

AUBO SDK

0.26.0

Generated by Doxygen 1.16.1

1 Aubo SDK	1
2 Topic Index	5
2.1 Topics	5
3 Directory Hierarchy	7
3.1 Directories	7
4 Namespace Index	9
4.1 Namespace List	9
5 Hierarchical Index	11
5.1 Class Hierarchy	11
6 Class Index	13
6.1 Class List	13
7 File Index	15
7.1 File List	15
8 Topic Documentation	17
8.1 AuboApi ()	17
8.1.1 Detailed Description	18
8.1.2 Function Documentation	18
8.1.2.1 getAxisInterface()	18
8.1.2.2 getAxisNames()	18
8.1.2.3 getGripperInterface()	18
8.1.2.4 getMath()	19
8.1.2.5 getRegisterControl()	19
8.1.2.6 getRobotInterface()	19
8.1.2.7 getRobotNames()	20
8.1.2.8 getRuntimeMachine()	21
8.1.2.9 getSerial()	21
8.1.2.10 getSocket()	21
8.1.2.11 getSyncMove()	22
8.1.2.12 getSystemInfo()	22
8.1.2.13 getTrace()	22
8.2 AxisInterface ()	22
8.2.1 Detailed Description	24
8.2.2 Function Documentation	24
8.2.2.1 clearAxisError()	24
8.2.2.2 enableExtAxis()	24
8.2.2.3 followAnotherAxis()	24
8.2.2.4 getAxisModeType()	25
8.2.2.5 getErrorCode()	25
8.2.2.6 getExtAxisAcceleration()	25

8.2.2.7	<code>getExtAxisBusCurrent()</code>	25
8.2.2.8	<code>getExtAxisBusVoltage()</code>	26
8.2.2.9	<code>getExtAxisCurrent()</code>	26
8.2.2.10	<code>getExtAxisMaxAcceleration()</code>	26
8.2.2.11	<code>getExtAxisMaxPosition()</code>	26
8.2.2.12	<code>getExtAxisMaxVelocity()</code>	26
8.2.2.13	<code>getExtAxisMountingPose()</code>	27
8.2.2.14	<code>getExtAxisPose()</code>	27
8.2.2.15	<code>getExtAxisPosition()</code>	27
8.2.2.16	<code>getExtAxisTemperature()</code>	27
8.2.2.17	<code>getExtAxisType()</code>	27
8.2.2.18	<code>getExtAxisVelocity()</code>	28
8.2.2.19	<code>getExtMinPosition()</code>	28
8.2.2.20	<code>moveExtJoint()</code>	28
8.2.2.21	<code>poweroffExtAxis()</code>	29
8.2.2.22	<code>poweronExtAxis()</code>	29
8.2.2.23	<code>setExtAxisMountingPose()</code>	29
8.2.2.24	<code>speedExtJoint()</code>	29
8.2.2.25	<code>stopExtJoint()</code>	30
8.2.2.26	<code>stopFollowAnotherAxis()</code>	30
8.3	<code>GripperInterface</code> ()	31
8.3.1	Detailed Description	31
8.3.2	Function Documentation	31
8.3.2.1	<code>gripperAdd()</code>	31
8.3.2.2	<code>gripperConnect()</code>	31
8.3.2.3	<code>gripperDelete()</code>	32
8.3.2.4	<code>gripperDisconnect()</code>	32
8.3.2.5	<code>gripperEnable()</code>	32
8.3.2.6	<code>gripperGetHardwareVersion()</code>	32
8.3.2.7	<code>gripperGetNames()</code>	32
8.3.2.8	<code>gripperGetStatusCode()</code>	32
8.3.2.9	<code>gripperGetSupportedModels()</code>	32
8.3.2.10	<code>gripperGetWorkMode()</code>	32
8.3.2.11	<code>gripperIsConnected()</code>	32
8.3.2.12	<code>gripperIsEnabled()</code>	33
8.3.2.13	<code>gripperMove()</code>	33
8.3.2.14	<code>gripperRename()</code>	33
8.3.2.15	<code>gripperResetSlaveId()</code>	33
8.3.2.16	<code>gripperScanDevices()</code>	33
8.3.2.17	<code>gripperSetMountPose()</code>	33
8.3.2.18	<code>gripperSetPosition()</code>	33
8.3.2.19	<code>gripperSetWorkMode()</code>	33
8.3.2.20	<code>gripperStop()</code>	34

8.4 Math ()	34
8.4.1 Detailed Description	35
8.4.2 Function Documentation	35
8.4.2.1 calculateCircleFourthPoint()	35
8.4.2.2 calibrateCoordinate()	36
8.4.2.3 changePoseWithXYRef()	37
8.4.2.4 deltaPoseAdd()	37
8.4.2.5 deltaPoseTrans()	38
8.4.2.6 forceTrans()	38
8.4.2.7 getDeltaPoseBySensorDistance()	39
8.4.2.8 homMatrixToPose()	39
8.4.2.9 interpolatePose()	40
8.4.2.10 poseAdd()	41
8.4.2.11 poseAngleDistance()	41
8.4.2.12 poseDistance()	42
8.4.2.13 poseEqual()	43
8.4.2.14 poseInverse()	43
8.4.2.15 poseRotation()	44
8.4.2.16 poseSub()	45
8.4.2.17 poseToHomMatrix()	45
8.4.2.18 poseTrans()	46
8.4.2.19 poseTransInv()	47
8.4.2.20 quaternionToRpy()	48
8.4.2.21 rpyToQuaternion()	48
8.4.2.22 tcpOffsetIdentify()	49
8.4.2.23 transferRefFrame()	50
8.5 RegisterControl ()	51
8.5.1 Detailed Description	54
8.5.2 Function Documentation	54
8.5.2.1 addInt32RegEncoder()	54
8.5.2.2 addModbusEncoder()	54
8.5.2.3 clearNamedVariable()	55
8.5.2.4 deleteVirtualEncoder()	55
8.5.2.5 getBool()	56
8.5.2.6 getBoolInput()	56
8.5.2.7 getBoolOutput()	57
8.5.2.8 getDouble()	58
8.5.2.9 getDoubleInput()	59
8.5.2.10 getDoubleOutput()	60
8.5.2.11 getFloat()	61
8.5.2.12 getFloatInput()	61
8.5.2.13 getFloatOutput()	62
8.5.2.14 getInt16Register()	63

8.5.2.15	getInt16RegisterBit()	64
8.5.2.16	getInt32()	65
8.5.2.17	getInt32Input()	66
8.5.2.18	getInt32Output()	66
8.5.2.19	getModbusDeviceStatus()	67
8.5.2.20	getNamedVariableType()	68
8.5.2.21	getString()	69
8.5.2.22	getVecChar()	69
8.5.2.23	getVecDouble()	70
8.5.2.24	getVecFloat()	71
8.5.2.25	getVecInt32()	72
8.5.2.26	getWatchDogAction()	73
8.5.2.27	getWatchDogTimeout()	73
8.5.2.28	hasNamedVariable()	73
8.5.2.29	modbusAddSignal()	74
8.5.2.30	modbusDeleteAllSignals()	76
8.5.2.31	modbusDeleteSignal()	76
8.5.2.32	modbusGetSignalError()	77
8.5.2.33	modbusGetSignalErrors()	77
8.5.2.34	modbusGetSignalIndex()	77
8.5.2.35	modbusGetSignalNames()	78
8.5.2.36	modbusGetSignalStatus()	78
8.5.2.37	modbusGetSignalTypes()	79
8.5.2.38	modbusGetSignalValues()	80
8.5.2.39	modbusSendCustomCommand()	80
8.5.2.40	modbusSetDigitalInputAction()	81
8.5.2.41	modbusSetOutputRunstate()	82
8.5.2.42	modbusSetOutputSignal()	83
8.5.2.43	modbusSetOutputSignal1()	84
8.5.2.44	modbusSetOutputSignalPulse()	85
8.5.2.45	modbusSetSignalUpdateFrequency()	85
8.5.2.46	setBool()	86
8.5.2.47	setBoolInput()	87
8.5.2.48	setBoolOutput()	88
8.5.2.49	setDouble()	89
8.5.2.50	setDoubleInput()	89
8.5.2.51	setDoubleOutput()	90
8.5.2.52	setFloat()	91
8.5.2.53	setFloatInput()	92
8.5.2.54	setFloatOutput()	93
8.5.2.55	setInt16Register()	94
8.5.2.56	setInt16RegisterBit()	94
8.5.2.57	setInt32()	95

8.5.2.58	setInt32Input()	96
8.5.2.59	setInt32Output()	97
8.5.2.60	setString()	98
8.5.2.61	setVecChar()	98
8.5.2.62	setVecDouble()	99
8.5.2.63	setVecFloat()	100
8.5.2.64	setVecInt32()	101
8.5.2.65	setWatchDog()	102
8.5.2.66	variableUpdated()	102
8.6	RobotInterface()	103
8.6.1	Detailed Description	104
8.6.2	Function Documentation	104
8.6.2.1	getForceControl()	104
8.6.2.2	getIoControl()	105
8.6.2.3	getMotionControl()	105
8.6.2.4	getRobotAlgorithm()	105
8.6.2.5	getRobotConfig()	106
8.6.2.6	getRobotManage()	106
8.6.2.7	getRobotState()	106
8.6.2.8	getSyncMove()	107
8.6.2.9	getTrace()	107
8.6.3	ForceControl()	107
8.6.3.1	Detailed Description	110
8.6.3.2	Function Documentation	110
8.6.4	IoControl (IO)	144
8.6.4.1	Detailed Description	148
8.6.4.2	Function Documentation	149
8.6.5	MotionControl()	219
8.6.5.1	Detailed Description	224
8.6.5.2	Function Documentation	225
8.6.6	RobotAlgorithm()	319
8.6.6.1	Detailed Description	321
8.6.6.2	Function Documentation	321
8.6.7	RobotConfig()	346
8.6.7.1	Detailed Description	350
8.6.7.2	Function Documentation	350
8.6.8	RobotManage()	404
8.6.8.1	Detailed Description	406
8.6.8.2	Function Documentation	406
8.6.9	RobotState()	432
8.6.9.1	Detailed Description	434
8.6.9.2	Function Documentation	435
8.7	RuntimeMachine()	480

8.7.1 Detailed Description	482
8.7.2 Function Documentation	482
8.7.2.1 abort()	482
8.7.2.2 arbitraryResume()	482
8.7.2.3 clearBreakPoints()	483
8.7.2.4 clearPreloadPrograms()	483
8.7.2.5 deleteTask()	483
8.7.2.6 detachTask()	483
8.7.2.7 enterCritical()	484
8.7.2.8 exitCritical()	484
8.7.2.9 getAdvancePlanContext()	485
8.7.2.10 getAdvancePtr()	486
8.7.2.11 getExecutionStatus()	486
8.7.2.12 getInterpPtr()	486
8.7.2.13 getMainPtr()	487
8.7.2.14 getPlanContext()	487
8.7.2.15 getPreloadProgram()	488
8.7.2.16 getStatus()	489
8.7.2.17 getTaskQueueSize()	489
8.7.2.18 getTimer()	489
8.7.2.19 getTriggInterrupts()	490
8.7.2.20 gotoLine()	490
8.7.2.21 isTaskAlive()	491
8.7.2.22 loadProgram()	491
8.7.2.23 newTask()	492
8.7.2.24 nop()	492
8.7.2.25 pause()	493
8.7.2.26 preloadProgram()	493
8.7.2.27 removeBreakPoint()	493
8.7.2.28 resume()	494
8.7.2.29 runProgram()	495
8.7.2.30 setBreakPoint()	495
8.7.2.31 setLabel()	495
8.7.2.32 setPlanContext()	496
8.7.2.33 setResumeWait()	497
8.7.2.34 start()	497
8.7.2.35 step()	498
8.7.2.36 stop()	498
8.7.2.37 switchTask()	498
8.7.2.38 timerDelete()	499
8.7.2.39 timerReset()	499
8.7.2.40 timerStart()	500
8.7.2.41 timerStop()	501

8.7.2.42	triggBegin()	501
8.7.2.43	triggEnd()	502
8.7.2.44	triggInterrupt()	502
8.8	Serial ()	503
8.8.1	Detailed Description	504
8.8.2	Function Documentation	504
8.8.2.1	serialClose()	504
8.8.2.2	serialOpen()	504
8.8.2.3	serialReadByte()	505
8.8.2.4	serialReadByteList()	506
8.8.2.5	serialReadString()	506
8.8.2.6	serialSendAllString()	507
8.8.2.7	serialSendByte()	508
8.8.2.8	serialSendInt()	508
8.8.2.9	serialSendLine()	509
8.8.2.10	serialSendString()	509
8.9	Socket (socket)	510
8.9.1	Detailed Description	511
8.9.2	Function Documentation	511
8.9.2.1	socketClose()	511
8.9.2.2	socketHasConnected()	512
8.9.2.3	socketOpen()	512
8.9.2.4	socketReadAllString()	513
8.9.2.5	socketReadAsciiFloat()	514
8.9.2.6	socketReadBinaryInteger()	515
8.9.2.7	socketReadByteList()	515
8.9.2.8	socketReadString()	516
8.9.2.9	socketSendAllString()	517
8.9.2.10	socketSendByte()	518
8.9.2.11	socketSendInt()	519
8.9.2.12	socketSendLine()	519
8.9.2.13	socketSendString()	520
8.10	SyncMove ()	521
8.10.1	Detailed Description	523
8.10.2	Function Documentation	523
8.10.2.1	axisGroupAdd()	523
8.10.2.2	axisGroupAddAxis()	523
8.10.2.3	axisGroupDelete()	524
8.10.2.4	axisGroupGetActualPositions()	524
8.10.2.5	axisGroupGetAxisIndex()	525
8.10.2.6	axisGroupGetAxisName()	525
8.10.2.7	axisGroupGetTargetPositions()	525
8.10.2.8	axisGroupMoveJoint()	526

8.10.2.9	axisGroupOffsetPositions()	526
8.10.2.10	axisGroupSpeedJoint()	527
8.10.2.11	axisGroupUpdateAxis()	527
8.10.2.12	frameAdd()	528
8.10.2.13	frameAttach()	528
8.10.2.14	frameConvertPose()	529
8.10.2.15	frameDelete()	529
8.10.2.16	frameDeleteAll()	530
8.10.2.17	frameExist()	530
8.10.2.18	frameGetChildren()	530
8.10.2.19	frameGetParent()	531
8.10.2.20	frameGetPose()	531
8.10.2.21	frameMove()	532
8.10.2.22	isSyncMoveOn()	533
8.10.2.23	syncMoveOff()	533
8.10.2.24	syncMoveOn()	533
8.10.2.25	syncMoveResume()	534
8.10.2.26	syncMoveSegment()	534
8.10.2.27	syncMoveSuspend()	535
8.10.2.28	syncMoveUndo()	535
8.10.2.29	waitSyncTasks()	536
8.11	SystemInfo ()	536
8.11.1	Detailed Description	537
8.11.2	Function Documentation	537
8.11.2.1	getControlSoftwareBuildDate()	537
8.11.2.2	getControlSoftwareFullVersion()	538
8.11.2.3	getControlSoftwareVersionCode()	538
8.11.2.4	getControlSoftwareVersionHash()	539
8.11.2.5	getControlSystemTime()	539
8.11.2.6	getInterfaceVersionCode()	540
8.12	Trace ()	540
8.12.1	Detailed Description	541
8.12.2	Function Documentation	541
8.12.2.1	alarm()	541
8.12.2.2	peek()	541
8.12.2.3	popup()	542
8.12.2.4	textmsg()	543
9	Directory Documentation	545
9.1	include/aubo Directory Reference	545
9.2	doc Directory Reference	546
9.3	include/aubo/error_stack Directory Reference	546
9.4	include Directory Reference	547

9.5 include/aubo/robot Directory Reference	547
10 Namespace Documentation	549
10.1 arcs Namespace Reference	549
10.2 arcs::common_interface Namespace Reference	549
10.2.1 Typedef Documentation	552
10.2.1.1 AuboApiPtr	552
10.2.1.2 AxisInterfacePtr	552
10.2.1.3 Box	552
10.2.1.4 Cylinder	552
10.2.1.5 DynamicsModel	552
10.2.1.6 ForceControlPtr	553
10.2.1.7 ForceSensorCalibResult	553
10.2.1.8 ForceSensorCalibResultWithError	553
10.2.1.9 GripperInterfacePtr	553
10.2.1.10 GripperStatusVector	553
10.2.1.11 IoControlPtr	553
10.2.1.12 MathPtr	553
10.2.1.13 MotionControlPtr	554
10.2.1.14 Payload	554
10.2.1.15 RegisterControlPtr	554
10.2.1.16 ResultWithErrno	554
10.2.1.17 ResultWithErrno1	554
10.2.1.18 ResultWithErrno2	554
10.2.1.19 ResultWithErrno3	554
10.2.1.20 RobotAlgorithmPtr	555
10.2.1.21 RobotConfigPtr	555
10.2.1.22 RobotInterfacePtr	555
10.2.1.23 RobotManagePtr	555
10.2.1.24 RobotMsgVector	555
10.2.1.25 RobotStatePtr	555
10.2.1.26 RuntimeMachinePtr	555
10.2.1.27 SerialPtr	555
10.2.1.28 SocketPtr	556
10.2.1.29 Sphere	556
10.2.1.30 SyncMovePtr	556
10.2.1.31 SystemInfoPtr	556
10.2.1.32 TaskSet	556
10.2.1.33 TracePtr	556
10.2.1.34 Vector3d	556
10.2.1.35 Vector3f	556
10.2.1.36 Vector4d	557
10.2.1.37 Vector4f	557

