUIDSTATLIBRARY_EXPORT AisMottSchottkyElement final : public AisAbstractElement {
27 public:
39     explicit AisMottSchottkyElement(double startingPotential, double endingPotential, double voltageStep,
40                                     double startFrequency, double endFrequency, double stepsPerDecade,
41                                     double amplitude, double minCycles);
42
47     explicit AisMottSchottkyElement(const AisMottSchottkyElement& other);
48
54     AisMottSchottkyElement& operator=(const AisMottSchottkyElement& other);
55
59     ~AisMottSchottkyElement() override;
60
65     QString getName() const override;
66
71     QStringList getCategory() const override;
72
73     // Getter and Setter methods
74
79     double getStartingPotential() const;
80
85     void setStartingPotential(double startingPotential);
86
91     double getEndingPotential() const;
92
97     void setEndingPotential(double endingPotential);
98
103     double getVoltageStep() const;
104
109     void setVoltageStep(double voltageStep);
110
115     double getStartFrequency() const;
116
121     void setStartFrequency(double startFrequency);
122
127     double getEndFrequency() const;
128
133     void setEndFrequency(double endFrequency);
134
139     double getStepsPerDecade() const;
140
145     void setStepsPerDecade(double stepsPerDecade);
146
151     double getAmplitude() const;
152
157     void setAmplitude(double amplitude);
158
163     double getMinCycles() const;
164
169     void setMinCycles(double minCycles);
170
175     double getQuietTime() const;
176
181     void setQuietTime(double quietTime);
182
187     double getQuietTimeSampInterval() const;
188
193     void setQuietTimeSampInterval(double quietTimeSampInterval);
194
199     double getStepQuietTime() const;
200
205     void setStepQuietTime(double stepQuietTime);
206
211     double getStepQuietSampInterval() const;
212
217     void setStepQuietSampInterval(double stepQuietTimeSampInterval);
218
223     bool isStartVoltageVsOCP() const;
224
229     void setStartVoltageVsOCP(bool startVsOCP);
230
235     bool isEndVoltageVsOCP() const;
236
241     void setEndVoltageVsOCP(bool endVsOCP);
242
243 private:
244     std::shared_ptr<MottSchottkyElement> m_dataDerived;
245 };

```

## 16.22 AisNormalPulseVoltammetryElement.h

```
1 #pragma once
```

```

2
3 #include "AisAbstractElement.h"
4 #include "AisSquidstatGlobal.h"
5 #include <QString>
6
7 class NormalPulseVoltammetryElement;
8
23 class SQUIDSTATLIBRARY_EXPORT AisNormalPulseVoltammetryElement final : public AisAbstractElement {
24 public:
33     explicit AisNormalPulseVoltammetryElement (
34         double startVoltage,
35         double endVoltage,
36         double voltageStep,
37         double pulseWidth,
38         double pulsePeriod);
42     explicit AisNormalPulseVoltammetryElement(const AisNormalPulseVoltammetryElement&);
46     AisNormalPulseVoltammetryElement& operator=(const AisNormalPulseVoltammetryElement&);
47
48     ~AisNormalPulseVoltammetryElement() override;
49
54     QString getName() const override;
55
60     QStringList getCategory() const override;
61
66     double getQuietTime() const;
67
72     void setQuietTime(double quietTime);
73
78     double getQuietTimeSamplingInterval() const;
79
84     void setQuietTimeSamplingInterval(double quietTimeSamplingInterval);
85
90     double getStartVoltage() const;
91
96     void setStartVoltage(double startVoltage);
97
104     bool isStartVoltageVsOCP() const;
105
113     void setStartVoltageVsOCP(bool startVoltageVsOCP);
114
121     double getEndVoltage() const;
122
129     void setEndVoltage(double endVoltage);
130
136     bool isEndVoltageVsOCP() const;
137
145     void setEndVoltageVsOCP(bool endVoltageVsOcp);
146
152     double getVStep() const;
153
160     void setVStep(double vStep);
161
168     double getPulseWidth() const;
169
176     void setPulseWidth(double pulseWidth);
177
183     double getPulsePeriod() const;
184
191     void setPulsePeriod(double pulsePeriod);
192
197     bool isAutoRange() const;
198
204     void setAutoRange();
205
211     double getApproxMaxCurrent() const;
212
220     void setApproxMaxCurrent(double approxMaxCurrent);
221
228     double getAlphaFactor();
229
237     void setAlphaFactor(double alphaFactor);
238
239 private:
240     std::shared_ptr<NormalPulseVoltammetryElement> m_dataDerived;
241 };

```

## 16.23 AisOpenCircuitElement.h

```

1 #pragma once
2
3 #include "AisAbstractElement.h"
4 #include "AisSquidstatGlobal.h"
5 #include <QString>

```

```

6
7 class OpenCircuitElement;
8
15 class SQUIDSTATLIBRARY_EXPORT AisOpenCircuitElement final : public AisAbstractElement {
16 public:
22     explicit AisOpenCircuitElement(
23         double duration,
24         double samplingInterval);
28     explicit AisOpenCircuitElement(const AisOpenCircuitElement&);
32     AisOpenCircuitElement& operator=(const AisOpenCircuitElement&);
33
34     ~AisOpenCircuitElement() override;
35
40     QString getName() const override;
41
46     QStringList getCategory() const override;
47
52     double getSamplingInterval() const;
53
58     void setSamplingInterval(double samplingInterval);
59
64     double getMaxDuration() const;
65
70     void setMaxDuration(double maxDuration);
71
78     double getMaxVoltage() const;
79
88     void setMaxVoltage(double maxVoltage);
89
95     double getMinVoltage() const;
96
105    void setMinVoltage(double minVoltage);
106
113    double getMindVdt() const;
114
123    void setMindVdt(double mindVdt);
124
129    bool isAutoVoltageRange() const;
130
136    void setAutoVoltageRange();
137
143    double getApproxMaxVoltage() const;
144
152    void setApproxMaxVoltage(double approxMaxVoltage);
153
154 private:
155     std::shared_ptr<OpenCircuitElement> m_dataDerived;
156 };

```

## 16.24 AisSquareWaveVoltammetryElement.h

```

1 #pragma once
2
3 #include "AisAbstractElement.h"
4 #include "AisSquidstatGlobal.h"
5 #include <QString>
6
7 class SquareWaveVoltammetryElement;
8
25 class SQUIDSTATLIBRARY_EXPORT AisSquareWaveVoltammetryElement final : public AisAbstractElement {
26 public:
35     explicit AisSquareWaveVoltammetryElement(
36         double startVoltage,
37         double endVoltage,
38         double voltageStep,
39         double pulseAmp,
40         double pulseFrequency);
41
45     explicit AisSquareWaveVoltammetryElement(const AisSquareWaveVoltammetryElement&);
49     AisSquareWaveVoltammetryElement& operator=(const AisSquareWaveVoltammetryElement&);
50
51     ~AisSquareWaveVoltammetryElement() override;
52
57     QString getName() const override;
58
63     QStringList getCategory() const override;
64
69     double getQuietTime() const;
70
75     void setQuietTime(double quietTime);
76
81     double getQuietTimeSamplingInterval() const;
82

```

```

87     void setQuietTimeSamplingInterval(double quietTimeSamplingInterval);
88
93     double getStartVoltage() const;
94
99     void setStartVoltage(double startVoltage);
100
107     bool isStartVoltageVsOCP() const;
108
116     void setStartVoltageVsOCP(bool startVoltageVsOcp);
117
124     double getEndVoltage() const;
125
132     void setEndVoltage(double endVoltage);
133
139     bool isEndVoltageVsOCP() const;
140
148     void setEndVoltageVsOCP(bool endVoltageVsOcp);
149
155     double getVStep() const;
156
163     void setVStep(double vStep);
164
170     double getPulseAmp() const;
171
179     void setPulseAmp(double pulseAmp);
180
185     double getPulseFreq() const;
186
191     void setPulseFreq(double pulseFreq);
192
197     bool isAutoRange() const;
198
204     void setAutoRange();
205
211     double getApproxMaxCurrent() const;
212
220     void setApproxMaxCurrent(double approxMaxCurrent);
221
228     double getAlphaFactor();
229
237     void setAlphaFactor(double alphaFactor);
238
239 private:
240     std::shared_ptr<SquareWaveVoltammetryElement> m_dataDerived;
241 };

```

## 16.25 AisStaircasePotentialVoltammetryElement.h

```

1  #pragma once
2
3  #include "AisAbstractElement.h"
4  #include "AisSquidstatGlobal.h"
5  #include <QString>
6
7  class StaircasePotentialVoltammetryElement;
8
17 class SQUIDSTATLIBRARY_EXPORT AisStaircasePotentialVoltammetryElement final : public AisAbstractElement {
18 public:
29     AisStaircasePotentialVoltammetryElement(double startVoltage,
30         double firstVoltageLimit,
31         double secondVoltageLimit,
32         double endVoltage,
33         double stepSize,
34         double stepDuration,
35         double samplingInterval);
36
41     AisStaircasePotentialVoltammetryElement(const AisStaircasePotentialVoltammetryElement& other);
42
48     AisStaircasePotentialVoltammetryElement& operator=(const AisStaircasePotentialVoltammetryElement&
49         other);
50
53     ~AisStaircasePotentialVoltammetryElement() override;
54
59     QString getName() const override;
60
65     QStringList getCategory() const override;
66
71     double getQuietTime() const;
72
77     void setQuietTime(double quietTime);
78
83     double getQuietTimeSamplingInterval() const;
84

```



```

89     void setQuietTimeSamplingInterval(double quietTimeSamplingInterval);
90
91     double getStartVoltage() const;
92
93     void setStartVoltage(double startVoltage);
94
95     bool isStartVoltageVsOCP() const;
96
97     void setStartVoltageVsOCP(bool startVsOCP);
98
99     double getEndVoltage() const;
100
101     void setEndVoltage(double endVoltage);
102
103     bool isEndVoltageVsOCP() const;
104
105     void setEndVoltageVsOCP(bool endVsOCP);
106
107     double getFirstVoltageLimit() const;
108
109     void setFirstVoltageLimit(double firstVoltageLimit);
110
111     bool isFirstVoltageLimitVsOCP() const;
112
113     void setFirstVoltageLimitVsOCP(bool firstVoltageLimitVsOCP);
114
115     double getSecondVoltageLimit() const;
116
117     void setSecondVoltageLimit(double secondVoltageLimit);
118
119     bool isSecondVoltageLimitVsOCP() const;
120
121     void setSecondVoltageLimitVsOCP(bool secondVoltageLimitVsOCP);
122
123     double getStepSize() const;
124
125     void setStepSize(double stepSize);
126
127     double getStepDuration() const;
128
129     void setStepDuration(double stepDuration);
130
131     double getSamplingInterval() const;
132
133     void setSamplingInterval(double samplingInterval);
134
135     bool isAutorange() const;
136
137     void setAutorange();
138
139     double getApproxMaxCurrent() const;
140
141     void setApproxMaxCurrent(double approxMaxCurrent);
142
143     double getNumberOfCycles();
144
145     void setNumberOfCycles(int cycles);
146
147 private:
148     std::shared_ptr<StaircasePotentialVoltammetryElement> m_dataDerived;
149 };

```

## 16.26 AisSteppedCurrentElement.h

```

1  #pragma once
2
3  #include "AisAbstractElement.h"
4  #include "AisSquidstatGlobal.h"
5  #include <QString>
6
7  class SteppedCurrent;
8
9  class SQUIDSTATLIBRARY_EXPORT AisSteppedCurrentElement final : public AisAbstractElement {
10 public:
11     explicit AisSteppedCurrentElement(
12         double startCurrent,
13         double endCurrent,
14         double stepSize,
15         double stepDuration,
16         double samplingInterval);
17     explicit AisSteppedCurrentElement(const AisSteppedCurrentElement&);
18     AisSteppedCurrentElement& operator=(const AisSteppedCurrentElement&);
19 };

```

```

43     ~AisSteppedCurrentElement() override;
44
45     QString getName() const override;
46
47     QStringList getCategory() const override;
48
49     double getSamplingInterval() const;
50
51     void setSamplingInterval(double samplingInterval);
52
53     double getEndCurrent() const;
54
55     double getStepSize() const;
56
57     double getStartCurrent() const;
58
59     double getStepDuration() const;
60
61     void setEndCurrent(double iEnd);
62
63     void setStepSize(double iStep);
64
65     void setStartCurrent(double iStart);
66
67     void setStepDuration(double tStep);
68
69     double getApproxMaxVoltage() const;
70
71     void setApproxMaxVoltage(double approxMaxVoltage);
72
73     double getApproxMinVoltage() const;
74
75     void setApproxMinVoltage(double approxMinVoltage);
76
77 private:
78     std::shared_ptr<SteppedCurrent> m_dataDerived;
79 };

```

## 16.27 AisSteppedVoltageElement.h

```

1  #pragma once
2
3  #pragma once
4
5  #include "AisAbstractElement.h"
6  #include "AisSquidstatGlobal.h"
7  #include <QString>
8
9  class SteppedVoltage;
10
11  class SQUIDSTATLIBRARY_EXPORT AisSteppedVoltageElement final : public AisAbstractElement {
12 public:
13     explicit AisSteppedVoltageElement(double startVoltage,
14         double endVoltage,
15         double voltageStep,
16         double voltageStepDuration,
17         double samplingInterval);
18
19     explicit AisSteppedVoltageElement(const AisSteppedVoltageElement& other);
20
21     AisSteppedVoltageElement& operator=(const AisSteppedVoltageElement& other);
22
23     ~AisSteppedVoltageElement() override;
24
25     QString getName() const override;
26
27     QStringList getCategory() const override;
28
29     double getStartVoltage() const;
30
31     double getEndVoltage() const;
32
33     double getStepSize() const;
34
35     double getStepDuration() const;
36
37     double getSamplingInterval() const;
38
39     double getApproxMaxCurrent() const;
40
41     bool isStartVoltageVsOCP() const;
42
43     bool isEndVoltageVsOCP() const;

```