10.2.1.38	Vector6f	557
10.2.2	Enumeration Type Documentation	557
10.2.2.1	AuboErrorCodes	557
10.2.2.2	AxisModeType	557
10.2.2.3	EnvelopingShape	557
10.2.2.4	error_type	558
10.2.2.5	ExceptionCode	558
10.2.2.6	ForceControlState	559
10.2.2.7	JointServoModeType	559
10.2.2.8	JointStateType	559
10.2.2.9	OperationalModeType	560
10.2.2.10	PathBufferType	560
10.2.2.11	PayloadIdentifyMoveAxis	560
10.2.2.12	RefFrameType	561
10.2.2.13	RobotControlModeType	561
10.2.2.14	RobotEmergencyStopType	561
10.2.2.15	RobotModeType	562
10.2.2.16	RuntimeState	562
10.2.2.17	SafeguedStopType	562
10.2.2.18	SafetyInputAction	562
10.2.2.19	SafetyModeType	563
10.2.2.20	SafetyOutputRunState	563
10.2.2.21	StandardInputAction	563
10.2.2.22	StandardOutputRunState	563
10.2.2.23	TaskFrameType	564
10.2.2.24	TraceLevel	564
10.2.3	Function Documentation	564
10.2.3.1	operator<<() [1/7]	564
10.2.3.2	operator<<() [2/7]	564
10.2.3.3	operator<<() [3/7]	565
10.2.3.4	operator<<() [4/7]	565
10.2.3.5	operator<<() [5/7]	565
10.2.3.6	operator<<() [6/7]	565
10.2.3.7	operator<<() [7/7]	565
10.2.3.8	returnValue2Str()	565
10.3	arcs::error_stack Namespace Reference	566
10.3.1	Enumeration Type Documentation	566
10.3.1.1	ErrorCodes	566
10.3.2	Function Documentation	566
10.3.2.1	codeCompose()	566
10.3.2.2	dump()	566
10.3.2.3	errorCode2Str()	567
10.3.2.4	mod()	567

10.3.2.5 str2ErrorCode()	567
11 Class Documentation	569
11.1 arcs::common_interface::AuboApi Class Reference	569
11.1.1 Detailed Description	570
11.1.2 Constructor & Destructor Documentation	570
11.1.2.1 AuboApi()	570
11.1.2.2 ~AuboApi()	570
11.1.3 Member Data Documentation	570
11.1.3.1 d_	570
11.2 arcs::common_interface::AuboException Class Reference	571
11.2.1 Detailed Description	572
11.2.2 Constructor & Destructor Documentation	572
11.2.2.1 AuboException() [1/2]	572
11.2.2.2 AuboException() [2/2]	572
11.2.3 Member Function Documentation	573
11.2.3.1 code()	573
11.2.3.2 type()	573
11.2.3.3 what()	573
11.2.4 Member Data Documentation	573
11.2.4.1 code_	573
11.2.4.2 message_	574
11.3 arcs::common_interface::AxisInterface Class Reference	574
11.3.1 Detailed Description	575
11.3.2 Constructor & Destructor Documentation	575
11.3.2.1 AxisInterface()	575
11.3.2.2 ~AxisInterface()	575
11.3.3 Member Data Documentation	576
11.3.3.1 d_	576
11.4 arcs::common_interface::CircleParameters Struct Reference	576
11.4.1 Detailed Description	577
11.4.2 Member Data Documentation	577
11.4.2.1 a	577
11.4.2.2 blend_radius	577
11.4.2.3 direction	577
11.4.2.4 duration	577
11.4.2.5 helix	577
11.4.2.6 loop_times	578
11.4.2.7 pose_to	578
11.4.2.8 pose_via	578
11.4.2.9 spiral	578
11.4.2.10 v	578
11.5 arcs::common_interface::Enveloping Struct Reference	579

11.5.1 Detailed Description	579
11.5.2 Member Data Documentation	579
11.5.2.1 ep_args	579
11.5.2.2 shape	579
11.5.2.3 stl_path	580
11.6 arcs::common_interface::ForceControl Class Reference	580
11.6.1 Detailed Description	582
11.6.2 Constructor & Destructor Documentation	582
11.6.2.1 ForceControl()	582
11.6.2.2 ~ForceControl()	582
11.6.3 Member Data Documentation	583
11.6.3.1 d_	583
11.7 arcs::common_interface::GripperInterface Class Reference	583
11.7.1 Detailed Description	584
11.7.2 Constructor & Destructor Documentation	584
11.7.2.1 GripperInterface()	584
11.7.2.2 ~GripperInterface()	584
11.7.3 Member Function Documentation	584
11.7.3.1 gripperGetAngle()	584
11.7.3.2 gripperGetForce()	584
11.7.3.3 gripperGetMotionState()	584
11.7.3.4 gripperGetMountPose()	584
11.7.3.5 gripperGetObjectDetection()	585
11.7.3.6 gripperGetPosition()	585
11.7.3.7 gripperGetRVelocity()	585
11.7.3.8 gripperGetSoftwareVersion()	585
11.7.3.9 gripperGetTemperature()	585
11.7.3.10 gripperGetTorque()	585
11.7.3.11 gripperGetVelocity()	585
11.7.3.12 gripperGetVoltage()	586
11.7.3.13 gripperSetAngle()	586
11.7.3.14 gripperSetForce()	586
11.7.3.15 gripperSetRVelocity()	586
11.7.3.16 gripperSetTorque()	586
11.7.3.17 gripperSetVelocity()	586
11.7.4 Member Data Documentation	586
11.7.4.1 d_	586
11.8 arcs::common_interface::GripperStatus Struct Reference	587
11.8.1 Detailed Description	587
11.8.2 Member Data Documentation	588
11.8.2.1 angle	588
11.8.2.2 force	588
11.8.2.3 is_connected	588

11.8.2.4	is_enabled	588
11.8.2.5	motion_state	588
11.8.2.6	name	588
11.8.2.7	object_detection	588
11.8.2.8	position	589
11.8.2.9	r_velocity	589
11.8.2.10	status_code	589
11.8.2.11	temperature	589
11.8.2.12	torque	589
11.8.2.13	velocity	589
11.8.2.14	voltage	589
11.9	arcs::common_interface::IoControl Class Reference	590
11.9.1	Detailed Description	593
11.9.2	Constructor & Destructor Documentation	593
11.9.2.1	IoControl()	593
11.9.2.2	~IoControl()	593
11.9.3	Member Data Documentation	593
11.9.3.1	d_	593
11.10	arcs::common_interface::Math Class Reference	594
11.10.1	Detailed Description	595
11.10.2	Constructor & Destructor Documentation	595
11.10.2.1	Math()	595
11.10.2.2	~Math()	595
11.10.3	Member Data Documentation	595
11.10.3.1	d_	595
11.11	arcs::common_interface::MotionControl Class Reference	595
11.11.1	Detailed Description	601
11.11.2	Constructor & Destructor Documentation	601
11.11.2.1	MotionControl()	601
11.11.2.2	~MotionControl()	601
11.11.3	Member Function Documentation	601
11.11.3.1	getConveyorTrackNextItem()	601
11.11.3.2	servoCartesianWithAxisGroup()	602
11.11.3.3	servoJointWithAxisGroup()	602
11.11.4	Member Data Documentation	602
11.11.4.1	d_	602
11.12	arcs::common_interface::RegisterControl Class Reference	602
11.12.1	Detailed Description	605
11.12.2	Constructor & Destructor Documentation	605
11.12.2.1	RegisterControl()	605
11.12.2.2	~RegisterControl()	606
11.12.3	Member Function Documentation	606
11.12.3.1	modbusSetOutputSignalWithTimeout()	606

11.12.4 Member Data Documentation	607
11.12.4.1 d_	607
11.13 arcs::common_interface::RobotAlgorithm Class Reference	607
11.13.1 Detailed Description	609
11.13.2 Constructor & Destructor Documentation	609
11.13.2.1 RobotAlgorithm()	609
11.13.2.2 ~RobotAlgorithm()	609
11.13.3 Member Function Documentation	609
11.13.3.1 pathMovej()	609
11.13.4 Member Data Documentation	610
11.13.4.1 d_	610
11.14 arcs::common_interface::RobotConfig Class Reference	610
11.14.1 Detailed Description	614
11.14.2 Constructor & Destructor Documentation	614
11.14.2.1 RobotConfig()	614
11.14.2.2 ~RobotConfig()	614
11.14.3 Member Function Documentation	614
11.14.3.1 disableAxisGroup()	614
11.14.3.2 enableAxisGroup()	614
11.14.3.3 getRobotBaseParent()	615
11.14.4 Member Data Documentation	615
11.14.4.1 d_	615
11.15 arcs::common_interface::RobotInterface Class Reference	615
11.15.1 Detailed Description	616
11.15.2 Constructor & Destructor Documentation	616
11.15.2.1 RobotInterface()	616
11.15.2.2 ~RobotInterface()	616
11.15.3 Member Data Documentation	616
11.15.3.1 d_	616
11.16 arcs::common_interface::RobotManage Class Reference	616
11.16.1 Detailed Description	618
11.16.2 Constructor & Destructor Documentation	618
11.16.2.1 RobotManage()	618
11.16.2.2 ~RobotManage()	618
11.16.3 Member Data Documentation	618
11.16.3.1 d_	618
11.17 arcs::common_interface::RobotMsg Struct Reference	619
11.17.1 Detailed Description	619
11.17.2 Member Data Documentation	620
11.17.2.1 args	620
11.17.2.2 code	620
11.17.2.3 level	620
11.17.2.4 source	620

11.17.2.5 timestamp	620
11.18 arcs::common_interface::RobotSafetyParameterRange Struct Reference	621
11.18.1 Detailed Description	623
11.18.2 Constructor & Destructor Documentation	623
11.18.2.1 RobotSafetyParameterRange()	623
11.18.3 Member Data Documentation	623
11.18.3.1 allow_manual_high_speed	623
11.18.3.2 crc32	623
11.18.3.3 [struct]	623
11.18.3.4 elbow_force	624
11.18.3.5 elbow_speed	624
11.18.3.6 joint_torque	624
11.18.3.7 momentum	624
11.18.3.8 orig	624
11.18.3.9 [struct]	624
11.18.3.10 plane	624
11.18.3.11 planes	625
11.18.3.12 power	625
11.18.3.13 qdmax	625
11.18.3.14 qmax	625
11.18.3.15 qmin	625
11.18.3.16 reduced_entry_distance	625
11.18.3.17 reduced_entry_time	626
11.18.3.18 restrict_elbow	626
11.18.3.19 safety_home	626
11.18.3.20 safety_input_auto_safeguard_reset	626
11.18.3.21 safety_input_auto_safeguard_stop	626
11.18.3.22 safety_input_emergency_stop	626
11.18.3.23 safety_input_handguide	627
11.18.3.24 safety_input_operational_mode	627
11.18.3.25 safety_input_reduced_mode	627
11.18.3.26 safety_input_safeguard_reset	627
11.18.3.27 safety_input_safeguard_stop	627
11.18.3.28 safety_input_three_position_switch	627
11.18.3.29 safety_output_emergency_stop	628
11.18.3.30 safety_output_not_emergency_stop	628
11.18.3.31 safety_output_not_reduced_mode	628
11.18.3.32 safety_output_reduced_mode	628
11.18.3.33 safety_output_robot_moving	628
11.18.3.34 safety_output_robot_not_stopping	628
11.18.3.35 safety_output_robot_steady	629
11.18.3.36 safety_output_safe_home	629
11.18.3.37 safety_output_safetyguard_stop	629

11.18.3.38	size	629
11.18.3.39	stop_distance	629
11.18.3.40	stop_time	629
11.18.3.41	tcp_force	630
11.18.3.42	tcp_speed	630
11.18.3.43	tool_azimuth	630
11.18.3.44	tool_deviation	630
11.18.3.45	tool_inclination	630
11.18.3.46	tool_orientation	630
11.18.3.47	tools	631
11.18.3.48	tp_3pe_for_handguide	631
11.18.3.49	[struct]	631
11.19	arcs::common_interface::RobotState Class Reference	631
11.19.1	Detailed Description	634
11.19.2	Constructor & Destructor Documentation	634
11.19.2.1	RobotState()	634
11.19.2.2	~RobotState()	634
11.19.3	Member Function Documentation	635
11.19.3.1	getBaseForceSensor()	635
11.19.4	Member Data Documentation	635
11.19.4.1	d_	635
11.20	arcs::common_interface::RtdeRecipe Struct Reference	636
11.20.1	Detailed Description	636
11.20.2	Member Data Documentation	637
11.20.2.1	chanel	637
11.20.2.2	frequency	637
11.20.2.3	segments	637
11.20.2.4	to_server	637
11.20.2.5	trigger	637
11.21	arcs::common_interface::RuntimeMachine Class Reference	637
11.21.1	Detailed Description	639
11.21.2	Constructor & Destructor Documentation	640
11.21.2.1	RuntimeMachine()	640
11.21.2.2	~RuntimeMachine()	640
11.21.3	Member Function Documentation	640
11.21.3.1	getExecutionStatus1()	640
11.21.3.2	getRuntimeState()	640
11.21.4	Member Data Documentation	640
11.21.4.1	d_	640
11.22	arcs::common_interface::Serial Class Reference	640
11.22.1	Detailed Description	641
11.22.2	Constructor & Destructor Documentation	641
11.22.2.1	Serial()	641

11.22.2.2 <code>~Serial()</code>	641
11.22.3 Member Data Documentation	642
11.22.3.1 <code>d__</code>	642
11.23 <code>arcs::common_interface::Socket</code> Class Reference	642
11.23.1 Detailed Description	643
11.23.2 Constructor & Destructor Documentation	643
11.23.2.1 <code>Socket()</code>	643
11.23.2.2 <code>~Socket()</code>	643
11.23.3 Member Data Documentation	643
11.23.3.1 <code>d__</code>	643
11.24 <code>arcs::common_interface::SpiralParameters</code> Struct Reference	643
11.24.1 Detailed Description	644
11.24.2 Member Data Documentation	644
11.24.2.1 <code>angle</code>	644
11.24.2.2 <code>frame</code>	644
11.24.2.3 <code>helix</code>	644
11.24.2.4 <code>plane</code>	644
11.24.2.5 <code>spiral</code>	645
11.25 <code>arcs::common_interface::SyncMove</code> Class Reference	645
11.25.1 Detailed Description	646
11.25.2 Constructor & Destructor Documentation	647
11.25.2.1 <code>SyncMove()</code>	647
11.25.2.2 <code>~SyncMove()</code>	647
11.25.3 Member Data Documentation	647
11.25.3.1 <code>d__</code>	647
11.26 <code>arcs::common_interface::SystemInfo</code> Class Reference	647
11.26.1 Detailed Description	648
11.26.2 Constructor & Destructor Documentation	648
11.26.2.1 <code>SystemInfo()</code>	648
11.26.2.2 <code>~SystemInfo()</code>	648
11.26.3 Member Data Documentation	648
11.26.3.1 <code>d__</code>	648
11.27 <code>arcs::common_interface::Trace</code> Class Reference	648
11.27.1 Detailed Description	649
11.27.2 Constructor & Destructor Documentation	649
11.27.2.1 <code>Trace()</code>	649
11.27.2.2 <code>~Trace()</code>	649
11.27.3 Member Function Documentation	649
11.27.3.1 <code>notify()</code>	649
11.27.4 Member Data Documentation	649
11.27.4.1 <code>d__</code>	649
11.28 <code>arcs::common_interface::TrajConfig</code> Struct Reference	650
11.28.1 Detailed Description	651