```
117
122     bool isAutoRange() const;
123
128     void setStartVoltage(double vStart);
129
134     void setEndVoltage(double vEnd);
135
140     void setStepSize(double stepsize);
141
146     void setStepDuration(double duration);
147
152     void setSamplingInterval(double samplingInterval);
153
158     void setApproxMaxCurrent(double approxMaxCurrent);
159
164     void setStartVoltageVsOCP(bool startVsOCP);
165
170     void setEndVoltageVsOCP(bool endVsOCP);
171
175     void setCurrentAutorange();
176
177 private:
178     std::shared_ptr<SteppedVoltage> m_dataDerived;
179 };
180
```



# Index

- activeACDataReady
  - AisInstrumentHandler, 170
- activeDCDataReady
  - AisInstrumentHandler, 171
- AisAbstractElement.h, 263
- AisACData, 49
  - numberOfCycles, 50
- AisCompRange, 50
  - AisCompRange, 51
  - getBandwidthIndex, 51
  - getCompRangeName, 51
  - getStabilityFactor, 52
  - setBandwidthIndex, 52
  - setCompRangeName, 52
  - setStabilityFactor, 53
- AisCompRange.h, 257
- AisConstantCurrentElement, 53
  - AisConstantCurrentElement, 55
  - getApproxMaxCurrent, 55
  - getApproxMaxVoltage, 56
  - getCategory, 56
  - getCurrent, 56
  - getMaxCapacity, 57
  - getMaxDuration, 57
  - getMaxVoltage, 57
  - getMinSamplingVoltageDifference, 57
  - getMinVoltage, 58
  - getName, 58
  - getSamplingInterval, 58
  - isAutoRange, 59
  - isAutoVoltageRange, 59
  - setApproxMaxCurrent, 59
  - setApproxMaxVoltage, 60
  - setAutoRange, 60
  - setAutoVoltageRange, 60
  - setCurrent, 60
  - setMaxCapacity, 60
  - setMaxDuration, 61
  - setMaxVoltage, 61
  - setMinSamplingVoltageDifference, 61
  - setMinVoltage, 62
  - setSamplingInterval, 62
- AisConstantCurrentElement.h, 263
- AisConstantPotElement, 63
  - AisConstantPotElement, 64
  - getApproxMaxCurrent, 65
  - getCategory, 65
  - getMaxCapacity, 65
  - getMaxCurrent, 66
  - getMaxDuration, 66
  - getMinCurrent, 66
  - getMindIdt, 67
  - getName, 67
  - getPotential, 67
  - getSamplingInterval, 68
  - isAutoRange, 68
  - isVoltageVsOCP, 68
  - setApproxMaxCurrent, 68
  - setAutoRange, 69
  - setMaxCapacity, 69
  - setMaxCurrent, 69
  - setMaxDuration, 69
  - setMinCurrent, 70
  - setMindIdt, 70
  - setPotential, 70
  - setSamplingInterval, 71
  - setVoltageRange, 71
  - setVoltageVsOCP, 71
- AisConstantPotElement.h, 264
- AisConstantPowerElement, 72
  - AisConstantPowerElement, 74
  - getCategory, 74
  - getMaxCapacity, 74
  - getMaxCurrent, 74
  - getMaxDuration, 75
  - getMaxVoltage, 75
  - getMinCurrent, 75
  - getMinVoltage, 76
  - getName, 76
  - getPower, 76
  - getSamplingInterval, 77
  - isCharge, 77
  - isMaximumVoltageVsOCP, 77
  - isMinimumVoltageVsOCP, 77
  - setCharge, 78
  - setMaxCapacity, 78
  - setMaxCurrent, 78
  - setMaxDuration, 79
  - setMaximumVoltageVsOCP, 79
  - setMaxVoltage, 79
  - setMinCurrent, 80
  - setMinimumVoltageVsOCP, 80
  - setMinVoltage, 80
  - setPower, 81
  - setSamplingInterval, 81
- AisConstantPowerElement.h, 265
- AisConstantResistanceElement, 82
  - AisConstantResistanceElement, 83

- getCategory, [84](#)
- getMaxCapacity, [84](#)
- getMaxCurrent, [84](#)
- getMaxDuration, [84](#)
- getMaxVoltage, [85](#)
- getMinCurrent, [85](#)
- getMinVoltage, [85](#)
- getName, [86](#)
- getResistance, [86](#)
- getSamplingInterval, [86](#)
- isMaximumVoltageVsOCP, [86](#)
- isMinimumVoltageVsOCP, [87](#)
- setMaxCapacity, [87](#)
- setMaxCurrent, [88](#)
- setMaxDuration, [88](#)
- setMaximumVoltageVsOCP, [88](#)
- setMaxVoltage, [89](#)
- setMinCurrent, [89](#)
- setMinimumVoltageVsOCP, [89](#)
- setMinVoltage, [90](#)
- setResistance, [90](#)
- setSamplingInterval, [90](#)
- AisConstantResistanceElement.h, [266](#)
- AisCyclicVoltammetryElement, [91](#)
  - AisCyclicVoltammetryElement, [93](#)
  - getAlphaFactor, [93](#)
  - getApproxMaxCurrent, [93](#)
  - getCategory, [94](#)
  - getdEdt, [94](#)
  - getEndVoltage, [94](#)
  - getFirstVoltageLimit, [94](#)
  - getName, [95](#)
  - getNumberOfCycles, [95](#)
  - getQuietTime, [95](#)
  - getQuietTimeSamplingInterval, [95](#)
  - getSamplingInterval, [96](#)
  - getSecondVoltageLimit, [96](#)
  - getStartVoltage, [96](#)
  - isAutoRange, [96](#)
  - isEndVoltageVsOCP, [97](#)
  - isFirstVoltageLimitVsOCP, [97](#)
  - isSecondVoltageLimitVsOCP, [97](#)
  - isStartVoltageVsOCP, [98](#)
  - setAlphaFactor, [98](#)
  - setApproxMaxCurrent, [98](#)
  - setAutoRange, [99](#)
  - setdEdt, [99](#)
  - setEndVoltage, [99](#)
  - setEndVoltageVsOCP, [99](#)
  - setFirstVoltageLimit, [100](#)
  - setFirstVoltageLimitVsOCP, [100](#)
  - setNumberOfCycles, [100](#)
  - setQuietTime, [101](#)
  - setQuietTimeSamplingInterval, [101](#)
  - setSamplingInterval, [101](#)
  - setSecondVoltageLimit, [102](#)
  - setSecondVoltageLimitVsOCP, [102](#)
  - setStartVoltage, [102](#)
  - setStartVoltageVsOCP, [103](#)
- AisCyclicVoltammetryElement.h, [267](#)
- AisDataManipulator, [103](#)
  - getBaseCurrent, [104](#)
  - getBaseVoltage, [104](#)
  - getFrequency, [104](#)
  - getPulseCurrent, [105](#)
  - getPulsePeriod, [105](#)
  - getPulseVoltage, [105](#)
  - getPulseWidth, [105](#)
  - isPulseCompleted, [106](#)
  - loadPrimaryData, [106](#)
  - setPulseType, [106, 107](#)
- AisDataManipulator.h, [257](#)
- AisDataPoints.h, [258](#)
- AisDCCurrentSweepElement, [107](#)
  - AisDCCurrentSweepElement, [109](#)
  - getAlphaFactor, [110](#)
  - getCategory, [110](#)
  - getEndingCurrent, [110](#)
  - getMaxVoltage, [110](#)
  - getMinVoltage, [111](#)
  - getName, [111](#)
  - getQuietTime, [111](#)
  - getQuietTimeSamplingInterval, [112](#)
  - getSamplingInterval, [112](#)
  - getScanRate, [112](#)
  - getStartingCurrent, [112](#)
  - setAlphaFactor, [113](#)
  - setEndingCurrent, [113](#)
  - setMaxVoltage, [113](#)
  - setMinVoltage, [115](#)
  - setQuietTime, [115](#)
  - setQuietTimeSamplingInterval, [115](#)
  - setSamplingInterval, [116](#)
  - setScanRate, [116](#)
  - setStartingCurrent, [116](#)
- AisDCCurrentSweepElement.h, [268](#)
- AisDCData, [116](#)
- AisDCPotentialSweepElement, [117](#)
  - AisDCPotentialSweepElement, [119](#)
  - getAlphaFactor, [120](#)
  - getApproxMaxCurrent, [120](#)
  - getCategory, [120](#)
  - getEndingPot, [120](#)
  - getMaxAbsoluteCurrent, [121](#)
  - getMinAbsoluteCurrent, [121](#)
  - getName, [121](#)
  - getQuietTime, [122](#)
  - getQuietTimeSamplingInterval, [122](#)
  - getSamplingInterval, [122](#)
  - getScanRate, [122](#)
  - getStartingPot, [123](#)
  - isAutoRange, [123](#)
  - isEndVoltageVsOCP, [123](#)
  - isStartVoltageVsOCP, [123](#)
  - setAlphaFactor, [124](#)
  - setApproxMaxCurrent, [124](#)

- setAutoRange, 124