11.28.2 Member Data Documentation	651
11.28.2.1 envelopings	651
11.28.2.2 init_joint	651
11.28.2.3 lower_joint_bound	651
11.28.2.4 max_acceleration	651
11.28.2.5 max_velocity	651
11.28.2.6 move_axis	651
11.28.2.7 upper_joint_bound	652
11.29 arcs::common_interface::VibrationRecalibrationParameter Struct Reference	652
11.29.1 Detailed Description	653
11.29.2 Member Data Documentation	653
11.29.2.1 cog	653
11.29.2.2 inertia	653
11.29.2.3 mass	653
11.29.2.4 points	653
11.29.2.5 stiff_param	653
11.29.2.6 stiff_section	653
11.30 arcs::common_interface::WObjectData Struct Reference	654
11.30.1 Detailed Description	654
11.30.2 Member Data Documentation	654
11.30.2.1 attach_frame	654
11.30.2.2 obj_coord	655
11.30.2.3 remote_tool	655
11.30.2.4 user_coord	655
12 File Documentation	657
12.1 doc/SDK_overview.md File Reference	657
12.2 include/aubo/aubo_api.h File Reference	657
12.2.1 Detailed Description	658
12.3 aubo_api.h	658
12.4 include/aubo/axis_interface.h File Reference	662
12.5 axis_interface.h	663
12.6 include/aubo/error_stack/error_stack.h File Reference	668
12.6.1 Detailed Description	670
12.6.2 Macro Definition Documentation	670
12.6.2.1 _D [1/6]	670
12.6.2.2 _D [2/6]	670
12.6.2.3 _D [3/6]	670
12.6.2.4 _D [4/6]	671
12.6.2.5 _D [5/6]	671
12.6.2.6 _D [6/6]	671
12.6.2.7 _PH1_	671
12.6.2.8 _PH2_	671

12.6.2.9	_PH3	672
12.6.2.10	_PH4	672
12.6.2.11	ARCS_ERROR_CODES	672
12.6.2.12	AUBO_SDK_HAL_ERROR_H	672
12.6.2.13	AUBO_SDK_RTM_ERROR_H	672
12.6.2.14	AUBO_SDK_SYSTEM_ERROR_H	672
12.6.2.15	HAL_ERRORS	673
12.6.2.16	HARDWARE_INTERFACE_ERRORS	673
12.6.2.17	JOINT_ERRORS	673
12.6.2.18	PEDSTRAL_ERRORS	673
12.6.2.19	RTM_ERRORS	673
12.6.2.20	SAFETY_INTERFACE_BOARD_ERRORS	673
12.6.2.21	SYSTEM_ERRORS	674
12.6.2.22	TOOL_ERRORS	674
12.7	error_stack.h	674
12.8	include/aubo/error_stack/hal_error.h File Reference	675
12.8.1	Detailed Description	676
12.8.2	Macro Definition Documentation	676
12.8.2.1	HAL_ERRORS	676
12.8.2.2	HARDWARE_INTERFACE_ERRORS	676
12.8.2.3	JOINT_ERRORS	677
12.8.2.4	PEDSTRAL_ERRORS	677
12.8.2.5	SAFETY_INTERFACE_BOARD_ERRORS	677
12.8.2.6	TOOL_ERRORS	678
12.9	hal_error.h	678
12.10	include/aubo/error_stack/rtm_error.h File Reference	681
12.10.1	Detailed Description	681
12.10.2	Macro Definition Documentation	681
12.10.2.1	RTM_ERRORS	681
12.11	rtm_error.h	682
12.12	include/aubo/error_stack/system_error.h File Reference	685
12.12.1	Detailed Description	686
12.12.2	Macro Definition Documentation	686
12.12.2.1	SYSTEM_ERRORS	686
12.13	system_error.h	686
12.14	include/aubo/gripper_interface.h File Reference	687
12.14.1	Detailed Description	688
12.15	gripper_interface.h	688
12.16	include/aubo/math.h File Reference	692
12.16.1	Detailed Description	693
12.17	math.h	693
12.18	include/aubo/register_control.h File Reference	706
12.18.1	Detailed Description	707

12.18.2 Enumeration Type Documentation	707
12.18.2.1 ModbusErrorNum	707
12.19 register_control.h	708
12.20 include/aubo/robot/force_control.h File Reference	747
12.20.1 Detailed Description	748
12.21 force_control.h	748
12.22 include/aubo/robot/io_control.h File Reference	779
12.22.1 Detailed Description	780
12.23 io_control.h	780
12.24 include/aubo/robot/motion_control.h File Reference	837
12.24.1 Detailed Description	839
12.25 motion_control.h	839
12.26 include/aubo/robot/robot_algorithm.h File Reference	915
12.26.1 Detailed Description	916
12.27 robot_algorithm.h	917
12.28 include/aubo/robot/robot_config.h File Reference	938
12.28.1 Detailed Description	939
12.29 robot_config.h	939
12.30 include/aubo/robot/robot_manage.h File Reference	983
12.30.1 Detailed Description	984
12.31 robot_manage.h	985
12.32 include/aubo/robot/robot_state.h File Reference	1007
12.32.1 Detailed Description	1008
12.33 robot_state.h	1008
12.34 include/aubo/robot_interface.h File Reference	1044
12.34.1 Detailed Description	1045
12.35 robot_interface.h	1045
12.36 include/aubo/runtime_machine.h File Reference	1049
12.36.1 Detailed Description	1050
12.37 runtime_machine.h	1051
12.38 include/aubo/serial.h File Reference	1069
12.38.1 Detailed Description	1070
12.39 serial.h	1071
12.40 include/aubo/socket.h File Reference	1075
12.40.1 Detailed Description	1076
12.41 socket.h	1077
12.42 include/aubo/sync_move.h File Reference	1084
12.42.1 Detailed Description	1085
12.43 sync_move.h	1086
12.44 include/aubo/system_info.h File Reference	1097
12.44.1 Detailed Description	1098
12.45 system_info.h	1099
12.46 include/aubo/trace.h File Reference	1103

12.46.1 Detailed Description	1104
12.47 trace.h	1104
12.48 include/aubo/type_def.h File Reference	1107
12.48.1 Detailed Description	1111
12.48.2 Macro Definition Documentation	1111
12.48.2.1 CARTESIAN_DOOF	1111
12.48.2.2 DECL_TO_STRING_FUNC	1111
12.48.2.3 ENUM_AuboErrorCodes_DECLARES	1111
12.48.2.4 ENUM_AxisModeType_DECLARES	1111
12.48.2.5 ENUM_EnvelopingShape_DECLARES	1112
12.48.2.6 ENUM_ITEM [1/5]	1112
12.48.2.7 ENUM_ITEM [2/5]	1112
12.48.2.8 ENUM_ITEM [3/5]	1112
12.48.2.9 ENUM_ITEM [4/5]	1113
12.48.2.10 ENUM_ITEM [5/5]	1113
12.48.2.11 ENUM_JointServoModeType_DECLARES	1113
12.48.2.12 ENUM_JointStateType_DECLARES	1113
12.48.2.13 ENUM_OperationalModeType_DECLARES	1114
12.48.2.14 ENUM_PayloadIdentifyMoveAxis_DECLARES	1114
12.48.2.15 ENUM_RobotControlModeType_DECLARES	1114
12.48.2.16 ENUM_RobotEmergencyStopType_DECLARES	1114
12.48.2.17 ENUM_RobotModeType_DECLARES	1115
12.48.2.18 ENUM_RuntimeState_DECLARES	1115
12.48.2.19 ENUM_SafeguedStopType_DECLARES	1115
12.48.2.20 ENUM_SafetyInputAction_DECLARES	1116
12.48.2.21 ENUM_SafetyModeType_DECLARES	1116
12.48.2.22 ENUM_SafetyOutputRunState_DECLARES	1116
12.48.2.23 ENUM_StandardInputAction_DECLARES	1117
12.48.2.24 ENUM_StandardOutputRunState_DECLARES	1117
12.48.2.25 ENUM_TaskFrameType_DECLARES	1117
12.48.2.26 ENUM_TraceLevel_DECLARES	1118
12.48.2.27 SAFETY_CUBIC_NUM	1118
12.48.2.28 SAFETY_PARAM_SELECT_NUM	1118
12.48.2.29 SAFETY_PLANES_NUM	1118
12.48.2.30 TOOL_CONFIGURATION_NUM	1118
12.49 type_def.h	1119
Index	1131

Chapter 1

Aubo SDK

API [Aubo SDK API .pdf](#)

Aubo SDK SDK [AuboApi \(\)](#)

-
- SDK
-
-
-
-
-

```
// 1.    API
auto rpc_cli = std::make_shared<RpcClient>();

// 2.
auto robot_names = rpc_cli->getRobotNames();
auto robot_name = robot_names.front();

// 3.
auto robot = rpc_cli->getRobotInterface(robot_name);

// 4.
auto motion = robot->getMotionControl();
auto state = robot->getRobotState();
```

SDK

AuboApi ()

```

SystemInfo ( )
RuntimeMachine ( )
RegisterControl ( )
Math ( )
SyncMove ( )
Trace ( )

RobotInterface ( )
  RobotConfig ( )
  MotionControl ( )
  ForceControl ( )
  IoControl (IO )
  RobotAlgorithm ( )
  RobotManage ( )
  RobotState ( )
AxisInterface ( )
GripperInterface ( )

Socket ( )
Serial ( )

```

	MotionControl	robot/motion_control.h	
	RobotConfig	robot/robot_config.h	TCP
	RobotState	robot/robot_state.h	
	RobotManage	robot/robot_manage.h	
IO	IoControl	robot/io_control.h	IO IO IO
	ForceControl	robot/force_control.h	/
	RobotAlgorithm	robot/robot_algorithm.h	/
	AxisInterface	axis_interface.h	
	GripperInterface	gripper_interface.h	
	SystemInfo	system_info.h	
	RuntimeMachine	runtime_machine.h	
	RegisterControl	register_control.h	
	Math	math.h	
	Socket	socket.h	TCP
	Serial	serial.h	
	SyncMove	sync_move.h	
	Trace	trace.h	

```
auto rpc_cli = std::make_shared<RpcClient>();
```

```

auto robot = rpc_cli->getRobotInterface("rob1");
auto state = robot->getRobotState();

//
auto joint_angles = state->getJointState();


auto motion = robot->getMotionControl();

//
motion->setEqradius(0.8);

//
std::vector<double> target_pose = {0, -1.57, 1.57, 0, 0, 0};
motion->moveJoint(target_pose, 0.5, 0.5);

```

IO

```

auto io = robot->getIoControl();

//
io->setStandardDigitalOutput(0, true);

//
bool input_state = io->getStandardDigitalInput(0);

```

```

namespace arcs {
namespace common_interface {
//
}
}

```

0 AuboException

- C++
- Python
- Lua
- JSON-RPC

- [type_def.h](#) -
- [global_config.h](#) -
- [error_stack/](#) -

Chapter 2

Topic Index

2.1 Topics

Here is a list of all topics with brief descriptions:

AuboApi ()	17
AxisInterface ()	22
GripperInterface ()	31
Math ()	34
RegisterControl ()	51
RobotInterface()	103
ForceControl()	107
IoControl (IO)	144
MotionControl ()	219
RobotAlgorithm ()	319
RobotConfig ()	346
RobotManage ()	404
RobotState ()	432
RuntimeMachine ()	480
Serial ()	503
Socket (socket)	510
SyncMove ()	521
SystemInfo ()	536
Trace ()	540

Chapter 3

Directory Hierarchy

3.1 Directories

aubo	545
error_stack	546
error_stack.h	668
hal_error.h	675
rtm_error.h	681
system_error.h	685
robot	547
force_control.h	747
io_control.h	779
motion_control.h	837
robot_algorithm.h	915
robot_config.h	938
robot_manage.h	983
robot_state.h	1007
aubo_api.h	657
axis_interface.h	662
gripper_interface.h	687
math.h	692
register_control.h	706
robot_interface.h	1044
runtime_machine.h	1049
serial.h	1069
socket.h	1075
sync_move.h	1084
system_info.h	1097
trace.h	1103
type_def.h	1107
doc	546
error_stack	546
error_stack.h	668
hal_error.h	675
rtm_error.h	681
system_error.h	685
include	547
aubo	545
error_stack	546
error_stack.h	668

hal_error.h	675
rtm_error.h	681
system_error.h	685
robot	547
force_control.h	747
io_control.h	779
motion_control.h	837
robot_algorithm.h	915
robot_config.h	938
robot_manage.h	983
robot_state.h	1007
aubo_api.h	657
axis_interface.h	662
gripper_interface.h	687
math.h	692
register_control.h	706
robot_interface.h	1044
runtime_machine.h	1049
serial.h	1069
socket.h	1075
sync_move.h	1084
system_info.h	1097
trace.h	1103
type_def.h	1107
robot	547
force_control.h	747
io_control.h	779
motion_control.h	837
robot_algorithm.h	915
robot_config.h	938
robot_manage.h	983
robot_state.h	1007

Chapter 4

Namespace Index

4.1 Namespace List

Here is a list of all namespaces with brief descriptions:

arcs	549
arcs::common_interface	549
arcs::error_stack	566

Chapter 5

Hierarchical Index

5.1 Class Hierarchy

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

arcs::common_interface::AuboApi	569
arcs::common_interface::AxisInterface	574
arcs::common_interface::CircleParameters	576
arcs::common_interface::Enveloping	579
std::exception	
arcs::common_interface::AuboException	571
arcs::common_interface::ForceControl	580
arcs::common_interface::GripperInterface	583
arcs::common_interface::GripperStatus	587
arcs::common_interface::IoControl	590
arcs::common_interface::Math	594
arcs::common_interface::MotionControl	595
arcs::common_interface::RegisterControl	602
arcs::common_interface::RobotAlgorithm	607
arcs::common_interface::RobotConfig	610
arcs::common_interface::RobotInterface	615
arcs::common_interface::RobotManage	616
arcs::common_interface::RobotMsg	619
arcs::common_interface::RobotSafetyParameterRange	621
arcs::common_interface::RobotState	631
arcs::common_interface::RtdeRecipe	636
arcs::common_interface::RuntimeMachine	637
arcs::common_interface::Serial	640
arcs::common_interface::Socket	642
arcs::common_interface::SpiralParameters	643
arcs::common_interface::SyncMove	645
arcs::common_interface::SystemInfo	647
arcs::common_interface::Trace	648
arcs::common_interface::TrajConfig	650
arcs::common_interface::VibrationRecalibrationParameter	652
arcs::common_interface::WObjectData	654

Chapter 6

Class Index

6.1 Class List

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

arcs::common_interface::AuboApi	569
arcs::common_interface::AuboException	
Custom exception class AuboException	571
arcs::common_interface::AxisInterface	574
arcs::common_interface::CircleParameters	
Circular motion parameters definition	576
arcs::common_interface::Enveloping	579
arcs::common_interface::ForceControl	580
arcs::common_interface::GripperInterface	583
arcs::common_interface::GripperStatus	587
arcs::common_interface::IoControl	590
arcs::common_interface::Math	594
arcs::common_interface::MotionControl	595
arcs::common_interface::RegisterControl	602
arcs::common_interface::RobotAlgorithm	607
arcs::common_interface::RobotConfig	610
arcs::common_interface::RobotInterface	615
arcs::common_interface::RobotManage	616
arcs::common_interface::RobotMsg	619
arcs::common_interface::RobotSafetyParameterRange	621
arcs::common_interface::RobotState	631
arcs::common_interface::RtdeRecipe	
RTDE menu	636
arcs::common_interface::RuntimeMachine	637
arcs::common_interface::Serial	640
arcs::common_interface::Socket	642
arcs::common_interface::SpiralParameters	643
arcs::common_interface::SyncMove	645
arcs::common_interface::SystemInfo	647
arcs::common_interface::Trace	648
arcs::common_interface::TrajConfig	
Trajectory configuration for payload identification	650
arcs::common_interface::VibrationRecalibrationParameter	652
arcs::common_interface::WObjectData	654

Chapter 7

File Index

7.1 File List

Here is a list of all files with brief descriptions:

include/aubo/ aubo_api.h	
API for controlling the robot and external axis	657
include/aubo/ axis_interface.h	662
include/aubo/ gripper_interface.h	
	687
include/aubo/ math.h	
Mathematic operation interface, such as euler to quaternion conversion, addition/ subtraction of poses	692
include/aubo/ register_control.h	
	706
include/aubo/ robot_interface.h	
API	1044
include/aubo/ runtime_machine.h	
Script interpreter runtime interface, allows pausing the script interpreter and setting/ removing breakpoints	1049
include/aubo/ serial.h	
	1069
include/aubo/ socket.h	
Socket	1075
include/aubo/ sync_move.h	
	1084
include/aubo/ system_info.h	
	1097
include/aubo/ trace.h	
Interface for injecting logs into the controller's logging system	1103
include/aubo/ type_def.h	
Enum type definitions	1107
include/aubo/error_stack/ error_stack.h	
	668
include/aubo/error_stack/ hal_error.h	
	675
include/aubo/error_stack/ rtm_error.h	
	681
include/aubo/error_stack/ system_error.h	
	685

include/aubo/robot/ force_control.h	
747	
include/aubo/robot/ io_control.h	
IO	779
include/aubo/robot/ motion_control.h	
Motion control interface	837
include/aubo/robot/ robot_algorithm.h	
.	915
include/aubo/robot/ robot_config.h	
DH	938
include/aubo/robot/ robot_manage.h	
.	983
include/aubo/robot/ robot_state.h	
/	1007

Chapter 8

Topic Documentation

8.1 AuboApi ()

AuboApi.