- setEndingPot, 125
- setEndVoltageVsOCP, 125
- setMaxAbsoluteCurrent, 125
- setMinAbsoluteCurrent, 126
- setQuietTime, 126
- setQuietTimeSamplingInterval, 126
- setSamplingInterval, 127
- setScanRate, 127
- setStartingPot, 127
- setStartVoltageVsOCP, 127
- AisDCPotentialSweepElement.h, 269
- AisDeviceTracker, 128
  - connectAllPluggedInDevices, 129
  - connectToDeviceOnComPort, 129
  - deviceDisconnected, 131
  - getConnectedDevices, 131
  - getInstrumentHandler, 131
  - newDeviceConnected, 132
  - saveLogToFile, 132
  - setLogFilePath, 133
  - updateFirmwareOnAllAvailableDevices, 133
  - updateFirmwareOnComPort, 134
- AisDeviceTracker.h, 259
- AisDiffPulseVoltammetryElement, 134
  - AisDiffPulseVoltammetryElement, 137
  - getAlphaFactor, 137
  - getApproxMaxCurrent, 137
  - getCategory, 138
  - getEndVoltage, 138
  - getName, 138
  - getPulseHeight, 138
  - getPulsePeriod, 139
  - getPulseWidth, 139
  - getQuietTime, 139
  - getQuietTimeSamplingInterval, 140
  - getStartVoltage, 140
  - getVStep, 140
  - isAutoRange, 140
  - isEndVoltageVsOCP, 141
  - isStartVoltageVsOCP, 141
  - setAlphaFactor, 141
  - setApproxMaxCurrent, 142
  - setAutoRange, 142
  - setEndVoltage, 142
  - setEndVoltageVsOCP, 142
  - setPulseHeight, 143
  - setPulsePeriod, 143
  - setPulseWidth, 143
  - setQuietTime, 144
  - setQuietTimeSamplingInterval, 144
  - setStartVoltage, 144
  - setStartVoltageVsOCP, 145
  - setVStep, 145
- AisDiffPulseVoltammetryElement.h, 270
- AisEISGalvanostaticElement, 146
  - AisEISGalvanostaticElement, 147
  - getAmplitude, 148
  - getBiasCurrent, 148
  - getCategory, 148
  - getEndFreq, 148
  - getMinimumCycles, 148
  - getName, 149
  - getQuietTime, 149
  - getQuietTimeSamplingInterval, 149
  - getStartFreq, 149
  - getStepsPerDecade, 150
  - setAmplitude, 150
  - setBiasCurrent, 150
  - setEndFreq, 151
  - setMinimumCycles, 151
  - setQuietTime, 151
  - setQuietTimeSamplingInterval, 151
  - setStartFreq, 152
  - setStepsPerDecade, 152
- AisEISGalvanostaticElement.h, 271
- AisEISPotentiostaticElement, 152
  - AisEISPotentiostaticElement, 154
  - getAmplitude, 155
  - getBiasVoltage, 155
  - getCategory, 155
  - getEndFreq, 155
  - getMinimumCycles, 155
  - getName, 156
  - getQuietTime, 156
  - getQuietTimeSamplingInterval, 156
  - getStartFreq, 156
  - getStepsPerDecade, 157
  - isBiasVoltageVsOCP, 157
  - setAmplitude, 157
  - setBiasVoltage, 158
  - setBiasVoltageVsOCP, 158
  - setEndFreq, 158
  - setMinimumCycles, 158
  - setQuietTime, 159
  - setQuietTimeSamplingInterval, 159
  - setStartFreq, 159
  - setStepsPerDecade, 160
- AisEISPotentiostaticElement.h, 272
- AisErrorCode, 160
  - BusyChannel, 161
  - ChannelNotBusy, 162
  - ConnectionFailed, 161
  - DeviceCommunicationFailed, 162
  - DeviceNotFound, 161
  - ErrorCode, 161
  - ExperimentAlreadyPaused, 162
  - ExperimentAlreadyRun, 162
  - ExperimentIsEmpty, 161
  - ExperimentNotUploaded, 161
  - ExperimentUploaded, 162
  - FailedToPauseExperiment, 162
  - FailedToResumeExperiment, 162
  - FailedToSetCompRange, 162
  - FailedToSetIRComp, 162
  - FailedToSetManualModeConstantCurrent, 162

- FailedToSetManualModeConstantVoltage, 162
- FailedToSetManualModeCurrentRange, 162
- FailedToSetManualModeInOCP, 162
- FailedToSetManualModeSamplingInterval, 162
- FailedToStopExperiment, 162
- FailedToUploadExperiment, 162
- FirmwareNotSupported, 161
- InvalidChannel, 161
- InvalidParameters, 161
- ManualExperimentNotRunning, 161
- message, 162
- Success, 161
- Unknown, 161
- value, 162
- AisErrorCode.h, 259
- AisExperiment, 163
  - AisExperiment, 164
  - appendElement, 164
  - appendSubExperiment, 165
  - getCategory, 165
  - getDescription, 165
  - getExperimentName, 166
  - operator=, 166
  - setDescription, 166
  - setExperimentName, 167
- AisExperiment.h, 260
- AisExperimentNode, 167
- AisInstrumentHandler, 168
  - activeACDataReady, 170
  - activeDCDataReady, 171
  - deviceDisconnected, 171
  - deviceError, 171
  - eraseRecoverData, 171
  - experimentNewElementStarting, 172
  - experimentPaused, 172
  - experimentResumed, 172
  - experimentStopped, 173
  - getExperimentUTCStartTime, 173
  - getFreeChannels, 173
  - getLinkedChannels, 173
  - getManualModeCurrentRangeList, 174
  - getManualModeVoltageRangeList, 174
  - getNumberOfChannels, 175
  - groundFloatStateChanged, 175
  - hasBipolarMode, 175
  - idleDCDataReady, 176
  - isChannelBusy, 176
  - isChannelPaused, 176
  - pauseExperiment, 177
  - recoverDataErased, 177
  - recoveryACDataReady, 177
  - recoveryDCDataReady, 178
  - resumeExperiment, 178
  - setBipolarLinkedChannels, 179
  - setCompRange, 179
  - setIRComp, 180
  - setLinkedChannels, 180
  - setManualModeConstantCurrent, 181
  - setManualModeConstantVoltage, 181, 182
  - setManualModeCurrentAutorange, 182
  - setManualModeCurrentRange, 183
  - setManualModeOCP, 183
  - setManualModeSamplingInterval, 184
  - setManualModeVoltageAutorange, 184
  - setManualModeVoltageRange, 185
  - skipExperimentStep, 185
  - startIdleSampling, 186
  - startManualExperiment, 186
  - startUploadedExperiment, 187
  - stopExperiment, 187
  - uploadExperimentToChannel, 188
- AisInstrumentHandler.h, 261
- AisManipulatorType.h, 262
- AisMottSchottkyElement, 189
  - AisMottSchottkyElement, 192, 193
  - getAmplitude, 193
  - getCategory, 193
  - getEndFrequency, 193
  - getEndingPotential, 193
  - getMinCycles, 194
  - getName, 194
  - getQuietTime, 194
  - getQuietTimeSamplInterval, 194
  - getStartFrequency, 195
  - getStartingPotential, 195
  - getStepQuietSamplInterval, 195
  - getStepQuietTime, 195
  - getStepsPerDecade, 196
  - getVoltageStep, 196
  - isEndVoltageVsOCP, 196
  - isStartVoltageVsOCP, 196
  - operator=, 197
  - setAmplitude, 197
  - setEndFrequency, 197
  - setEndingPotential, 198
  - setEndVoltageVsOCP, 198
  - setMinCycles, 198
  - setQuietTime, 198
  - setQuietTimeSamplInterval, 199
  - setStartFrequency, 199
  - setStartingPotential, 199
  - setStartVoltageVsOCP, 200
  - setStepQuietSamplInterval, 200
  - setStepQuietTime, 200
  - setStepsPerDecade, 200
  - setVoltageStep, 201
- AisMottSchottkyElement.h, 272
- AisNormalPulseVoltammetryElement, 201
  - AisNormalPulseVoltammetryElement, 203
  - getAlphaFactor, 204
  - getApproxMaxCurrent, 204
  - getCategory, 204
  - getEndVoltage, 205
  - getName, 205
  - getPulsePeriod, 205
  - getPulseWidth, 205



- getQuietTime, 206
- getQuietTimeSamplingInterval, 206
- getStartVoltage, 206
- getVStep, 206
- isAutoRange, 207
- isEndVoltageVsOCP, 207
- isStartVoltageVsOCP, 207
- setAlphaFactor, 208
- setApproxMaxCurrent, 208
- setAutoRange, 209
- setEndVoltage, 209
- setEndVoltageVsOCP, 209
- setPulsePeriod, 209
- setPulseWidth, 210
- setQuietTime, 210
- setQuietTimeSamplingInterval, 210
- setStartVoltage, 211
- setStartVoltageVsOCP, 211
- setVStep, 211
- AisNormalPulseVoltammetryElement.h, 273
- AisOpenCircuitElement, 212
  - AisOpenCircuitElement, 213
  - getApproxMaxVoltage, 214
  - getCategory, 214
  - getMaxDuration, 214
  - getMaxVoltage, 214
  - getMindVdt, 215
  - getMinVoltage, 215
  - getName, 215
  - getSamplingInterval, 216
  - isAutoVoltageRange, 216
  - setApproxMaxVoltage, 216
  - setAutoVoltageRange, 217
  - setMaxDuration, 217
  - setMaxVoltage, 217
  - setMindVdt, 217
  - setMinVoltage, 218
  - setSamplingInterval, 218
- AisOpenCircuitElement.h, 274
- AisSquareWaveVoltammetryElement, 218
  - AisSquareWaveVoltammetryElement, 221
  - getAlphaFactor, 221
  - getApproxMaxCurrent, 221
  - getCategory, 222
  - getEndVoltage, 222
  - getName, 222
  - getPulseAmp, 222
  - getPulseFreq, 223
  - getQuietTime, 223
  - getQuietTimeSamplingInterval, 223
  - getStartVoltage, 223
  - getVStep, 224
  - isAutoRange, 224
  - isEndVoltageVsOCP, 224
  - isStartVoltageVsOCP, 224
  - setAlphaFactor, 225
  - setApproxMaxCurrent, 225
  - setAutoRange, 226
  - setEndVoltage, 226
  - setEndVoltageVsOCP, 226
  - setPulseAmp, 226
  - setPulseFreq, 227
  - setQuietTime, 227
  - setQuietTimeSamplingInterval, 227
  - setStartVoltage, 228
  - setStartVoltageVsOCP, 228
  - setVStep, 228
- AisSquareWaveVoltammetryElement.h, 275
- AisSquidstatGlobal.h, 262
- AisStaircasePotentialVoltammetryElement, 229
  - AisStaircasePotentialVoltammetryElement, 231, 232
  - getApproxMaxCurrent, 232
  - getCategory, 232
  - getEndVoltage, 232
  - getFirstVoltageLimit, 232
  - getName, 233
  - getNumberOfCycles, 233
  - getQuietTime, 233
  - getQuietTimeSamplingInterval, 233
  - getSamplingInterval, 234
  - getSecondVoltageLimit, 234
  - getStartVoltage, 234
  - getStepDuration, 234
  - getStepSize, 235
  - isAutorange, 235
  - isEndVoltageVsOCP, 235
  - isFirstVoltageLimitVsOCP, 235
  - isSecondVoltageLimitVsOCP, 236
  - isStartVoltageVsOCP, 236
  - operator=, 236
  - setApproxMaxCurrent, 237
  - setEndVoltage, 237
  - setEndVoltageVsOCP, 237
  - setFirstVoltageLimit, 237
  - setFirstVoltageLimitVsOCP, 238
  - setNumberOfCycles, 238
  - setQuietTime, 238
  - setQuietTimeSamplingInterval, 239
  - setSamplingInterval, 239
  - setSecondVoltageLimit, 239
  - setSecondVoltageLimitVsOCP, 239
  - setStartVoltage, 240
  - setStartVoltageVsOCP, 240
  - setStepDuration, 240
  - setStepSize, 240
- AisStaircasePotentialVoltammetryElement.h, 276
- AisSteppedCurrentElement, 241
  - AisSteppedCurrentElement, 242
  - getApproxMaxVoltage, 243
  - getApproxMinVoltage, 243
  - getCategory, 243
  - getEndCurrent, 244
  - getName, 244
  - getSamplingInterval, 244
  - getStartCurrent, 244

- getStepDuration, 245
- getStepSize, 245
- setApproxMaxVoltage, 245
- setApproxMinVoltage, 246
- setEndCurrent, 246
- setSamplingInterval, 246
- setStartCurrent, 246
- setStepDuration, 248
- setStepSize, 248
- AisSteppedCurrentElement.h, 277
- AisSteppedVoltageElement, 248
  - AisSteppedVoltageElement, 250, 251
  - getApproxMaxCurrent, 251
  - getCategory, 251
  - getEndVoltage, 251
  - getName, 251
  - getSamplingInterval, 252
  - getStartVoltage, 252
  - getStepDuration, 252
  - getStepSize, 252
  - isAutoRange, 253
  - isEndVoltageVsOCP, 253
  - isStartVoltageVsOCP, 253
  - operator=, 253
  - setApproxMaxCurrent, 254
  - setEndVoltage, 254
  - setEndVoltageVsOCP, 254
  - setSamplingInterval, 255
  - setStartVoltage, 255
  - setStartVoltageVsOCP, 255
  - setStepDuration, 255
  - setStepSize, 256
- AisSteppedVoltageElement.h, 278
- appendElement
  - AisExperiment, 164
- appendSubExperiment
  - AisExperiment, 165
- BusyChannel
  - AisErrorCode, 161
- ChannelNotBusy
  - AisErrorCode, 162
- connectAllPluggedInDevices
  - AisDeviceTracker, 129
- ConnectionFailed
  - AisErrorCode, 161
- connectToDeviceOnComPort
  - AisDeviceTracker, 129
- DeviceCommunicationFailed
  - AisErrorCode, 162
- deviceDisconnected
  - AisDeviceTracker, 131
  - AisInstrumentHandler, 171
- deviceError
  - AisInstrumentHandler, 171
- DeviceNotFound
  - AisErrorCode, 161
- eraseRecoverData
  - AisInstrumentHandler, 171
- ErrorCode
  - AisErrorCode, 161
- ExperimentAlreadyPaused
  - AisErrorCode, 162
- ExperimentAlreadyRun
  - AisErrorCode, 162
- ExperimentIsEmpty
  - AisErrorCode, 161
- experimentNewElementStarting
  - AisInstrumentHandler, 172
- ExperimentNotUploaded
  - AisErrorCode, 161
- experimentPaused
  - AisInstrumentHandler, 172
- experimentResumed
  - AisInstrumentHandler, 172
- experimentStopped
  - AisInstrumentHandler, 173
- ExperimentUploaded
  - AisErrorCode, 162
- FailedToPauseExperiment
  - AisErrorCode, 162
- FailedToResumeExperiment
  - AisErrorCode, 162
- FailedToSetCompRange
  - AisErrorCode, 162
- FailedToSetIRComp
  - AisErrorCode, 162
- FailedToSetManualModeConstantCurrent
  - AisErrorCode, 162
- FailedToSetManualModeConstantVoltage
  - AisErrorCode, 162
- FailedToSetManualModeCurrentRange
  - AisErrorCode, 162
- FailedToSetManualModeInOCP
  - AisErrorCode, 162
- FailedToSetManualModeSamplingInterval
  - AisErrorCode, 162
- FailedToStopExperiment
  - AisErrorCode, 162
- FailedToUploadExperiment
  - AisErrorCode, 162
- FirmwareNotSupported
  - AisErrorCode, 161
- getAlphaFactor
  - AisCyclicVoltammetryElement, 93
  - AisDCCurrentSweepElement, 110
  - AisDCPotentialSweepElement, 120
  - AisDiffPulseVoltammetryElement, 137
  - AisNormalPulseVoltammetryElement, 204
  - AisSquareWaveVoltammetryElement, 221
- getAmplitude
  - AisEISGalvanostaticElement, 148
  - AisEISPotentiostaticElement, 155
  - AisMottSchottkyElement, 193

- getApproxMaxCurrent
  - AisConstantCurrentElement, 55
  - AisConstantPotElement, 65
  - AisCyclicVoltammetryElement, 93
  - AisDCPotentialSweepElement, 120
  - AisDiffPulseVoltammetryElement, 137
  - AisNormalPulseVoltammetryElement, 204
  - AisSquareWaveVoltammetryElement, 221
  - AisStaircasePotentialVoltammetryElement, 232
  - AisSteppedVoltageElement, 251
- getApproxMaxVoltage
  - AisConstantCurrentElement, 56
  - AisOpenCircuitElement, 214
  - AisSteppedCurrentElement, 243
- getApproxMinVoltage
  - AisSteppedCurrentElement, 243
- getBandwidthIndex
  - AisCompRange, 51
- getBaseCurrent
  - AisDataManipulator, 104
- getBaseVoltage
  - AisDataManipulator, 104
- getBiasCurrent
  - AisEISGalvanostaticElement, 148
- getBiasVoltage
  - AisEISPotentiostaticElement, 155
- getCategory
  - AisConstantCurrentElement, 56
  - AisConstantPotElement, 65
  - AisConstantPowerElement, 74
  - AisConstantResistanceElement, 84