Functions

- [MathPtr arcs::common_interface::AuboApi::getMath \(\)](#)
[Math \(\)](#)
Get pure mathematic related API
- [SystemInfoPtr arcs::common_interface::AuboApi::getSystemInfo \(\)](#)
[SystemInfo \(\)](#)
Get system info
- [RuntimeMachinePtr arcs::common_interface::AuboApi::getRuntimeMachine \(\)](#)
[RuntimeMachine \(\)](#)
Get runtime api
- [RegisterControlPtr arcs::common_interface::AuboApi::getRegisterControl \(\)](#)
[RegisterControl \(\)](#)
External registers api
- [std::vector< std::string > arcs::common_interface::AuboApi::getRobotNames \(\)](#)
Get robot list
- [RobotInterfacePtr arcs::common_interface::AuboApi::getRobotInterface \(const std::string &name\)](#)
[RobotInterface\(\)](#)
Get [RobotInterfacePtr](#) based on name
- [std::vector< std::string > arcs::common_interface::AuboApi::getAxisNames \(\)](#)
Get external axis list.
- [AxisInterfacePtr arcs::common_interface::AuboApi::getAxisInterface \(const std::string &name\)](#)
[AxisInterface \(\)](#)
Get external axis interface
- [SocketPtr arcs::common_interface::AuboApi::getSocket \(\)](#)
[Socket \(socket \)](#)
Get socket
- [SerialPtr arcs::common_interface::AuboApi::getSerial \(\)](#)
[Serial \(\)](#)
- [SyncMovePtr arcs::common_interface::AuboApi::getSyncMove \(const std::string &name\)](#)
[SyncMove \(\)](#) Get synchronous move interface
- [TracePtr arcs::common_interface::AuboApi::getTrace \(const std::string &name\)](#)
[Trace \(\)](#) Get alert interface
- [GripperInterfacePtr arcs::common_interface::AuboApi::getGripperInterface \(\)](#)
[GripperInterface \(\)](#) Get gripper interface

8.1.1 Detailed Description

[AuboApi](#).

8.1.2 Function Documentation

8.1.2.1 `getAxisInterface()`

[AxisInterfacePtr](#) arcs::common_interface::AuboApi::getAxisInterface (
const std::string & name)

[AxisInterface](#) ()

Get external axis interface

Parameters

name	
------	--

Returns

8.1.2.2 `getAxisNames()`

std::vector< std::string > arcs::common_interface::AuboApi::getAxisNames ()

Get external axis list.

Returns

8.1.2.3 `getGripperInterface()`

[GripperInterfacePtr](#) arcs::common_interface::AuboApi::getGripperInterface ()

[GripperInterface](#) () Get gripper interface

Returns

Shared pointer of gripper object

8.1.2.4 getMath()

[MathPtr](#) arcs::common_interface::AuboApi::getMath ()

[Math](#) ()

Get pure mathematic related API

Returns

Shared pointer to a [Math](#) object

Python function prototype

```
getMath(self: pyaubo_sdk.AuboApi) -> pyaubo_sdk.Math
```

C++ example

```
auto rpc_cli = std::make_shared<RpcClient>();
MathPtr ptr = rpc_cli->getMath();
```

8.1.2.5 getRegisterControl()

[RegisterControlPtr](#) arcs::common_interface::AuboApi::getRegisterControl ()

[RegisterControl](#) ()

External registers api

Returns

Shared pointer to [RegisterControl](#) object

Python function prototype

```
getRegisterControl(self: pyaubo_sdk.AuboApi) -> pyaubo_sdk.RegisterControl
```

C++ example

```
auto rpc_cli = std::make_shared<RpcClient>();
RegisterControlPtr ptr = rpc_cli->getRegisterControl();
```

8.1.2.6 getRobotInterface()

[RobotInterfacePtr](#) arcs::common_interface::AuboApi::getRobotInterface (
const std::string & name)

[RobotInterface](#)()

Parameters

name	Robot name
------	------------

Returns

Shared pointer to a [RobotInterface](#) object

Python function prototype

```
getRobotInterface(self: pyaubo_sdk.AuboApi, arg0: str) -> pyaubo_sdk.RobotInterface
```

C++ example

```
auto rpc_cli = std::make_shared<RpcClient>();
auto robot_name = rpc_cli->getRobotNames().front();
RobotInterfacePtr ptr = rpc_cli->getRobotInterface(robot_name);
```

8.1.2.7 getRobotNames()

```
std::vector< std::string > arcs::common_interface::AuboApi::getRobotNames ()
```

Get robot list

Returns

robot list

Python function prototype

```
getRobotNames(self: pyaubo_sdk.AuboApi) -> List[str]
```

C++ example

```
auto rpc_cli = std::make_shared<RpcClient>();
auto robot_name = rpc_cli->getRobotNames().front();
```

JSON-RPC request example

```
{"jsonrpc": "2.0", "method": "getRobotNames", "params": [], "id": 1}
```

JSON-RPC response example

```
{"id": 1, "jsonrpc": "2.0", "result": ["rob1"]}
```

8.1.2.8 getRuntimeMachine()

[RuntimeMachinePtr](#) arcs::common_interface::AuboApi::getRuntimeMachine ()

[RuntimeMachine](#) ()

Get runtime api

Returns

Shared pointer to [RuntimeMachine](#) object Python function prototype `getRuntimeMachine(self:↵
pyaubo_sdk.AuboApi) -> pyaubo_sdk.RuntimeMachine`

C++ example

```
auto rpc_cli = std::make_shared<RpcClient>();
RuntimeMachinePtr ptr = rpc_cli->getRuntimeMachine();
```

8.1.2.9 getSerial()

[SerialPtr](#) arcs::common_interface::AuboApi::getSerial ()

[Serial](#) ()

Returns

Shared pointer to [Serial](#) object

Python function prototype

`getSerial(self: pyaubo_sdk.AuboApi) -> arcs::common_interface::Serial`

C++ example

```
auto rpc_cli = std::make_shared<RpcClient>();
SerialPtr ptr = rpc_cli->getSerial();
```

8.1.2.10 getSocket()

[SocketPtr](#) arcs::common_interface::AuboApi::getSocket ()

[Socket](#) (socket)

Get socket

Returns

Shared pointer to a socket object

Python function prototype

`getSocket(self: pyaubo_sdk.AuboApi) -> arcs::common_interface::Socket`

C++ example

```
auto rpc_cli = std::make_shared<RpcClient>();
SocketPtr ptr = rpc_cli->getSocket();
```

8.1.2.11 getSyncMove()

```
SyncMovePtr arcs::common_interface::AuboApi::getSyncMove (
    const std::string & name)
```

[SyncMove](#) () Get synchronous move interface

Returns

Shared pointer to [SyncMove](#) object

8.1.2.12 getSystemInfo()

```
SystemInfoPtr arcs::common_interface::AuboApi::getSystemInfo ()
```

[SystemInfo](#) ()

Get system info

Returns

Shared pointer to [SystemInfo](#) object

Python function prototype

```
getSystemInfo(self: pyaubo_sdk.AuboApi) -> pyaubo_sdk.SystemInfo
```

C++ example

```
auto rpc_cli = std::make_shared<RpcClient>();
SystemInfoPtr ptr = rpc_cli->getSystemInfo();
```

8.1.2.13 getTrace()

```
TracePtr arcs::common_interface::AuboApi::getTrace (
    const std::string & name)
```

[Trace](#) () Get alert interface

Returns

Shared pointer of trace object

8.2 AxisInterface ()

External axis API interface.

Functions

- `int arcs::common_interface::AxisInterface::poweronExtAxis ()`
Power on.
- `int arcs::common_interface::AxisInterface::poweroffExtAxis ()`
Power off.
- `int arcs::common_interface::AxisInterface::enableExtAxis ()`
Enable.
- `int arcs::common_interface::AxisInterface::setExtAxisMountingPose (const std::vector< double > &pose)`
Set mounting pose of external axis (wrt world frame).
- `int arcs::common_interface::AxisInterface::moveExtJoint (double pos, double v, double a, double duration)`
move to pos, rotation or linear
- `int arcs::common_interface::AxisInterface::speedExtJoint (double v, double a, double duration)`
Set target speed, acceleration and duration
- `int arcs::common_interface::AxisInterface::stopExtJoint (double a)`
stop ext joint
- `int arcs::common_interface::AxisInterface::getExtAxisType ()`
Get external axis type: 0 for rotation, 1 for linear.
- `AxisModeType arcs::common_interface::AxisInterface::getAxisModeType ()`
Get external axis status
- `std::vector< double > arcs::common_interface::AxisInterface::getExtAxisMountingPose ()`
Get external axis mounting pose
- `std::vector< double > arcs::common_interface::AxisInterface::getExtAxisPose ()`
Get pose wrt mounting coordinate system, axis can be positioner or linear rail
- `double arcs::common_interface::AxisInterface::getExtAxisPosition ()`
Get external axis position
- `double arcs::common_interface::AxisInterface::getExtAxisVelocity ()`
Get external axis speed
- `double arcs::common_interface::AxisInterface::getExtAxisAcceleration ()`
Get external axis acceleration
- `double arcs::common_interface::AxisInterface::getExtAxisCurrent ()`
Get external axis current
- `double arcs::common_interface::AxisInterface::getExtAxisTemperature ()`
Get external axis temperature
- `double arcs::common_interface::AxisInterface::getExtAxisBusVoltage ()`
Get external axis voltage
- `double arcs::common_interface::AxisInterface::getExtAxisBusCurrent ()`
Get external axis current
- `double arcs::common_interface::AxisInterface::getExtAxisMaxPosition ()`
Get external axis max position
- `double arcs::common_interface::AxisInterface::getExtMinPosition ()`
Get external axis min position
- `double arcs::common_interface::AxisInterface::getExtAxisMaxVelocity ()`
Get external axis max speed
- `double arcs::common_interface::AxisInterface::getExtAxisMaxAcceleration ()`
Get external axis max acceleration
- `int arcs::common_interface::AxisInterface::followAnotherAxis (const std::string &target_name, double phase, double err)`
Follow motion of another external axis not to be used during motion

- `int arcs::common_interface::AxisInterface::stopFollowAnotherAxis ()`
stopFollowAnotherAxis(not to be used during motion)
- `int arcs::common_interface::AxisInterface::getErrorCode ()`
Get external axis error code
- `int arcs::common_interface::AxisInterface::clearAxisError ()`
Reset axis error

8.2.1 Detailed Description

External axis API interface.

8.2.2 Function Documentation

8.2.2.1 clearAxisError()

```
int arcs::common_interface::AxisInterface::clearAxisError ()
```

Reset axis error

Returns

8.2.2.2 enableExtAxis()

```
int arcs::common_interface::AxisInterface::enableExtAxis ()
```

Enable.

Returns

8.2.2.3 followAnotherAxis()

```
int arcs::common_interface::AxisInterface::followAnotherAxis (
    const std::string & target_name,
    double phase,
    double err)
```

Follow motion of another external axis not to be used during motion

Parameters

target_name	name of target axis
phase	phase difference
err	max error when following motion

Returns

8.2.2.4 getAxisModeType()

[AxisModeType](#) arcs::common_interface::AxisInterface::getAxisModeType ()

Get external axis status

Returns

Current exteral axis status

8.2.2.5 getErrorCode()

int arcs::common_interface::AxisInterface::getErrorCode ()

Get external axis error code

Returns

External axis error code

8.2.2.6 getExtAxisAcceleration()

double arcs::common_interface::AxisInterface::getExtAxisAcceleration ()

Get external axis acceleration

Returns

External axis acceleration

8.2.2.7 getExtAxisBusCurrent()

double arcs::common_interface::AxisInterface::getExtAxisBusCurrent ()

Get external axis current

Returns

external axis current

8.2.2.8 getExtAxisBusVoltage()

double arcs::common_interface::AxisInterface::getExtAxisBusVoltage ()

Get external axis voltage

Returns

External axis voltage

8.2.2.9 getExtAxisCurrent()

double arcs::common_interface::AxisInterface::getExtAxisCurrent ()

Get external axis current

Returns

External axis current

8.2.2.10 getExtAxisMaxAcceleration()

double arcs::common_interface::AxisInterface::getExtAxisMaxAcceleration ()

Get external axis max acceleration

Returns

External axis max acceleration

8.2.2.11 getExtAxisMaxPosition()

double arcs::common_interface::AxisInterface::getExtAxisMaxPosition ()

Get external axis max position

Returns

External axis max position

8.2.2.12 getExtAxisMaxVelocity()

double arcs::common_interface::AxisInterface::getExtAxisMaxVelocity ()

Get external axis max speed

Returns

External axis max speed

8.2.2.13 getExtAxisMountingPose()

```
std::vector< double > arcs::common_interface::AxisInterface::getExtAxisMountingPose ()
```

Get external axis mounting pose

Returns

External axis pose

8.2.2.14 getExtAxisPose()

```
std::vector< double > arcs::common_interface::AxisInterface::getExtAxisPose ()
```

Get pose wrt mounting coordinate system, axis can be positioner or linear rail

Returns

Pose wrt mounting coordinate system

8.2.2.15 getExtAxisPosition()

```
double arcs::common_interface::AxisInterface::getExtAxisPosition ()
```

Get external axis position

Returns

External axis position

8.2.2.16 getExtAxisTemperature()

```
double arcs::common_interface::AxisInterface::getExtAxisTemperature ()
```

Get external axis temperature

Returns

External axis temperature

8.2.2.17 getExtAxisType()

```
int arcs::common_interface::AxisInterface::getExtAxisType ()
```

Get external axis type: 0 for rotation, 1 for linear.

Returns

8.2.2.18 getExtAxisVelocity()

```
double arcs::common_interface::AxisInterface::getExtAxisVelocity ()
```

Get external axis speed

Returns

External axis speed

8.2.2.19 getExtMinPosition()

```
double arcs::common_interface::AxisInterface::getExtMinPosition ()
```

Get external axis min position

Returns

External axis min position

8.2.2.20 moveExtJoint()

```
int arcs::common_interface::AxisInterface::moveExtJoint (
    double pos,
    double v,
    double a,
    double duration)
```

move to pos, rotation or linear

Parameters

pos	
v	
a	
duration	

Returns

8.2.2.21 poweroffExtAxis()

```
int arcs::common_interface::AxisInterface::poweroffExtAxis ()
```

Power off.

Returns

8.2.2.22 poweronExtAxis()

```
int arcs::common_interface::AxisInterface::poweronExtAxis ()
```

Power on.

Returns

8.2.2.23 setExtAxisMountingPose()

```
int arcs::common_interface::AxisInterface::setExtAxisMountingPose (
    const std::vector< double > & pose)
```

Set mounting pose of external axis (wrt world frame).

Parameters

pose	
------	--

Returns

8.2.2.24 speedExtJoint()

```
int arcs::common_interface::AxisInterface::speedExtJoint (
    double v,
    double a,
    double duration)
```

Set target speed, acceleration and duration

Parameters

v	
a	
duration	

Returns

8.2.2.25 stopExtJoint()

```
int arcs::common_interface::AxisInterface::stopExtJoint (
    double a)
```

stop ext joint

Parameters

a	
---	--

Returns

8.2.2.26 stopFollowAnotherAxis()

```
int arcs::common_interface::AxisInterface::stopFollowAnotherAxis ()
```

stopFollowAnotherAxis(not to be used during motion)

Returns

8.3 GripperInterface ()

Functions

- `std::vector< std::string > arcs::common_interface::GripperInterface::gripperGetSupportedModels()`
- `ResultWithErrno2 arcs::common_interface::GripperInterface::gripperScanDevices (const std::string &model, const std::string &device_name)`
- `std::vector< std::string > arcs::common_interface::GripperInterface::gripperGetNames ()`
- `int arcs::common_interface::GripperInterface::gripperAdd (const std::string &name, const std::string &model)`
- `int arcs::common_interface::GripperInterface::gripperDelete (const std::string &name)`
- `int arcs::common_interface::GripperInterface::gripperRename (const std::string &name, const std::string &new_name)`
- `int arcs::common_interface::GripperInterface::gripperConnect (const std::string &name, const std::string &device_name)`
- `int arcs::common_interface::GripperInterface::gripperDisconnect (const std::string &name)`
- `bool arcs::common_interface::GripperInterface::gripperIsConnected (const std::string &name)`
- `int arcs::common_interface::GripperInterface::gripperSetWorkMode (const std::string &name, int work_mode)`
- `int arcs::common_interface::GripperInterface::gripperGetWorkMode (const std::string &name)`
- `int arcs::common_interface::GripperInterface::gripperSetMountPose (const std::string &name, const std::vector< double > &pose, bool enable_collision)`
- `int arcs::common_interface::GripperInterface::gripperEnable (const std::string &name, bool enable)`
- `bool arcs::common_interface::GripperInterface::gripperIsEnabled (const std::string &name)`
- `int arcs::common_interface::GripperInterface::gripperSetPosition (const std::string &name, const double position)`
- `int arcs::common_interface::GripperInterface::gripperMove (const std::string &name)`
- `int arcs::common_interface::GripperInterface::gripperStop (const std::string &name)`
- `std::string arcs::common_interface::GripperInterface::gripperGetHardwareVersion (const std::string &name)`
- `int arcs::common_interface::GripperInterface::gripperResetSlaveId (const std::string &name, const int slave_id)`
- `int arcs::common_interface::GripperInterface::gripperGetStatusCode (const std::string &name)`

8.3.1 Detailed Description

8.3.2 Function Documentation

8.3.2.1 gripperAdd()

```
int arcs::common_interface::GripperInterface::gripperAdd (
    const std::string & name,
    const std::string & model)
```

8.3.2.2 gripperConnect()

```
int arcs::common_interface::GripperInterface::gripperConnect (
    const std::string & name,
    const std::string & device_name)
```

8.3.2.3 gripperDelete()

```
int arcs::common_interface::GripperInterface::grripperDelete (  
    const std::string & name)
```

8.3.2.4 gripperDisconnect()

```
int arcs::common_interface::GripperInterface::grripperDisconnect (  
    const std::string & name)
```

8.3.2.5 gripperEnable()

```
int arcs::common_interface::GripperInterface::grripperEnable (  
    const std::string & name,  
    bool enable)
```

8.3.2.6 gripperGetHardwareVersion()

```
std::string arcs::common_interface::GripperInterface::grripperGetHardwareVersion (  
    const std::string & name)
```

8.3.2.7 gripperGetNames()

```
std::vector< std::string > arcs::common_interface::GripperInterface::grripperGetNames ()
```

8.3.2.8 gripperGetStatusCode()

```
int arcs::common_interface::GripperInterface::grripperGetStatusCode (  
    const std::string & name)
```

8.3.2.9 gripperGetSupportedModels()

```
std::vector< std::string > arcs::common_interface::GripperInterface::grripperGetSupportedModels ()
```

8.3.2.10 gripperGetWorkMode()

```
int arcs::common_interface::GripperInterface::grripperGetWorkMode (  
    const std::string & name)
```

8.3.2.11 gripperIsConnected()

```
bool arcs::common_interface::GripperInterface::grripperIsConnected (  
    const std::string & name)
```


8.3.2.12 gripperIsEnabled()

```
bool arcs::common_interface::GripperInterface::gripperIsEnabled (
    const std::string & name)
```

8.3.2.13 gripperMove()

```
int arcs::common_interface::GripperInterface::gripperMove (
    const std::string & name)
```

8.3.2.14 gripperRename()

```
int arcs::common_interface::GripperInterface::gripperRename (
    const std::string & name,
    const std::string & new_name)
```

8.3.2.15 gripperResetSlaveId()

```
int arcs::common_interface::GripperInterface::gripperResetSlaveId (
    const std::string & name,
    const int slave_id)
```

8.3.2.16 gripperScanDevices()

```
ResultWithErrno2 arcs::common_interface::GripperInterface::gripperScanDevices (
    const std::string & model,
    const std::string & device_name)
```

8.3.2.17 gripperSetMountPose()

```
int arcs::common_interface::GripperInterface::gripperSetMountPose (
    const std::string & name,
    const std::vector< double > & pose,
    bool enable_collision)
```

8.3.2.18 gripperSetPosition()

```
int arcs::common_interface::GripperInterface::gripperSetPosition (
    const std::string & name,
    const double position)
```

8.3.2.19 gripperSetWorkMode()

```
int arcs::common_interface::GripperInterface::gripperSetWorkMode (
    const std::string & name,
    int work_mode)
```

8.3.2.20 gripperStop()

```
int arcs::common_interface::GripperInterface::gripperStop (
    const std::string & name)
```

8.4 Math ()

Functions

- `std::vector< double > arcs::common_interface::Math::poseAdd` (const `std::vector< double > &p1`, const `std::vector< double > &p2`)
Pose addition
- `std::vector< double > arcs::common_interface::Math::poseSub` (const `std::vector< double > &p1`, const `std::vector< double > &p2`)
Pose subtraction
- `std::vector< double > arcs::common_interface::Math::interpolatePose` (const `std::vector< double > &p1`, const `std::vector< double > &p2`, double `alpha`)
Calculate linear interpolation
- `std::vector< double > arcs::common_interface::Math::poseTrans` (const `std::vector< double > &pose_from`, const `std::vector< double > &pose_to`)
Pose transformation
- `std::vector< double > arcs::common_interface::Math::poseTransInv` (const `std::vector< double > &pose_from`, const `std::vector< double > &pose_to`)
Pose inverse transformation
- `std::vector< double > arcs::common_interface::Math::poseInverse` (const `std::vector< double > &pose`)
Get the inverse of a pose
- `double arcs::common_interface::Math::poseDistance` (const `std::vector< double > &p1`, const `std::vector< double > &p2`)
Calculate distance between two poses
- `double arcs::common_interface::Math::poseAngleDistance` (const `std::vector< double > &p1`, const `std::vector< double > &p2`)
Calculate axis-angle difference between two poses
- `bool arcs::common_interface::Math::poseEqual` (const `std::vector< double > &p1`, const `std::vector< double > &p2`, double `eps=5e-5`)
Determine if two poses are equivalent
- `std::vector< double > arcs::common_interface::Math::transferRefFrame` (const `std::vector< double > &F_b_a_old`, const `Vector3d &V_in_a`, int `type`)
- `std::vector< double > arcs::common_interface::Math::poseRotation` (const `std::vector< double > &pose`, const `std::vector< double > &rotrv`)
Pose rotation
- `std::vector< double > arcs::common_interface::Math::rpyToQuaternion` (const `std::vector< double > &rpy`)
Euler angles to quaternions
- `std::vector< double > arcs::common_interface::Math::quaternionToRpy` (const `std::vector< double > &quat`)
Quaternions to euler angles
- `ResultWithErrno arcs::common_interface::Math::tcpOffsetIdentify` (const `std::vector< std::vector< double > > &poses`)
Four point method calibration for TCP offset

- [ResultWithErrno arcs::common_interface::Math::calibrateCoordinate](#) (const std::vector< std::vector< double > > &poses, int type)
Calibrate coordinate system with 3 points
- [ResultWithErrno arcs::common_interface::Math::calculateCircleFourthPoint](#) (const std::vector< double > &p1, const std::vector< double > &p2, const std::vector< double > &p3, int mode)
Based on three points on an arc, calculate the position of the midpoint of the other half of the fitted circle's arc
- std::vector< double > [arcs::common_interface::Math::forceTrans](#) (const std::vector< double > &pose_a_in_b, const std::vector< double > &force_in_a)
- std::vector< double > [arcs::common_interface::Math::getDeltaPoseBySensorDistance](#) (const std::vector< double > &distances, double position, double radius, double track_scale)
- std::vector< double > [arcs::common_interface::Math::deltaPoseTrans](#) (const std::vector< double > &pose_a_in_b, const std::vector< double > &ft_in_a)
- std::vector< double > [arcs::common_interface::Math::deltaPoseAdd](#) (const std::vector< double > &pose_a_in_b, const std::vector< double > &v_in_b)
- std::vector< double > [arcs::common_interface::Math::changePoseWithXYRef](#) (const std::vector< double > &pose_tar, const std::vector< double > &pose_ref)
- std::vector< double > [arcs::common_interface::Math::homMatrixToPose](#) (const std::vector< double > &homMatrix)
- std::vector< double > [arcs::common_interface::Math::poseToHomMatrix](#) (const std::vector< double > &pose)

8.4.1 Detailed Description

8.4.2 Function Documentation

8.4.2.1 calculateCircleFourthPoint()

[ResultWithErrno arcs::common_interface::Math::calculateCircleFourthPoint](#) (
const std::vector< double > & p1,
const std::vector< double > & p2,
const std::vector< double > & p3,
int mode)

Based on three points on an arc, calculate the position of the midpoint of the other half of the fitted circle's arc

Parameters

p1	start point of the arc
p2	middle point of the arc
p3	end point of the arc
mode	when mode = 1, need to plan for orientation around arc; when mode = 0, do not need to plan for orientation around arc.

Returns

position of the midpoint of the other half of the fitted circle's arc and whether the result is valid.

Lua

```
calculateCircleFourthPoint(p1: table, p2: table, p3: table, mode: int)
```

Lua

```
center_pose, cal_result = calculateCircleFourthPoint({0.5488696249770836,-0.1214996547187204,0.2631931199112321,-3.14159198038469,-3.673205103150083e-06,1.570796326792424}, {0.5488696249770835,-0.1214996547187207,0.3599720701808493,-3.14159198038469,-3.6732051029273e-06,1.570796326792423}, {0.5488696249770836,-0.0389996547187214,0.3599720701808496,-3.141591980384691,-3.673205102557476e-06,1.570796326792422},1)
```

JSON-RPC request example

```
{ "jsonrpc": "2.0", "method": "Math.calculateCircleFourthPoint", "params": [[0.5488696249770836,-0.1214996547187204,0.2631931199112321,-3.14159198038469,-3.673205103150083e-06,1.570796326792424], [0.5488696249770835,-0.1214996547187207,0.3599720701808493,-3.14159198038469,-3.6732051029273e-06,1.570796326792423], [0.5488696249770836,-0.0389996547187214,0.3599720701808496,-3.141591980384691,-3.673205102557476e-06,1.570796326792422],1], "id": 1 }
```

JSON-RPC response example

```
{ "id": 1, "jsonrpc": "2.0", "result": [[0.5488696249770837,-0.031860179583911546,0.27033259504604207,-3.1415919803846903,-3.67320510285378e-06,1.570796326792423],1] }
```

8.4.2.2 calibrateCoordinate()

```
ResultWithErrno arcs::common_interface::Math::calibrateCoordinate (
    const std::vector< std::vector< double > > & poses,
    int type)
```

Calibrate coordinate system with 3 points

Parameters

poses	set of 3 poses
type	type: 0 - oxy origin, +x axis, xy plane (+y direction) 1 - oxz origin, +x axis, xz plane (+z direction) 2 - oyz origin, +y axis, yz plane (+z direction) 3 - oyx origin, +y axis, yx plane (+x direction) 4 - ozx origin, +z axis, zx plane (+x direction) 5 - ozy origin, +z axis, zy plane (+y direction)

Returns

Coordinate system calibration result and whether the calibration result is valid

Lua function prototype

```
calibrateCoordinate(poses: table, type: int) -> table, number
```

Lua example

```
p1 = {0.55462,0.06219,0.37175,-3.142,0.0,1.580}
p2 = {0.63746,0.11805,0.37175,-3.142,0.0,1.580}
p3 = {0.40441,0.28489,0.37174,-3.142,0.0,1.580}
p = {p1,p2,p3}
coord_pose, cal_result = calibrateCoordinate(p,0)
```

JSON-RPC request example

```
{ "jsonrpc": "2.0", "method": "Math.calibrateCoordinate", "params": [[0.55462,0.06219,0.37175,-3.142,0.0,1.580], [0.63746,0.11805,0.37175,-3.142,0.0,1.580], [0.40441,0.28489,0.37174,-3.142,0.0,1.580]], 0], "id": 1 }
```

JSON-RPC response example

```
{ "id": 1, "jsonrpc": "2.0", "result": [[0.55462,0.06219,0.37175,-3.722688983883945e-05,-1.6940658945086007e-21,0.5932768162455785],0] }
```

8.4.2.3 changePoseWithXYRef()

```
std::vector< double > arcs::common_interface::Math::changePoseWithXYRef (
    const std::vector< double > & pose_tar,
    const std::vector< double > & pose_ref)
```

changePoseWithXYRef: Modify the XY axis direction of pose_tar to be as consistent as possible with pose_ref

Parameters

pose_tar	target pose to be modified
pose_ref	reference pose

Returns

Modified pose, using the xyz coordinates and z axis direction of pose_tar

8.4.2.4 deltaPoseAdd()

```
std::vector< double > arcs::common_interface::Math::deltaPoseAdd (
    const std::vector< double > & pose_a_in_b,
    const std::vector< double > & v_in_b)
```

addDeltaPose: Calculate the pose after unit time given a velocity

Parameters

pose_a_in↔ _b	current pose of a relative to b
v_in_b	velocity of frame a described in frame b at current time

Returns

pose_in_b, pose after unit time described in frame b

8.4.2.5 deltaPoseTrans()

```
std::vector< double > arcs::common_interface::Math::deltaPoseTrans (
    const std::vector< double > & pose_a_in_b,
    const std::vector< double > & ft_in_a)
```

changeFTFrame: Transform the reference frame of force and torque

Parameters

pose_a_in↔ _b	pose of frame a in frame b
ft_in_a	force and torque applied at point a, described in frame a

Returns

ft_in_b, force and torque applied at point b, described in frame b

8.4.2.6 forceTrans()

```
std::vector< double > arcs::common_interface::Math::forceTrans (
    const std::vector< double > & pose_a_in_b,
    const std::vector< double > & force_in_a)
```

forceTrans: Transform the reference frame of force and torque: $\text{force_in_b} = \text{pose_a_in_b} * \text{force_in_a}$

Parameters

pose_a_in↔ _b	pose of frame a in frame b
force_in_a	force and torque described in frame a

Returns

Force_in_b, force and torque described in frame b

8.4.2.7 getDeltaPoseBySensorDistance()

```
std::vector< double > arcs::common_interface::Math::getDeltaPoseBySensorDistance (
    const std::vector< double > & distances,
    double position,
    double radius,
    double track_scale)
```

Calculate pose increment in tool coordinate system based on sensor data

Parameters

distances	N distances, N >= 3
position	reference height to maintain from the trajectory
radius	effective radius from sensor center to tool TCP
track_scale	tracking ratio, range (0, 1], 1 means faster tracking

Returns

Pose increment in tool coordinate system

8.4.2.8 homMatrixToPose()

```
std::vector< double > arcs::common_interface::Math::homMatrixToPose (
    const std::vector< double > & homMatrix)
```

homMatrixToPose: Get pose from homogeneous transformation matrix

Parameters

homMatrix	4x4 homogeneous transformation matrix, input elements are arranged row-wise
-----------	---

Returns

corresponding pose

8.4.2.9 interpolatePose()

```
std::vector< double > arcs::common_interface::Math::interpolatePose (
    const std::vector< double > & p1,
    const std::vector< double > & p2,
    double alpha)
```

Calculate linear interpolation

Parameters

p1	starting TCP pose
p2	ending TCP pose
alpha	coefficient; When $0 < \alpha < 1$ return a point between p1 & p2 that is closer to p1, at alpha percentage of the path For example when $\alpha = 0.3$, point returned is closer to p1 at 30% of the total distance; When $\alpha > 1$, return p2 When $\alpha < 0$, return p1

Returns

interpolation result

Python interface prototype

```
interpolatePose(self: pyaubo_sdk.Math, arg0: List[float], arg1: List[float], arg2: float) -> List<
[float]
```

Lua interface prototype

```
interpolatePose(p1: table, p2: table, alpha: number) -> table
```

Lua example

```
pose_interpolate = interpolatePose({0.2, 0.2, 0.4, 0, 0, 0},{0.2, 0.2, 0.6, 0, 0, 0},0.5)
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"Math.interpolatePose","params":[[0.2, 0.2, 0.4, 0, 0, 0],[0.2, 0.2, 0.6, 0, 0, 0],0.5],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":[0.2,0.2,0.5,0.0,-0.0,0.0]}
```


8.4.2.10 poseAdd()

```
std::vector< double > arcs::common_interface::Math::poseAdd (
    const std::vector< double > & p1,
    const std::vector< double > & p2)
```

Pose addition

Both arguments contain three position parameters (x, y, z) jointly called P, and three rotation parameters (R_x, R_y, R_z) jointly called R. This function calculates the result x_3 as the addition of the given poses as follows:

$$p_3.P = p_1.P + p_2.P$$

$$p_3.R = p_1.R * p_2.R$$

Parameters

p1	Tool pose 1
p2	Tool pose 2

Returns

sum of position parts and product of rotation parts (pose)

Python interface prototype

```
poseAdd(self: pyaubo_sdk.Math, arg0: List[float], arg1: List[float]) -> List[float]
```

Lua interface prototype

```
poseAdd(p1: table, p2: table) -> table
```

Lua example

```
pose_add = poseAdd({0.2, 0.5, 0.1, 1.57, 0, 0},{0.2, 0.5, 0.6, 1.57, 0, 0})
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"Math.poseAdd","params":[[0.2, 0.5, 0.1, 1.57, 0, 0],[0.2, 0.5, 0.6, 1.57, 0, 0]],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":[0.4,1.0,0.7,3.14,-0.0,0.0]} \endenglish
```