  - AisCyclicVoltammetryElement, 94
  - AisDCCurrentSweepElement, 110
  - AisDCPotentialSweepElement, 120
  - AisDiffPulseVoltammetryElement, 138
  - AisEISGalvanostaticElement, 148
  - AisEISPotentiostaticElement, 155
  - AisExperiment, 165
  - AisMottSchottkyElement, 193
  - AisNormalPulseVoltammetryElement, 204
  - AisOpenCircuitElement, 214
  - AisSquareWaveVoltammetryElement, 222
  - AisStaircasePotentialVoltammetryElement, 232
  - AisSteppedCurrentElement, 243
  - AisSteppedVoltageElement, 251
- getCompRangeName
  - AisCompRange, 51
- getConnectedDevices
  - AisDeviceTracker, 131
- getCurrent
  - AisConstantCurrentElement, 56
- getdEdt
  - AisCyclicVoltammetryElement, 94
- getDescription
  - AisExperiment, 165
- getEndCurrent
  - AisSteppedCurrentElement, 244
- getEndFreq
  - AisEISGalvanostaticElement, 148
  - AisEISPotentiostaticElement, 155
- getEndFrequency
  - AisMottSchottkyElement, 193
- getEndingCurrent
  - AisDCCurrentSweepElement, 110
- getEndingPot
  - AisDCPotentialSweepElement, 120
- getEndingPotential
  - AisMottSchottkyElement, 193
- getEndVoltage
  - AisCyclicVoltammetryElement, 94
  - AisDiffPulseVoltammetryElement, 138
  - AisNormalPulseVoltammetryElement, 205
  - AisSquareWaveVoltammetryElement, 222
  - AisStaircasePotentialVoltammetryElement, 232
  - AisSteppedVoltageElement, 251
- getExperimentName
  - AisExperiment, 166
- getExperimentUTCStartTime
  - AisInstrumentHandler, 173
- getFirstVoltageLimit
  - AisCyclicVoltammetryElement, 94
  - AisStaircasePotentialVoltammetryElement, 232
- getFreeChannels
  - AisInstrumentHandler, 173
- getFrequency
  - AisDataManipulator, 104
- getInstrumentHandler
  - AisDeviceTracker, 131
- getLinkedChannels
  - AisInstrumentHandler, 173
- getManualModeCurrentRangeList
  - AisInstrumentHandler, 174
- getManualModeVoltageRangeList
  - AisInstrumentHandler, 174
- getMaxAbsoluteCurrent
  - AisDCPotentialSweepElement, 121
- getMaxCapacity
  - AisConstantCurrentElement, 57
  - AisConstantPotElement, 65
  - AisConstantPowerElement, 74
  - AisConstantResistanceElement, 84
- getMaxCurrent
  - AisConstantPotElement, 66
  - AisConstantPowerElement, 74
  - AisConstantResistanceElement, 84
- getMaxDuration
  - AisConstantCurrentElement, 57
  - AisConstantPotElement, 66
  - AisConstantPowerElement, 75
  - AisConstantResistanceElement, 84
  - AisOpenCircuitElement, 214
- getMaxVoltage
  - AisConstantCurrentElement, 57
  - AisConstantPowerElement, 75
  - AisConstantResistanceElement, 85
  - AisDCCurrentSweepElement, 110

- AisOpenCircuitElement, 214
- getMinAbsoluteCurrent
  - AisDCPotentialSweepElement, 121
- getMinCurrent
  - AisConstantPotElement, 66
  - AisConstantPowerElement, 75
  - AisConstantResistanceElement, 85
- getMinCycles
  - AisMottSchottkyElement, 194
- getMindIdt
  - AisConstantPotElement, 67
- getMindVdt
  - AisOpenCircuitElement, 215
- getMinimumCycles
  - AisEISGalvanostaticElement, 148
  - AisEISPotentiostaticElement, 155
- getMinSamplingVoltageDifference
  - AisConstantCurrentElement, 57
- getMinVoltage
  - AisConstantCurrentElement, 58
  - AisConstantPowerElement, 76
  - AisConstantResistanceElement, 85
  - AisDCCurrentSweepElement, 111
  - AisOpenCircuitElement, 215
- getName
  - AisConstantCurrentElement, 58
  - AisConstantPotElement, 67
  - AisConstantPowerElement, 76
  - AisConstantResistanceElement, 86
  - AisCyclicVoltammetryElement, 95
  - AisDCCurrentSweepElement, 111
  - AisDCPotentialSweepElement, 121
  - AisDiffPulseVoltammetryElement, 138
  - AisEISGalvanostaticElement, 149
  - AisEISPotentiostaticElement, 156
  - AisMottSchottkyElement, 194
  - AisNormalPulseVoltammetryElement, 205
  - AisOpenCircuitElement, 215
  - AisSquareWaveVoltammetryElement, 222
  - AisStaircasePotentialVoltammetryElement, 233
  - AisSteppedCurrentElement, 244
  - AisSteppedVoltageElement, 251
- getNumberOfChannels
  - AisInstrumentHandler, 175
- getNumberOfCycles
  - AisCyclicVoltammetryElement, 95
  - AisStaircasePotentialVoltammetryElement, 233
- getPotential
  - AisConstantPotElement, 67
- getPower
  - AisConstantPowerElement, 76
- getPulseAmp
  - AisSquareWaveVoltammetryElement, 222
- getPulseCurrent
  - AisDataManipulator, 105
- getPulseFreq
  - AisSquareWaveVoltammetryElement, 223
- getPulseHeight
  - AisDiffPulseVoltammetryElement, 138
- getPulsePeriod
  - AisDataManipulator, 105
  - AisDiffPulseVoltammetryElement, 139
  - AisNormalPulseVoltammetryElement, 205
- getPulseVoltage
  - AisDataManipulator, 105
- getPulseWidth
  - AisDataManipulator, 105
  - AisDiffPulseVoltammetryElement, 139
  - AisNormalPulseVoltammetryElement, 205
- getQuietTime
  - AisCyclicVoltammetryElement, 95
  - AisDCCurrentSweepElement, 111
  - AisDCPotentialSweepElement, 122
  - AisDiffPulseVoltammetryElement, 139
  - AisEISGalvanostaticElement, 149
  - AisEISPotentiostaticElement, 156
  - AisMottSchottkyElement, 194
  - AisNormalPulseVoltammetryElement, 206
  - AisSquareWaveVoltammetryElement, 223
  - AisStaircasePotentialVoltammetryElement, 233
- getQuietTimeSamplInterval
  - AisMottSchottkyElement, 194
- getQuietTimeSamplingInterval
  - AisCyclicVoltammetryElement, 95
  - AisDCCurrentSweepElement, 112
  - AisDCPotentialSweepElement, 122
  - AisDiffPulseVoltammetryElement, 140
  - AisEISGalvanostaticElement, 149
  - AisEISPotentiostaticElement, 156
  - AisNormalPulseVoltammetryElement, 206
  - AisSquareWaveVoltammetryElement, 223
  - AisStaircasePotentialVoltammetryElement, 233
- getResistance
  - AisConstantResistanceElement, 86
- getSamplingInterval
  - AisConstantCurrentElement, 58
  - AisConstantPotElement, 68
  - AisConstantPowerElement, 77
  - AisConstantResistanceElement, 86
  - AisCyclicVoltammetryElement, 96
  - AisDCCurrentSweepElement, 112
  - AisDCPotentialSweepElement, 122
  - AisOpenCircuitElement, 216
  - AisStaircasePotentialVoltammetryElement, 234
  - AisSteppedCurrentElement, 244
  - AisSteppedVoltageElement, 252
- getScanRate
  - AisDCCurrentSweepElement, 112
  - AisDCPotentialSweepElement, 122
- getSecondVoltageLimit
  - AisCyclicVoltammetryElement, 96
  - AisStaircasePotentialVoltammetryElement, 234
- getStabilityFactor
  - AisCompRange, 52
- getStartCurrent
  - AisSteppedCurrentElement, 244

- getStartFreq
  - AisEISGalvanostaticElement, 149
  - AisEISPotentiostaticElement, 156
- getStartFrequency
  - AisMottSchottkyElement, 195
- getStartingCurrent
  - AisDCCurrentSweepElement, 112
- getStartingPot
  - AisDCPotentialSweepElement, 123
- getStartingPotential
  - AisMottSchottkyElement, 195
- getStartVoltage
  - AisCyclicVoltammetryElement, 96
  - AisDiffPulseVoltammetryElement, 140
  - AisNormalPulseVoltammetryElement, 206