Parameters

p1	pose 1
p2	pose 2

Returns

axis angle difference

Lua

poseAngleDistance(p1: table, p2: table) -> number

Lua

```
pose_angle_distance = poseAngleDistance({0.1, 0.3, 0.1, 0.3142, 0.0, 1.571},{0.2, 0.5, 0.6, 0, -0.172, 0.0})
```

8.4.2.12 poseDistance()

```
double arcs::common_interface::Math::poseDistance (
    const std::vector< double > & p1,
    const std::vector< double > & p2)
```

Calculate distance between two poses

Parameters

p1	pose 1
p2	pose 2

Returns

distance between the poses

Lua function prototype

poseDistance(p1: table, p2: table) -> number

Lua example

```
pose_distance = poseDistance({0.1, 0.3, 0.1, 0.3142, 0.0, 1.571},{0.2, 0.5, 0.6, 0, -0.172, 0.0})
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"Math.poseDistance","params":[[0.1, 0.3, 0.1, 0.3142, 0.0, 1.571],[0.2, 0.5, 0.6, 0, -0.172, 0.0]],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":0.5477225575051661}
```

8.4.2.13 poseEqual()

```
bool arcs::common_interface::Math::poseEqual (
    const std::vector< double > & p1,
    const std::vector< double > & p2,
    double eps = 5e-5)
```

Determine if two poses are equivalent

Parameters

p1	pose 1
p2	pose 2
eps	error margin

Returns

true or false

Lua

```
poseEqual(p1: table, p2: table, eps: number) -> boolean
```

Lua

```
pose_equal = poseEqual({0.1, 0.3, 0.1, 0.3142, 0.0, 1.571},{0.1, 0.3, 0.1, 0.3142, 0.0, 1.5711},0.01)
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"Math.poseEqual","params":[[0.1, 0.3, 0.1, 0.3142, 0.0, 1.571],[0.1, 0.3, 0.1, 0.3142, 0.0, 1.5711],0.01],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":0.0}
```

8.4.2.14 poseInverse()

```
std::vector< double > arcs::common_interface::Math::poseInverse (
    const std::vector< double > & pose)
```

Get the inverse of a pose

Parameters

pose	tool pose (spatial vector)
------	----------------------------

Returns

inverse tool pose transformation (spatial vector)

Python interface prototype

```
poseInverse(self: pyaubo_sdk.Math, arg0: List[float]) -> List[float]
```

Lua interface prototype

```
poseInverse(pose: table) -> table
```

Lua example

```
pose_inverse = poseInverse({0.2, 0.5, 0.1, 1.57, 0, 3.14})
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"Math.poseInverse","params":[[0.2, 0.5, 0.1, 1.57, 0, 3.14]],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":[0.19920341988726448,-0.09960155178838484,-0.5003973704832628,1.5699999989900404,-0.0015926530848129354,-3.1415913853161266]}
```

8.4.2.15 poseRotation()

```
std::vector< double > arcs::common_interface::Math::poseRotation (
    const std::vector< double > & pose,
    const std::vector< double > & rotv)
```

Pose rotation

Parameters

pose	
rotv	

Returns

Python interface prototype

```
poseRotation(self: pyaubo_sdk.Math, arg0: List[float], arg1: List[float]) -> List[float]
```

Lua interface prototype

```
poseRotation(pose: table, rotv: table) -> table
```

8.4.2.16 poseSub()

```
std::vector< double > arcs::common_interface::Math::poseSub (
    const std::vector< double > & p1,
    const std::vector< double > & p2)
```

Pose subtraction

Both arguments contain three position parameters (x, y, z) jointly called P, and three rotation parameters (R_x, R_y, R_z) jointly called R. This function calculates the result x_3 as the addition of the given poses as follows:

$$p_3.P = p_1.P - p_2.P,$$

$$p_3.R = p_1.R * p_2.R.inverse$$

Parameters

p1	tool pose 1
p2	tool pose 2

Returns

difference between two poses

Python interface prototype

```
poseSub(self: pyaubo_sdk.Math, arg0: List[float], arg1: List[float]) -> List[float]
```

Lua interface prototype

```
poseSub(p1: table, p2: table) -> table
```

Lua example

```
pose_sub = poseSub({0.2, 0.5, 0.1, 1.57, 0, 0},{0.2, 0.5, 0.6, 1.57, 0, 0})
```

JSON-RPC request example

```
{ "jsonrpc": "2.0", "method": "Math.poseSub", "params": [[0.2, 0.5, 0.1, 1.57, 0, 0], [0.2, 0.5, 0.6, 1.57, 0, 0]], "id": 1 }
```

JSON-RPC response example

```
{ "id": 1, "jsonrpc": "2.0", "result": [0.0, 0.0, -0.5, 0.0, -0.0, 0.0] }
```

Parameters

pose	input pose
------	------------

Returns

output homogeneous transformation matrix, elements arranged row-wise

8.4.2.18 poseTrans()

```
std::vector< double > arcs::common_interface::Math::poseTrans (
    const std::vector< double > & pose_from,
    const std::vector< double > & pose_from_to)
```

Pose transformation

The first argument, `p_from`, is used to transform the second argument, `p_from_to`, and the result is then returned. This means that the result is the resulting pose, when starting at the coordinate system of `p_from`, and then in that coordinate system moving `p_from_to`.

This function can be seen in two different views. Either the function transforms, that is translates and rotates, `p_from_to` by the parameters of `p_from`. Or the function is used to get the resulting pose, when first making a move of `p_from` and then from there, a move of `p_from_to`. If the poses were regarded as transformation matrices, it would look like:

$$T_{\text{world} \rightarrow \text{to}} = T_{\text{world} \rightarrow \text{from}} * T_{\text{from} \rightarrow \text{to}} \quad T_{\text{x} \rightarrow \text{to}} = T_{\text{x} \rightarrow \text{from}} * T_{\text{from} \rightarrow \text{to}}$$

These two equations describes the foundations for pose transformation. Based on a starting pose and the pose transformation relative to the starting pose, we can get the target pose.

For example, we know pose of B relative to A, pose of C relative to B, find pose of C relative to A. param 1 is pose of B relative to A param 2 is pose of C relative to B the return value is the pose of C relative to A

Parameters

<code>pose_from</code>	starting pose vector in 3D space
<code>pose_from_to</code>	pose transformation relative to starting pose vector in 3D space

Returns

final pose (vector in 3D space)

Python interface prototype

```
poseTrans(self: pyaubo_sdk.Math, arg0: List[float], arg1: List[float]) -> List[float]
```

Lua interface prototype

```
poseTrans(pose_from: table, pose_to: table) -> table
```

Lua example

```
pose_trans = poseTrans({0.2, 0.5, 0.1, 1.57, 0, 0},{0.2, 0.5, 0.6, 1.57, 0, 0})
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"Math.poseTrans","params":[[0.2, 0.5, 0.1, 1.57, 0, 0],[0.2, 0.5, 0.6, 1.57, 0, 0]],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":[0.4,-0.09960164640373415,0.6004776374923573,3.14,-0.0,0.0]}
```

8.4.2.19 poseTransInv()

```
std::vector< double > arcs::common_interface::Math::poseTransInv (
    const std::vector< double > & pose_from,
    const std::vector< double > & pose_to_from)
```

Pose inverse transformation

Given pose of C relative to A, pose of C relative to B, find pose of B relative to A. param 1 is pose of C relative to A param 2 is pose of C relative to B the return value is the pose of B relative to A

Parameters

pose_from	starting pose
pose_to_from	pose transformation relative to final pose

Returns

resulting pose

Python interface prototype

```
poseTransInv(self: pyaubo_sdk.Math, arg0: List[float], arg1: List[float]) -> List[float]
```

Lua interface prototype

```
poseTransInv(pose_from: table, pose_to_from: table) -> table
```

Lua example

```
pose_trans_inv = poseTransInv({0.4, -0.0996016, 0.600478, 3.14, 0, 0},{0.2, 0.5, 0.6, 1.57, 0, 0})
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"Math.poseTransInv","params":[[0.4, -0.0996016, 0.600478, 3.14, 0, 0],[0.2, 0.5, 0.6, 1.57, 0, 0]],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":[0.2,0.5000000464037341,0.10000036250764266,1.57,-0.0,0.0]}
```

8.4.2.20 quaternionToRpy()

```
std::vector< double > arcs::common_interface::Math::quaternionToRpy (
    const std::vector< double > & quat)
```

Quaternions to euler angles

Parameters

quat	quaternions
------	-------------

Returns

euler angles

Python interface prototype

```
quaternionToRpy(self: pyaubo_sdk.Math, arg0: List[float]) -> List[float]
```

Lua interface prototype

```
quaternionToRpy(quat: table) -> table
```

Lua example

```
rpm = quaternionToRpy({0.834722,0.0780426, 0.451893, 0.304864})
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"Math.quaternionToRpy","params":[[0.834722, 0.0780426, 0.451893, 0.304864]],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":[0.6110000520523781,0.7849996877683915,0.960000543982093]}
```

8.4.2.21 rpyToQuaternion()

```
std::vector< double > arcs::common_interface::Math::rpyToQuaternion (
    const std::vector< double > & rpy)
```

Euler angles to quaternions

Parameters

rpy	euler angles
-----	--------------

Returns

quaternions

Python interface prototype

```
rpyToQuaternion(self: pyaubo_sdk.Math, arg0: List[float]) -> List[float]
```

Lua interface prototype

```
rpyToQuaternion(rpy: table) -> table
```

Lua example

```
quaternion = rpyToQuaternion({0.611, 0.785, 0.960})
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"Math.rpyToQuaternion","params":[[0.611, 0.785, 0.960]],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":[0.834721517970497,0.07804256900772265,0.4518931575790371,0.↵
3048637712043723]}
```

8.4.2.22 tcpOffsetIdentify()

```
ResultWithErrno arcs::common_interface::Math::tcpOffsetIdentify (
    const std::vector< std::vector< double > > & poses)
```

Four point method calibration for TCP offset

About a sharp point, move the robot's tcp in four different poses. Difference between each pose should be drastic. Result can be obtained based on these four poses

Parameters

poses	combination of four different poses
-------	-------------------------------------

Returns

TCP calibration result and whether successfull

Python interface prototype

```
tcpOffsetIdentify(self: pyaubo_sdk.Math, arg0: List[List[float]]) -> Tuple[List[float], int]
```

Lua interface prototype

```
tcpOffsetIdentify(poses: table) -> table
```

Lua example

```
p1 = {0.48659,0.10456,0.24824,-3.135,0.004,1.569}
p2 = {0.38610,0.09096,0.22432,2.552,-0.437,1.008}
p3 = {0.38610,0.05349,0.16791,-3.021,-0.981,0.517}
p4 = {0.49675,0.06417,0.21408,-2.438,0.458,0.651}
p = {p1,p2,p3,p4}
tcp_result = tcpOffsetIdentify(p)
```

8.4.2.23 transferRefFrame()

```
std::vector< double > arcs::common_interface::Math::transferRefFrame (
    const std::vector< double > & F_b_a_old,
    const Vector3d & V_in_a,
    int type)
```

Parameters

F_b_a_old	
V_in_a	
type	

Returns

Python interface prototype

```
transferRefFrame(self: pyaubo_sdk.Math, arg0: List[float], arg1: List[float[3]], arg2: int) -> List<[float]
```

Lua interface prototype

```
transferRefFrame(F_b_a_old: table, V_in_a: table, type: number) -> table
```

8.5 RegisterControl ()

General Registers.

Functions

- `bool arcs::common_interface::RegisterControl::getBoolInput (uint32_t address)`
Reads the boolean from one of the input registers, which can also be accessed by a Field bus.
- `int arcs::common_interface::RegisterControl::setBoolInput (uint32_t address, bool value)`
- `int arcs::common_interface::RegisterControl::getInt32Input (uint32_t address)`
Reads the integer from one of the input registers, which can also be accessed by a FieldBus.
- `int arcs::common_interface::RegisterControl::setInt32Input (uint32_t address, int value)`
- `float arcs::common_interface::RegisterControl::getFloatInput (uint32_t address)`
Reads the float from one of the input registers, which can also be accessed by a Field bus.
- `int arcs::common_interface::RegisterControl::setFloatInput (uint32_t address, float value)`
- `double arcs::common_interface::RegisterControl::getDoubleInput (uint32_t address)`
Reads the double value from one of the input registers, which can also be accessed by a FieldBus.
- `int arcs::common_interface::RegisterControl::setDoubleInput (uint32_t address, double value)`
- `bool arcs::common_interface::RegisterControl::getBoolOutput (uint32_t address)`
Reads the boolean from one of the output registers, which can also be accessed by a Field bus.
- `int arcs::common_interface::RegisterControl::setBoolOutput (uint32_t address, bool value)`
- `int arcs::common_interface::RegisterControl::getInt32Output (uint32_t address)`
Reads the integer from one of the output registers, which can also be accessed by a FieldBus.
- `int arcs::common_interface::RegisterControl::setInt32Output (uint32_t address, int value)`
- `float arcs::common_interface::RegisterControl::getFloatOutput (uint32_t address)`
Reads the float from one of the output registers, which can also be accessed by a FieldBus.
- `int arcs::common_interface::RegisterControl::setFloatOutput (uint32_t address, float value)`
- `double arcs::common_interface::RegisterControl::getDoubleOutput (uint32_t address)`
Reads the double value from one of the output registers.
- `int arcs::common_interface::RegisterControl::setDoubleOutput (uint32_t address, double value)`
- `int16_t arcs::common_interface::RegisterControl::getInt16Register (uint32_t address)`
Used for Modbus Slave
- `int arcs::common_interface::RegisterControl::setInt16Register (uint32_t address, int16_t value)`
- `bool arcs::common_interface::RegisterControl::getInt16RegisterBit (uint32_t address, uint8_t bit_offset)`
Get the status of a specific bit in an Int16 register
- `int arcs::common_interface::RegisterControl::setInt16RegisterBit (uint32_t address, uint8_t bit_offset, bool value)`
Set the status of a specific bit in an Int16 register
- `bool arcs::common_interface::RegisterControl::hasNamedVariable (const std::string &key)`
Whether the named variable exists
- `std::string arcs::common_interface::RegisterControl::getNamedVariableType (const std::string &key)`
Get the type of a named variable
- `bool arcs::common_interface::RegisterControl::variableUpdated (const std::string &key, uint64_t since)`
Whether the named variable has been updated
- `bool arcs::common_interface::RegisterControl::getBool (const std::string &key, bool default_value)`
Get variable value
- `int arcs::common_interface::RegisterControl::setBool (const std::string &key, bool value)`
Set or update the variable value

- `std::vector< char > arcs::common_interface::RegisterControl::getVecChar` (const std::string &key, const std::vector< char > &default_value)
Get variable value
- `int arcs::common_interface::RegisterControl::setVecChar` (const std::string &key, const std::vector< char > &value)
Set or update the variable value
- `int arcs::common_interface::RegisterControl::getInt32` (const std::string &key, int default_value)
Get variable value
- `int arcs::common_interface::RegisterControl::setInt32` (const std::string &key, int value)
Set or update the variable value
- `std::vector< int32_t > arcs::common_interface::RegisterControl::getVecInt32` (const std::string &key, const std::vector< int32_t > &default_value)
Get variable value
- `int arcs::common_interface::RegisterControl::setVecInt32` (const std::string &key, const std::vector< int32_t > &value)
Set or update the variable value
- `float arcs::common_interface::RegisterControl::getFloat` (const std::string &key, float default_value)
Get variable value
- `int arcs::common_interface::RegisterControl::setFloat` (const std::string &key, float value)
Set or update the variable value
- `std::vector< float > arcs::common_interface::RegisterControl::getVecFloat` (const std::string &key, const std::vector< float > &default_value)
Get variable value
- `int arcs::common_interface::RegisterControl::setVecFloat` (const std::string &key, const std::vector< float > &value)
Set or update the variable value
- `double arcs::common_interface::RegisterControl::getDouble` (const std::string &key, double default_value)
Get variable value
- `int arcs::common_interface::RegisterControl::setDouble` (const std::string &key, double value)
Set or update the variable value
- `std::vector< double > arcs::common_interface::RegisterControl::getVecDouble` (const std::string &key, const std::vector< double > &default_value)
Get variable value
- `int arcs::common_interface::RegisterControl::setVecDouble` (const std::string &key, const std::vector< double > &value)
Set or update the variable value
- `std::string arcs::common_interface::RegisterControl::getString` (const std::string &key, const std::string &default_value)
Get variable value
- `int arcs::common_interface::RegisterControl::setString` (const std::string &key, const std::string &value)
Set or update the variable value
- `int arcs::common_interface::RegisterControl::clearNamedVariable` (const std::string &key)
Clear variable
- `int arcs::common_interface::RegisterControl::setWatchDog` (const std::string &key, double timeout, int action)
Set the watchdog
- `int arcs::common_interface::RegisterControl::getWatchDogAction` (const std::string &key)
Get the watchdog action
- `int arcs::common_interface::RegisterControl::getWatchDogTimeout` (const std::string &key)