  - AisSquareWaveVoltammetryElement, 223
  - AisStaircasePotentialVoltammetryElement, 234
  - AisSteppedVoltageElement, 252
- getStepDuration
  - AisStaircasePotentialVoltammetryElement, 234
  - AisSteppedCurrentElement, 245
  - AisSteppedVoltageElement, 252
- getStepQuietSamplInterval
  - AisMottSchottkyElement, 195
- getStepQuietTime
  - AisMottSchottkyElement, 195
- getStepSize
  - AisStaircasePotentialVoltammetryElement, 235
  - AisSteppedCurrentElement, 245
  - AisSteppedVoltageElement, 252
- getStepsPerDecade
  - AisEISGalvanostaticElement, 150
  - AisEISPotentiostaticElement, 157
  - AisMottSchottkyElement, 196
- getVoltageStep
  - AisMottSchottkyElement, 196
- getVStep
  - AisDiffPulseVoltammetryElement, 140
  - AisNormalPulseVoltammetryElement, 206
  - AisSquareWaveVoltammetryElement, 224
- groundFloatStateChanged
  - AisInstrumentHandler, 175
- hasBipolarMode
  - AisInstrumentHandler, 175
- idleDCDataReady
  - AisInstrumentHandler, 176
- InvalidChannel
  - AisErrorCode, 161
- InvalidParameters
  - AisErrorCode, 161
- isAutoRange
  - AisConstantCurrentElement, 59
  - AisConstantPotElement, 68
  - AisCyclicVoltammetryElement, 96
  - AisDCPotentialSweepElement, 123
  - AisDiffPulseVoltammetryElement, 140
  - AisNormalPulseVoltammetryElement, 207
  - AisSquareWaveVoltammetryElement, 224
  - AisStaircasePotentialVoltammetryElement, 235
  - AisSteppedVoltageElement, 253
- isAutorange
  - AisStaircasePotentialVoltammetryElement, 235
- isAutoVoltageRange
  - AisConstantCurrentElement, 59
  - AisOpenCircuitElement, 216
- isBiasVoltageVsOCP
  - AisEISPotentiostaticElement, 157
- isChannelBusy
  - AisInstrumentHandler, 176
- isChannelPaused
  - AisInstrumentHandler, 176
- isCharge
  - AisConstantPowerElement, 77
- isEndVoltageVsOCP
  - AisCyclicVoltammetryElement, 97
  - AisDCPotentialSweepElement, 123
  - AisDiffPulseVoltammetryElement, 141
  - AisMottSchottkyElement, 196
  - AisNormalPulseVoltammetryElement, 207
  - AisSquareWaveVoltammetryElement, 224
  - AisStaircasePotentialVoltammetryElement, 235
  - AisSteppedVoltageElement, 253
- isFirstVoltageLimitVsOCP
  - AisCyclicVoltammetryElement, 97
  - AisStaircasePotentialVoltammetryElement, 235
- isMaximumVoltageVsOCP
  - AisConstantPowerElement, 77
  - AisConstantResistanceElement, 86
- isMinimumVoltageVsOCP
  - AisConstantPowerElement, 77
  - AisConstantResistanceElement, 87
- isPulseCompleted
  - AisDataManipulator, 106
- isSecondVoltageLimitVsOCP
  - AisCyclicVoltammetryElement, 97
  - AisStaircasePotentialVoltammetryElement, 236
- isStartVoltageVsOCP
  - AisCyclicVoltammetryElement, 98
  - AisDCPotentialSweepElement, 123
  - AisDiffPulseVoltammetryElement, 141
  - AisMottSchottkyElement, 196
  - AisNormalPulseVoltammetryElement, 207
  - AisSquareWaveVoltammetryElement, 224
  - AisStaircasePotentialVoltammetryElement, 236
  - AisSteppedVoltageElement, 253
- isVoltageVsOCP
  - AisConstantPotElement, 68
- loadPrimaryData
  - AisDataManipulator, 106
- ManualExperimentNotRunning
  - AisErrorCode, 161
- message
  - AisErrorCode, 162
- newDeviceConnected

- AisDeviceTracker, 132
- numberOfCycles
  - AisACData, 50
- operator=
  - AisExperiment, 166
  - AisMottSchottkyElement, 197
  - AisStaircasePotentialVoltammetryElement, 236
  - AisSteppedVoltageElement, 253
- pauseExperiment
  - AisInstrumentHandler, 177
- recoverDataErased
  - AisInstrumentHandler, 177
- recoveryACDataReady
  - AisInstrumentHandler, 177
- recoveryDCDataReady
  - AisInstrumentHandler, 178
- resumeExperiment
  - AisInstrumentHandler, 178
- saveLogToFile
  - AisDeviceTracker, 132
- setAlphaFactor
  - AisCyclicVoltammetryElement, 98
  - AisDCCurrentSweepElement, 113
  - AisDCPotentialSweepElement, 124
  - AisDiffPulseVoltammetryElement, 141
  - AisNormalPulseVoltammetryElement, 208
  - AisSquareWaveVoltammetryElement, 225
- setAmplitude
  - AisEISGalvanostaticElement, 150
  - AisEISPotentiostaticElement, 157
  - AisMottSchottkyElement, 197
- setApproxMaxCurrent
  - AisConstantCurrentElement, 59
  - AisConstantPotElement, 68
  - AisCyclicVoltammetryElement, 98
  - AisDCPotentialSweepElement, 124
  - AisDiffPulseVoltammetryElement, 142
  - AisNormalPulseVoltammetryElement, 208
  - AisSquareWaveVoltammetryElement, 225
  - AisStaircasePotentialVoltammetryElement, 237
  - AisSteppedVoltageElement, 254
- setApproxMaxVoltage
  - AisConstantCurrentElement, 60
  - AisOpenCircuitElement, 216
  - AisSteppedCurrentElement, 245
- setApproxMinVoltage
  - AisSteppedCurrentElement, 246
- setAutoRange
  - AisConstantCurrentElement, 60
  - AisConstantPotElement, 69
  - AisCyclicVoltammetryElement, 99
  - AisDCPotentialSweepElement, 124
  - AisDiffPulseVoltammetryElement, 142
  - AisNormalPulseVoltammetryElement, 209
  - AisSquareWaveVoltammetryElement, 226
- setAutoVoltageRange
  - AisConstantCurrentElement, 60
  - AisOpenCircuitElement, 217
- setBandwidthIndex
  - AisCompRange, 52
- setBiasCurrent
  - AisEISGalvanostaticElement, 150
- setBiasVoltage
  - AisEISPotentiostaticElement, 158
- setBiasVoltageVsOCP
  - AisEISPotentiostaticElement, 158
- setBipolarLinkedChannels
  - AisInstrumentHandler, 179
- setCharge
  - AisConstantPowerElement, 78
- setCompRange
  - AisInstrumentHandler, 179
- setCompRangeName
  - AisCompRange, 52
- setCurrent
  - AisConstantCurrentElement, 60
- setdEdt
  - AisCyclicVoltammetryElement, 99
- setDescription
  - AisExperiment, 166
- setEndCurrent
  - AisSteppedCurrentElement, 246
- setEndFreq
  - AisEISGalvanostaticElement, 151
  - AisEISPotentiostaticElement, 158
- setEndFrequency
  - AisMottSchottkyElement, 197
- setEndingCurrent
  - AisDCCurrentSweepElement, 113
- setEndingPot
  - AisDCPotentialSweepElement, 125
- setEndingPotential
  - AisMottSchottkyElement, 198
- setEndVoltage
  - AisCyclicVoltammetryElement, 99
  - AisDiffPulseVoltammetryElement, 142
  - AisNormalPulseVoltammetryElement, 209
  - AisSquareWaveVoltammetryElement, 226
  - AisStaircasePotentialVoltammetryElement, 237
  - AisSteppedVoltageElement, 254
- setEndVoltageVsOCP
  - AisCyclicVoltammetryElement, 99
  - AisDCPotentialSweepElement, 125
  - AisDiffPulseVoltammetryElement, 142
  - AisMottSchottkyElement, 198
  - AisNormalPulseVoltammetryElement, 209
  - AisSquareWaveVoltammetryElement, 226
  - AisStaircasePotentialVoltammetryElement, 237
  - AisSteppedVoltageElement, 254
- setExperimentName
  - AisExperiment, 167
- setFirstVoltageLimit
  - AisCyclicVoltammetryElement, 100



- AisStaircasePotentialVoltammetryElement, 237
- setFirstVoltageLimitVsOCP
  - AisCyclicVoltammetryElement, 100
  - AisStaircasePotentialVoltammetryElement, 238
- setIRComp
  - AisInstrumentHandler, 180
- setLinkedChannels
  - AisInstrumentHandler, 180
- setLogFilePath
  - AisDeviceTracker, 133
- setManualModeConstantCurrent
  - AisInstrumentHandler, 181
- setManualModeConstantVoltage
  - AisInstrumentHandler, 181, 182
- setManualModeCurrentAutorange
  - AisInstrumentHandler, 182
- setManualModeCurrentRange
  - AisInstrumentHandler, 183
- setManualModeOCP
  - AisInstrumentHandler, 183
- setManualModeSamplingInterval
  - AisInstrumentHandler, 184
- setManualModeVoltageAutorange
  - AisInstrumentHandler, 184
- setManualModeVoltageRange
  - AisInstrumentHandler, 185
- setMaxAbsoluteCurrent
  - AisDCPotentialSweepElement, 125
- setMaxCapacity
  - AisConstantCurrentElement, 60
  - AisConstantPotElement, 69
  - AisConstantPowerElement, 78
  - AisConstantResistanceElement, 87
- setMaxCurrent
  - AisConstantPotElement, 69
  - AisConstantPowerElement, 78
  - AisConstantResistanceElement, 88
- setMaxDuration
  - AisConstantCurrentElement, 61
  - AisConstantPotElement, 69
  - AisConstantPowerElement, 79
  - AisConstantResistanceElement, 88
  - AisOpenCircuitElement, 217
- setMaximumVoltageVsOCP
  - AisConstantPowerElement, 79
  - AisConstantResistanceElement, 88
- setMaxVoltage
  - AisConstantCurrentElement, 61
  - AisConstantPowerElement, 79
  - AisConstantResistanceElement, 89
  - AisDCCurrentSweepElement, 113
  - AisOpenCircuitElement, 217
- setMinAbsoluteCurrent
  - AisDCPotentialSweepElement, 126
- setMinCurrent
  - AisConstantPotElement, 70
  - AisConstantPowerElement, 80
  - AisConstantResistanceElement, 89
- setMinCycles
  - AisMottSchottkyElement, 198
- setMindIdt
  - AisConstantPotElement, 70
- setMindVdt
  - AisOpenCircuitElement, 217
- setMinimumCycles
  - AisEISGalvanostaticElement, 151
  - AisEISPotentiostaticElement, 158
- setMinimumVoltageVsOCP
  - AisConstantPowerElement, 80
  - AisConstantResistanceElement, 89
- setMinSamplingVoltageDifference
  - AisConstantCurrentElement, 61
- setMinVoltage
  - AisConstantCurrentElement, 62
  - AisConstantPowerElement, 80
  - AisConstantResistanceElement, 90
  - AisDCCurrentSweepElement, 115
  - AisOpenCircuitElement, 218
- setNumberOfCycles
  - AisCyclicVoltammetryElement, 100
  - AisStaircasePotentialVoltammetryElement, 238
- setPotential
  - AisConstantPotElement, 70
- setPower
  - AisConstantPowerElement, 81
- setPulseAmp
  - AisSquareWaveVoltammetryElement, 226
- setPulseFreq
  - AisSquareWaveVoltammetryElement, 227
- setPulseHeight
  - AisDiffPulseVoltammetryElement, 143
- setPulsePeriod
  - AisDiffPulseVoltammetryElement, 143
  - AisNormalPulseVoltammetryElement, 209
- setPulseType
  - AisDataManipulator, 106, 107
- setPulseWidth
  - AisDiffPulseVoltammetryElement, 143
  - AisNormalPulseVoltammetryElement, 210
- setQuietTime
  - AisCyclicVoltammetryElement, 101
  - AisDCCurrentSweepElement, 115
  - AisDCPotentialSweepElement, 126
  - AisDiffPulseVoltammetryElement, 144
  - AisEISGalvanostaticElement, 151
  - AisEISPotentiostaticElement, 159
  - AisMottSchottkyElement, 198
  - AisNormalPulseVoltammetryElement, 210
  - AisSquareWaveVoltammetryElement, 227
  - AisStaircasePotentialVoltammetryElement, 238
- setQuietTimeSamplInterval
  - AisMottSchottkyElement, 199
- setQuietTimeSamplingInterval
  - AisCyclicVoltammetryElement, 101
  - AisDCCurrentSweepElement, 115
  - AisDCPotentialSweepElement, 126

- AisDiffPulseVoltammetryElement, 144
- AisEISGalvanostaticElement, 151
- AisEISPotentiostaticElement, 159
- AisNormalPulseVoltammetryElement, 210
- AisSquareWaveVoltammetryElement, 227
- AisStaircasePotentialVoltammetryElement, 239
- setResistance
  - AisConstantResistanceElement, 90
- setSamplingInterval
  - AisConstantCurrentElement, 62
  - AisConstantPotElement, 71
  - AisConstantPowerElement, 81
  - AisConstantResistanceElement, 90
  - AisCyclicVoltammetryElement, 101
  - AisDCCurrentSweepElement, 116
  - AisDCPotentialSweepElement, 127
  - AisOpenCircuitElement, 218
  - AisStaircasePotentialVoltammetryElement, 239
  - AisSteppedCurrentElement, 246
  - AisSteppedVoltageElement, 255
- setScanRate
  - AisDCCurrentSweepElement, 116
  - AisDCPotentialSweepElement, 127
- setSecondVoltageLimit
  - AisCyclicVoltammetryElement, 102
  - AisStaircasePotentialVoltammetryElement, 239
- setSecondVoltageLimitVsOCP
  - AisCyclicVoltammetryElement, 102
  - AisStaircasePotentialVoltammetryElement, 239
- setStabilityFactor
  - AisCompRange, 53
- setStartCurrent
  - AisSteppedCurrentElement, 246
- setStartFreq
  - AisEISGalvanostaticElement, 152
  - AisEISPotentiostaticElement, 159
- setStartFrequency
  - AisMottSchottkyElement, 199
- setStartingCurrent
  - AisDCCurrentSweepElement, 116
- setStartingPot
  - AisDCPotentialSweepElement, 127
- setStartingPotential
  - AisMottSchottkyElement, 199
- setStartVoltage
  - AisCyclicVoltammetryElement, 102
  - AisDiffPulseVoltammetryElement, 144
  - AisNormalPulseVoltammetryElement, 211
  - AisSquareWaveVoltammetryElement, 228
  - AisStaircasePotentialVoltammetryElement, 240
  - AisSteppedVoltageElement, 255
- setStartVoltageVsOCP
  - AisCyclicVoltammetryElement, 103
  - AisDCPotentialSweepElement, 127
  - AisDiffPulseVoltammetryElement, 145
  - AisMottSchottkyElement, 200
  - AisNormalPulseVoltammetryElement, 211
  - AisSquareWaveVoltammetryElement, 228
- AisStaircasePotentialVoltammetryElement, 240
- AisSteppedVoltageElement, 255
- setStepDuration
  - AisStaircasePotentialVoltammetryElement, 240
  - AisSteppedCurrentElement, 248
  - AisSteppedVoltageElement, 255
- setStepQuietSampleInterval
  - AisMottSchottkyElement, 200
- setStepQuietTime
  - AisMottSchottkyElement, 200
- setStepSize
  - AisStaircasePotentialVoltammetryElement, 240
  - AisSteppedCurrentElement, 248
  - AisSteppedVoltageElement, 256
- setStepsPerDecade
  - AisEISGalvanostaticElement, 152
  - AisEISPotentiostaticElement, 160
  - AisMottSchottkyElement, 200
- setVoltageRange
  - AisConstantPotElement, 71
- setVoltageStep
  - AisMottSchottkyElement, 201
- setVoltageVsOCP
  - AisConstantPotElement, 71
- setVStep
  - AisDiffPulseVoltammetryElement, 145
  - AisNormalPulseVoltammetryElement, 211
  - AisSquareWaveVoltammetryElement, 228
- skipExperimentStep
  - AisInstrumentHandler, 185
- startIdleSampling
  - AisInstrumentHandler, 186
- startManualExperiment
  - AisInstrumentHandler, 186
- startUploadedExperiment
  - AisInstrumentHandler, 187
- stopExperiment
  - AisInstrumentHandler, 187
- Success
  - AisErrorCode, 161
- Unknown
  - AisErrorCode, 161
- updateFirmwareOnAllAvailableDevices
  - AisDeviceTracker, 133
- updateFirmwareOnComPort
  - AisDeviceTracker, 134
- uploadExperimentToChannel
  - AisInstrumentHandler, 188
- value
  - AisErrorCode, 162