- Get the watchdog timeout value

 - int `arcs::common_interface::RegisterControl::modbusAddSignal` (const std::string &device_info, int slave_number, int signal_address, int signal_type, const std::string &signal_name, bool sequential_mode)

Adds a new modbus signal for the controller to supervise.
- int `arcs::common_interface::RegisterControl::modbusDeleteSignal` (const std::string &signal_name)

Deletes the signal identified by the supplied signal name.
- int `arcs::common_interface::RegisterControl::modbusDeleteAllSignals` ()

Delete all modbus signals
- int `arcs::common_interface::RegisterControl::modbusGetSignalStatus` (const std::string &signal_name)

Reads the current value of a specific signal.
- std::vector< std::string > `arcs::common_interface::RegisterControl::modbusGetSignalNames` ()

Get the collection of all signal names
- std::vector< int > `arcs::common_interface::RegisterControl::modbusGetSignalTypes` ()

Get the collection of all signal types
- std::vector< int > `arcs::common_interface::RegisterControl::modbusGetSignalValues` ()

Get the collection of all signal values
- std::vector< int > `arcs::common_interface::RegisterControl::modbusGetSignalErrors` ()

Get the error status of all signal requests (0: no error, others: error) as a collection
- int `arcs::common_interface::RegisterControl::modbusSendCustomCommand` (const std::string &device_info, int slave_number, int function_code, const std::vector< uint8_t > &data)

Sends a command specified by the user to the modbus unit located on the specified IP address.
- int `arcs::common_interface::RegisterControl::modbusSetDigitalInputAction` (const std::string &robot_name, const std::string &signal_name, `StandardInputAction` action)

Sets the selected digital input signal to either a "default" or "freedrive" action.
- int `arcs::common_interface::RegisterControl::modbusSetOutputRunstate` (const std::string &robot_name, const std::string &signal_name, `StandardOutputRunState` runstate)

Set Modbus signal output action
- int `arcs::common_interface::RegisterControl::modbusSetOutputSignal` (const std::string &signal_name, uint16_t value)

Sets the output register signal identified by the given name to the given value.
- int `arcs::common_interface::RegisterControl::modbusSetOutputSignal1` (const std::string &signal_name, const std::vector< uint16_t > &values)

Sets multiple consecutive output register signals starting from the given name to the given values.
- int `arcs::common_interface::RegisterControl::modbusSetOutputSignalPulse` (const std::string &signal_name, uint16_t value, double duration)

Set modbus signal output pulse (only supports coil output type)
- int `arcs::common_interface::RegisterControl::modbusSetSignalUpdateFrequency` (const std::string &signal_name, int update_frequency)

Sets the frequency with which the robot will send requests to the Modbus controller to either read or write the signal value.
- int `arcs::common_interface::RegisterControl::modbusGetSignalIndex` (const std::string &signal_name)

Get the index of the specified modbus signal, starting from 0.
- int `arcs::common_interface::RegisterControl::modbusGetSignalError` (const std::string &signal_name)

Get the error status of the specified modbus signal
- int `arcs::common_interface::RegisterControl::getModbusDeviceStatus` (const std::string &device_name)

Get the connection status of the specified modbus device

- int [arcs::common_interface::RegisterControl::addModbusEncoder](#) (int encoder_id, int range_id, const std::string &signal_name)
Use a modbus register signal as an encoder
- int [arcs::common_interface::RegisterControl::addInt32RegEncoder](#) (int encoder_id, int range_id, const std::string &key)
Add a virtual encoder for an Int32 register
- int [arcs::common_interface::RegisterControl::deleteVirtualEncoder](#) (int encoder_id)
Delete virtual encoder

8.5.1 Detailed Description

General Registers.

8.5.2 Function Documentation

8.5.2.1 addInt32RegEncoder()

```
int arcs::common_interface::RegisterControl::addInt32RegEncoder (
    int encoder_id,
    int range_id,
    const std::string & key)
```

Add a virtual encoder for an Int32 register

Parameters

encoder_id	Encoder ID
range_id	Range ID
key	Variable name

Returns

8.5.2.2 addModbusEncoder()

```
int arcs::common_interface::RegisterControl::addModbusEncoder (
    int encoder_id,
    int range_id,
    const std::string & signal_name)
```

Use a modbus register signal as an encoder

Parameters

encoder_id	Must not be 0
signal_name	Name of the modbus signal, must be of register type

Returns

8.5.2.3 clearNamedVariable()

```
int arcs::common_interface::RegisterControl::clearNamedVariable (  
    const std::string & key)
```

Clear variable

Parameters

key	
-----	--

Returns

Python interface prototype

```
clearNamedVariable(self: pyaubo_sdk.RegisterControl, arg0: str) -> int
```

Lua interface prototype

```
clearNamedVariable(key: string) -> nil
```

Lua example

```
clearNamedVariable("custom")
```

JSON-RPC request example

```
{"jsonrpc": "2.0", "method": "RegisterControl.clearNamedVariable", "params": ["custom"], "id": 1}
```

JSON-RPC response example

```
Generated by Doxygen  
{"id": 1, "jsonrpc": "2.0", "result": 1}
```

8.5.2.4 deleteVirtualEncoder()

Parameters

encoder_id	
------------	--

Returns

8.5.2.5 getBool()

```
bool arcs::common_interface::RegisterControl::getBool (
    const std::string & key,
    bool default_value)
```

Get variable value

Parameters

key	
default_value	

Returns

Python interface prototype

```
getBool(self: pyaubo_sdk.RegisterControl, arg0: str, arg1: bool) -> bool
```

Lua interface prototype

```
getBool(key: string, default_value: boolean) -> boolean
```

Lua example

```
Bool_var = getBool("custom",false)
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"RegisterControl.getBool","params":["custom",false],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":true}
```


Parameters

address	Address of the register (0:127)
---------	---------------------------------

Returns

Boolean value held by the register (true, false)

Note

The lower range of the boolean input registers [0:63] is reserved for FieldBus/PLC interface usage. The upper range [64:127] cannot be accessed by FieldBus/PLC interfaces, since it is reserved for external RTDE clients.

Python interface prototype

```
getBoolInput(self: pyaubo_sdk.RegisterControl, arg0: int) -> bool
```

Lua interface prototype

```
getBoolInput(address: number) -> boolean
```

Lua example

```
BoolInput_0 = getBoolInput(0)
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"RegisterControl.getBoolInput","params":[0],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":false}
```

8.5.2.7 getBoolOutput()

```
bool arcs::common_interface::RegisterControl::getBoolOutput (
    uint32_t address)
```

Reads the boolean from one of the output registers, which can also be accessed by a Field bus.

Note, uses its own memory space.

Parameters

address	Address of the register (0:127)
---------	---------------------------------

Returns

The boolean value held by the register (true, false)

Note

The lower range of the boolean output registers [0:63] is reserved for FieldBus/PLC interface usage. The upper range [64:127] cannot be accessed by FieldBus/PLC interfaces, since it is reserved for external RTDE clients.

Python interface prototype

```
getBoolOutput(self: pyaubo_sdk.RegisterControl, arg0: int) -> bool
```

Lua interface prototype

```
getBoolOutput(address: number) -> boolean
```

Lua example

```
BoolOutput_0 = getBoolOutput(0)
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"RegisterControl.getBoolOutput","params":[0,"id":1]}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":false}
```

8.5.2.8 getDouble()

```
double arcs::common_interface::RegisterControl::getDouble (
    const std::string & key,
    double default_value)
```

Get variable value

Parameters

key	
default_value	

Returns

Python interface prototype

```
getDouble(self: pyaubo_sdk.RegisterControl, arg0: str, arg1: float) -> float
```

Lua interface prototype

```
getDouble(key: string, default_value: number) -> number
```

Lua example

```
var_Double = getDouble("custom",0.0)
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"RegisterControl.getDouble","params":["custom",0.0],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":0.0}
```

8.5.2.9 getDoubleInput()

```
double arcs::common_interface::RegisterControl::getDoubleInput (
    uint32_t address)
```

Reads the double value from one of the input registers, which can also be accessed by a FieldBus.

Note, uses its own memory space.

Parameters

address	Address of the register (0:47)
---------	--------------------------------

Returns

The double value held by the register

Python interface prototype

```
getDoubleInput(self: pyaubo_sdk.RegisterControl, arg0: int) -> float
```

Lua interface prototype

```
getDoubleInput(address: number) -> number
```

Lua example

```
DoubleInput_0 = getDoubleInput(0)
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"RegisterControl.getDoubleInput","params":[0],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":0.0}
```

8.5.2.10 getDoubleOutput()

```
double arcs::common_interface::RegisterControl::getDoubleOutput (
    uint32_t address)
```

Reads the double value from one of the output registers.

Parameters

address	Address of the register
---------	-------------------------

Returns

The double value held by the register

Python interface prototype

```
getDoubleOutput(self: pyaubo_sdk.RegisterControl, arg0: int) -> float
```

Lua interface prototype

```
getDoubleOutput(address: number) -> number
```

Lua example

```
DoubleOutput_0 = getDoubleOutput(0)
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"RegisterControl.getDoubleOutput","params":[0],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":0.0}
```

8.5.2.11 getFloat()

```
float arcs::common_interface::RegisterControl::getFloat (
    const std::string & key,
    float default_value)
```

Get variable value

Parameters

key	
default_value	

Returns

Python interface prototype

```
getFloat(self: pyaubo_sdk.RegisterControl, arg0: str, arg1: float) -> float
```

Lua interface prototype

```
getFloat(key: string, default_value: number) -> number
```

Lua example

```
var_Float = getFloat("custom",0.0)
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"RegisterControl.getFloat","params":["custom",0.0],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":4.400000095367432}
```

8.5.2.12 getFloatInput()

```
float arcs::common_interface::RegisterControl::getFloatInput (
    uint32_t address)
```

Reads the float from one of the input registers, which can also be accessed by a Field bus.

Note, uses it's own memory space.

Parameters

address	Address of the register (0:47)
---------	--------------------------------

Returns

The value held by the register (float)

Note

The lower range of the float input registers [0:23] is reserved for FieldBus/PLC interface usage. The upper range [24:47] cannot be accessed by FieldBus/PLC interfaces, since it is reserved for external RTDE clients.

Python interface prototype

```
getFloatInput(self: pyaubo_sdk.RegisterControl, arg0: int) -> float
```

Lua interface prototype

```
getFloatInput(address: number) -> number
```

Lua example

```
FloatInput_0_0 = getFloatInput(0)
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"RegisterControl.getFloatInput","params":[0],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":0.0}
```

8.5.2.13 getFloatOutput()

```
float arcs::common_interface::RegisterControl::getFloatOutput (
    uint32_t address)
```

Reads the float from one of the output registers, which can also be accessed by a FieldBus.

Note, uses its own memory space.

Parameters

address	Address of the register (0:47)
---------	--------------------------------

Returns

The value held by the register (float)

Note

The lower range of the float output registers [0:23] is reserved for FieldBus/PLC interface usage. The upper range [24:47] cannot be accessed by FieldBus/PLC interfaces, since it is reserved for external RTDE clients.

Python interface prototype

```
getFloatOutput(self: pyaubo_sdk.RegisterControl, arg0: int) -> float
```

Lua interface prototype

```
getFloatOutput(address: number) -> number
```

Lua example

```
FloatOutput_0 = getFloatOutput(0)
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"RegisterControl.getFloatOutput","params":[0],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":3.3}
```

8.5.2.14 getInt16Register()

```
int16_t arcs::common_interface::RegisterControl::getInt16Register (
    uint32_t address)
```

Used for Modbus Slave

Parameters

address	
---------	--

Returns

Python interface prototype

```
getInt16Register(self: pyaubo_sdk.RegisterControl, arg0: int) -> int
```

Lua interface prototype

```
getInt16Register(address: number) -> number
```

Lua example

```
Int16Register_0 = getInt16Register(0)
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"RegisterControl.getInt16Register","params":[0],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":0}
```

8.5.2.15 getInt16RegisterBit()

```
bool arcs::common_interface::RegisterControl::getInt16RegisterBit (
    uint32_t address,
    uint8_t bit_offset)
```

Get the status of a specific bit in an Int16 register

Parameters

address	Int16 register address
bit_offset	Bit offset in the register, 0 ~ 15

Returns

Status of the specified bit, true for 1, false for 0 or invalid parameters

Python interface prototype

```
getInt16RegisterBit(self: pyaubo_sdk.RegisterControl, address: int, bit_offset: int) -> bool
```

Lua interface prototype

```
getInt16RegisterBit(address: number, bit_offset: number) -> boolean
```

JSON-RPC request example

```
{"jsonrpc": "2.0", "method": "RegisterControl.getInt16RegisterBit", "params": [1, 5], "id": 1}
```

JSON-RPC response example

```
{"id": 1, "jsonrpc": "2.0", "result": true}
```

8.5.2.16 getInt32()

```
int arcs::common_interface::RegisterControl::getInt32 (
    const std::string & key,
    int default_value)
```

Get variable value

Parameters

key	
default_value	

Returns

Python interface prototype

```
getInt32(self: pyaubo_sdk.RegisterControl, arg0: str, arg1: int) -> int
```

Lua interface prototype

```
getInt32(key: string, default_value: number) -> number
```

Lua example

```
Int32 = getInt32("custom",0)
```

JSON-RPC request example

```
{"jsonrpc": "2.0", "method": "RegisterControl.getInt32", "params": ["custom", 0], "id": 1}
```

JSON-RPC response example

```
{"id": 1, "jsonrpc": "2.0", "result": 6}
```

8.5.2.17 getInt32Input()

```
int arcs::common_interface::RegisterControl::getInt32Input (
    uint32_t address)
```

Reads the integer from one of the input registers, which can also be accessed by a FieldBus.

Note, uses it's own memory space.

Parameters

address	Address of the register (0:47)
---------	--------------------------------

Returns

The value held by the register [-2,147,483,648 : 2,147,483,647]

Note

The lower range of the integer input registers [0:23] is reserved for FieldBus/PLC interface usage. The upper range [24:47] cannot be accessed by FieldBus/PLC interfaces, since it is reserved for external RTDE clients.

Python interface prototype

```
getInt32Input(self: pyaubo_sdk.RegisterControl, arg0: int) -> int
```

Lua interface prototype

```
getInt32Input(address: number) -> number
```

Lua example

```
Int32Input_0 = getInt32Input(0)
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"RegisterControl.getInt32Input","params":[0],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":0}
```

8.5.2.18 getInt32Output()

```
int arcs::common_interface::RegisterControl::getInt32Output (
    uint32_t address)
```

Reads the integer from one of the output registers, which can also be accessed by a FieldBus.

Parameters

address	Address of the register (0:47)
---------	--------------------------------

Returns

The int value held by the register [-2,147,483,648 : 2,147,483,647]

Note

The lower range of the integer output registers [0:23] is reserved for FieldBus/PLC interface usage. The upper range [24:47] cannot be accessed by FieldBus/PLC interfaces, since it is reserved for external RTDE clients.

Python interface prototype

```
getInt32Output(self: pyaubo_sdk.RegisterControl, arg0: int) -> int
```

Lua interface prototype

```
getInt32Output(address: number) -> number
```

Lua example

```
Int32Output_0 = getInt32Output(0)
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"RegisterControl.getInt32Output","params":[0],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":0}
```

8.5.2.19 getModbusDeviceStatus()

```
int arcs::common_interface::RegisterControl::getModbusDeviceStatus (
    const std::string & device_name)
```

Get the connection status of the specified modbus device

Parameters

device_name	Device name in TCP format: "ip:port", e.g. "127.0.0.1:502" Device name in RTU format: "serial_port", e.g. "/dev/ttyUSB0"
-------------	---

Returns

0: Device is connected -1: Device does not exist -2: Device is disconnected

JSON-RPC request example

```
{ "jsonrpc": "2.0", "method": "RegisterControl.getModbusDeviceStatus", "params": [ "172.16.26.248:502", "id": 1 ] }
```

JSON-RPC response example

```
{ "id": 1, "jsonrpc": "2.0", "result": 0 }
```

8.5.2.20 getNamedVariableType()

```
std::string arcs::common_interface::RegisterControl::getNamedVariableType (
    const std::string & key)
```

Get the type of a named variable

Parameters

key	
-----	--

Returns

Lua interface prototype

```
getNamedVariableType(key: string) -> string
```

Lua example

```
NamedVariableType = getNamedVariableType("custom")
```

JSON-RPC request example

```
{ "jsonrpc": "2.0", "method": "RegisterControl.getNamedVariableType", "params": [ "custom", "id": 1 ] }
```

JSON-RPC response example

```
{ "id": 1, "jsonrpc": "2.0", "result": "NONE" }
```

8.5.2.21 getString()

```
std::string arcs::common_interface::RegisterControl::getString (
    const std::string & key,
    const std::string & default_value)
```

Get variable value

Parameters

key	
default_value	

Returns

Python interface prototype

```
getString(self: pyaubo_sdk.RegisterControl, arg0: str, arg1: str) -> str
```

Lua interface prototype

```
getString(key: string, default_value: string) -> string
```

Lua example

```
var_String = getString("custom", "")
```

JSON-RPC request example

```
{"jsonrpc": "2.0", "method": "RegisterControl.getString", "params": ["custom", ""], "id": 1}
```

JSON-RPC response example

```
{"id": 1, "jsonrpc": "2.0", "result": "test"}
```

8.5.2.22 getVecChar()

```
std::vector< char > arcs::common_interface::RegisterControl::getVecChar (
    const std::string & key,
    const std::vector< char > & default_value)
```

Get variable value

Parameters

key	
default_value	

Returns

Python interface prototype

```
getVecChar(self: pyaubo_sdk.RegisterControl, arg0: str, arg1: List[str]) -> List[str]
```

Lua interface prototype

```
getVecChar(key: string, default_value: table) -> table
```

Lua example

```
VecChar = getVecChar("custom",{})
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"RegisterControl.getVecChar","params":["custom",[]],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":[0,1,0]}
```

8.5.2.23 getVecDouble()

```
std::vector< double > arcs::common_interface::RegisterControl::getVecDouble (
    const std::string & key,
    const std::vector< double > & default_value)
```

Get variable value

Parameters

key	
default_value	

Returns

Python interface prototype

```
getVecDouble(self: pyaubo_sdk.RegisterControl, arg0: str, arg1: List[float]) -> List[float]
```

Lua interface prototype

```
getVecDouble(key: string, default_value: table) -> table
```

Lua example

```
VecDouble = getVecDouble("custom",{})
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"RegisterControl.getVecDouble","params":["custom",[]],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":[0.1,0.2,0.3]}
```

8.5.2.24 getVecFloat()

```
std::vector< float > arcs::common_interface::RegisterControl::getVecFloat (
    const std::string & key,
    const std::vector< float > & default_value)
```

Get variable value

Parameters

key	
default_value	

Returns

Python interface prototype

```
getVecFloat(self: pyaubo_sdk.RegisterControl, arg0: str, arg1: List[float]) -> List[float]
```

Lua interface prototype

```
getVecFloat(key: string, default_value: table) -> table
```

Lua example

```
VecFloat = getVecFloat("custom",{})
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"RegisterControl.getVecFloat","params":["custom",[]],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":[0.0,0.10000000149011612,3.299999952316284]}
```

8.5.2.25 getVecInt32()

```
std::vector< int32_t > arcs::common_interface::RegisterControl::getVecInt32 (
    const std::string & key,
    const std::vector< int32_t > & default_value)
```

Get variable value

Parameters

key	
default_value	

Returns

Python interface prototype

```
getVecInt32(self: pyaubo_sdk.RegisterControl, arg0: str, arg1: List[int]) -> List[int]
```

Lua interface prototype

```
getVecInt32(key: string, default_value: table) -> table
```

Lua example

```
VecInt32 = getVecInt32("custom",{})
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"RegisterControl.getVecInt32","params":["custom",[]],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":[1,2,3,4]}
```


8.5.2.26 getWatchDogAction()

```
int arcs::common_interface::RegisterControl::getWatchDogAction (  
    const std::string & key)
```

Get the watchdog action

Parameters

key	
-----	--

Returns

8.5.2.27 getWatchDogTimeout()

```
int arcs::common_interface::RegisterControl::getWatchDogTimeout (  
    const std::string & key)
```

Get the watchdog timeout value

Parameters

key	
-----	--

Returns

8.5.2.28 hasNamedVariable()

```
bool arcs::common_interface::RegisterControl::hasNamedVariable (  
    const std::string & key)
```

Whether the named variable exists

Parameters

key	Variable name
-----	---------------

Returns

Lua interface prototype

```
hasNamedVariable(key: string) -> boolean
```

Lua example

```
NamedVariable = hasNamedVariable("custom")
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"RegisterControl.hasNamedVariable","params":["custom"],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":false}
```

8.5.2.29 modbusAddSignal()

```
int arcs::common_interface::RegisterControl::modbusAddSignal (
    const std::string & device_info,
    int slave_number,
    int signal_address,
    int signal_type,
    const std::string & signal_name,
    bool sequential_mode)
```

Adds a new modbus signal for the controller to supervise.

Expects no response.

Parameters

device_info	is rtu format. eg, "serial_port,baud,parity,data_bit,stop_bit" (1)The serial_port argument specifies the name of the serial port eg. On Linux, "/dev/ttyS0" or "/dev/ttyUSB0". On Windows, "\\.\COM10". (2)The baud argument specifies the baud rate of the communication, eg. 9600, 19200, 57600, 115200, etc. (3)parity:N for none,E for even,O for odd. (4)data_bit:The data_bits argument specifies the number of bits of data, the allowed values are 5, 6, 7 and 8. (5)stop_bit:The stop_bits argument specifies the bits of stop, the allowed values are 1 and 2.
-------------	--

device_info is tcp format.eg, "ip address,port" (1)The ip address parameter specifies the ip address of the server (2)The port parameter specifies the port number that the server is listening on.

Parameters

slave_number	An integer normally not used and set to 255, but is a free choice between 0 and 255.
signal_address	An integer specifying the address of the either the coil or the register that this new signal should reflect. Consult the configuration of the modbus unit for this information.
signal_type	An integer specifying the type of signal to add. 0 = digital input, 1 = digital output, 2 = register input and 3 = register output.
signal_name	A string uniquely identifying the signal. If a string is supplied which is equal to an already added signal, the new signal will replace the old one. The length of the string cannot exceed 20 characters.
sequential_mode	Setting to True forces the modbus client to wait for a response before sending the next request. This mode is required by some fieldbus units (Optional).

Returns

Python interface prototype

```
modbusAddSignal(self: pyaubo_sdk.RegisterControl, arg0: str, arg1: int, arg2: int, arg3: int,
arg4: str, arg5: bool) -> int
```

Lua interface prototype

```
modbusAddSignal(device_info: string, slave_number: number, signal_address: number, signal_type: number, signal_name: string, sequential_mode: boolean) -> nil
```

Lua example

```
modbusAddSignal("/dev/ttyRobotTool,115200,N,8,1", 1, signal_address: number, 264, "Modbus_0", false)
```

JSON-RPC request example

```
{"jsonrpc": "2.0", "method": "RegisterControl.modbusAddSignal", "params": ["/dev/ttyRobotTool,115200,N,8,1", 1, 264, 3, "Modbus_0", false], "id": 1}
```

JSON-RPC response example

```
{"id": 1, "jsonrpc": "2.0", "result": 0}
```

8.5.2.30 modbusDeleteAllSignals()

```
int arcs::common_interface::RegisterControl::modbusDeleteAllSignals ()
```

Delete all modbus signals

Returns

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"RegisterControl.modbusDeleteAllSignals","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":0}
```

8.5.2.31 modbusDeleteSignal()

```
int arcs::common_interface::RegisterControl::modbusDeleteSignal (
    const std::string & signal_name)
```

Deletes the signal identified by the supplied signal name.

Parameters

signal_name	A string equal to the name of the signal that should be deleted.
-------------	--

Returns

Python interface prototype

```
modbusDeleteSignal(self: pyaubo_sdk.RegisterControl, arg0: str) -> int
```

Lua interface prototype

```
modbusDeleteSignal(signal_name: string) -> nil
```

Lua example

```
modbusDeleteSignal("Modbus_1")
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"RegisterControl.modbusDeleteSignal","params":["Modbus_1"],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":0}
```

8.5.2.32 modbusGetSignalError()

```
int arcs::common_interface::RegisterControl::modbusGetSignalError (
    const std::string & signal_name)
```

Get the error status of the specified modbus signal

Parameters

signal_name	
-------------	--

Returns

Returns error code [ModbusErrorNum](#)

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"RegisterControl.modbusGetSignalError","params":["Modbus_0"],
,"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":6}
```

8.5.2.33 modbusGetSignalErrors()

```
std::vector< int > arcs::common_interface::RegisterControl::modbusGetSignalErrors ()
```

Get the error status of all signal requests (0: no error, others: error) as a collection

Returns

[ModbusErrorNum](#)

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"RegisterControl.modbusGetSignalErrors","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":[6,6,6,6]}
```

8.5.2.34 modbusGetSignalIndex()

```
Generated by Doxygen
int arcs::common_interface::RegisterControl::modbusGetSignalIndex (
    const std::string & signal_name)
```

Get the index of the specified modbus signal, starting from 0.

Parameters

signal_name	
-------------	--

Returns

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"RegisterControl.modbusGetSignalIndex","params":["Modbus_0"],
"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":0}
```

8.5.2.35 modbusGetSignalNames()

```
std::vector< std::string > arcs::common_interface::RegisterControl::modbusGetSignalNames ()
```

Get the collection of all signal names

Returns

Collection of all signal names

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"RegisterControl.modbusGetSignalNames","params":[], "id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":["Modbus_0"]}
```

8.5.2.36 modbusGetSignalStatus()

```
int arcs::common_interface::RegisterControl::modbusGetSignalStatus (
    const std::string & signal_name)
```

Reads the current value of a specific signal.

Parameters

signal_name	A string equal to the name of the signal for which the value should be gotten.
-------------	--

Returns

An integer or a boolean. For digital signals: 1 or 0. For register signals: The register value expressed as an integer. If the value is -1, it means the signal does not exist.

Python interface prototype

```
modbusGetSignalStatus(self: pyaubo_sdk.RegisterControl, arg0: str) -> int
```

Lua interface prototype

```
modbusGetSignalStatus(signal_name: string) -> number
```

Lua example

```
var0 = modbusGetSignalStatus("Modbus_0")
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"RegisterControl.modbusGetSignalStatus","params":["Modbus_0"],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":1}
```

8.5.2.37 modbusGetSignalTypes()

```
std::vector< int > arcs::common_interface::RegisterControl::modbusGetSignalTypes ()
```

Get the collection of all signal types

Returns

Collection of all signal types

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"RegisterControl.modbusGetSignalTypes","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":[1,0,2,3]}
```

8.5.2.38 modbusGetSignalValues()

```
std::vector< int > arcs::common_interface::RegisterControl::modbusGetSignalValues ()
```

Get the collection of all signal values

Returns

Collection of all signal values

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"RegisterControl.modbusGetSignalValues","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":[1,1,88,33]}
```

8.5.2.39 modbusSendCustomCommand()

```
int arcs::common_interface::RegisterControl::modbusSendCustomCommand (
    const std::string & device_info,
    int slave_number,
    int function_code,
    const std::vector< uint8_t > & data)
```

Sends a command specified by the user to the modbus unit located on the specified IP address.

Cannot be used to request data, since the response will not be received. The user is responsible for supplying data which is meaningful to the supplied function code. The builtin function takes care of constructing the modbus frame, so the user should not be concerned with the length of the command.

Parameters

device_info	is rtu format. eg,"serial_port,baud,parity,data_bit,stop_bit" (1)The serial_port argument specifies the name of the serial port eg. On Linux ,"/dev/ttyS0" or "/dev/ttyUSB0". On Windows, "\\.\COM10". (2)The baud argument specifies the baud rate of the communication, eg. 9600, 19200, 57600, 115200, etc. (3)parity:N for none,E for even,O for odd. (4)data_bit:The data_bits argument specifies the number of bits of data, the allowed values are 5, 6, 7 and 8. (5)stop_bit:The stop_bits argument specifies the bits of stop, the allowed values are 1 and 2.
-------------	---

device_info is tcp format.eg,"ip address,port" (1)The ip address parameter specifies the ip address of the server (2)The port parameter specifies the port number that the server is listening on.

Parameters

slave_number	An integer specifying the slave number to use for the custom command.
function_code	An integer specifying the function code for the custom command.

Modbus function codes MODBUS_FC_READ_COILS 0x01 MODBUS_FC_READ_DISCRETE_↵
INPUTS 0x02 MODBUS_FC_READ_HOLDING_REGISTERS 0x03 MODBUS_FC_READ_↵
INPUT_REGISTERS 0x04 MODBUS_FC_WRITE_SINGLE_COIL 0x05 MODBUS_FC_WRITE_↵
_SINGLE_REGISTER 0x06 MODBUS_FC_READ_EXCEPTION_STATUS 0x07 MODBUS_↵
FC_WRITE_MULTIPLE_COILS 0x0F MODBUS_FC_WRITE_MULTIPLE_REGISTERS 0x10
MODBUS_FC_REPORT_SLAVE_ID 0x11 MODBUS_FC_MASK_WRITE_REGISTER 0x16
MODBUS_FC_WRITE_AND_READ_REGISTERS 0x17

Parameters

data	An array of integers in which each entry must be a valid byte (0-255) value.
------	--

Returns

Python interface prototype

```
modbusSendCustomCommand(self: pyaubo_sdk.RegisterControl, arg0: str, arg1: int, arg2: int,
arg3: List[int]) -> int
```

Lua interface prototype

```
modbusSendCustomCommand(device_info: string, slave_number: number, function_code: num-
ber, data: table) -> nil
```

Lua example

```
modbusSendCustomCommand("/dev/ttyRobotTool,115200,N,8,1", 1, 10, {1,2,0,2,4,0,0,0}) -> nil
```

JSON-RPC request example

```
{ "jsonrpc": "2.0", "method": "RegisterControl.modbusSendCustomCommand", "params": ["/dev/↵
ttyRobotTool,115200,N,8,1", 1, 10, [1,2,0,2,4,0,0,0]], "id": 1 }
```

JSON-RPC response example

```
{ "id": 1, "jsonrpc": "2.0", "result": 0 }
```

Parameters

robot_name	A string identifying a robot name that connected robot
signal_name	A string identifying a digital input signal that was previously added.
action	The type of action. The action can either be "default" or "freedrive". (string)

Returns

Python interface prototype

```
modbusSetDigitalInputAction(self: pyaubo_sdk.RegisterControl, arg0: str, arg1: str, arg2: int)
```

Lua interface prototype

```
modbusSetDigitalInputAction(robot_name: string, signal_name: string, action: number) -> nil
```

Lua example

```
modbusSetDigitalInputAction("rob1", "Modbus_0", "Handguide")
```

JSON-RPC request example

```
{ "jsonrpc": "2.0", "method": "RegisterControl.modbusSetDigitalInputAction", "params": ["rob1", "Modbus_0", "Handguide"], "id": 1 }
```

JSON-RPC response example

```
{ "id": 1, "jsonrpc": "2.0", "result": 0 }
```

8.5.2.41 modbusSetOutputRunstate()

```
int arcs::common_interface::RegisterControl::modbusSetOutputRunstate (
    const std::string & robot_name,
    const std::string & signal_name,
    StandardOutputRunState runstate)
```

Set Modbus signal output action

Parameters

robot_name	
signal_name	
runstate	

Returns

JSON-RPC request example

```
{ "jsonrpc": "2.0", "method": "RegisterControl.modbusSetOutputRunstate", "params": [ "rob1", "↵
Modbus_0", "None" ], "id": 1 }
```

JSON-RPC response example

```
{ "id": 1, "jsonrpc": "2.0", "result": 0 }
```

8.5.2.42 modbusSetOutputSignal()

```
int arcs::common_interface::RegisterControl::modbusSetOutputSignal (
    const std::string & signal_name,
    uint16_t value)
```

Sets the output register signal identified by the given name to the given value.

Parameters

signal_name	A string identifying an output register signal that in advance has been added.
value	An integer which must be a valid word (0-65535)

Returns

Python interface prototype

```
modbusSetOutputSignal(self: pyaubo_sdk.RegisterControl, arg0: str, arg1: int) -> int
```

Lua interface prototype

```
modbusSetOutputSignal(signal_name: string, value: number) -> nil
```

Lua example

```
modbusSetOutputSignal("Modbus_0",0)
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"RegisterControl.modbusSetOutputSignal","params":["Modbus_0",0],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":0}
```

8.5.2.43 modbusSetOutputSignal1()

```
int arcs::common_interface::RegisterControl::modbusSetOutputSignal1 (
    const std::string & signal_name,
    const std::vector< uint16_t > & values)
```

Sets multiple consecutive output register signals starting from the given name to the given values.

Parameters

signal_name	A string identifying the starting output register signal that in advance has been added.
values	An array of integers where each element must be a valid word (0-65535), corresponding to multiple consecutive signals

Returns

Python interface prototype

```
modbusSetOutputSignal1(self: pyaubo_sdk.RegisterControl, arg0: str, arg1: list[int]) -> int
```

Lua interface prototype

```
modbusSetOutputSignal1(signal_name: string, value: table) -> nil
```

Lua example

```
modbusSetOutputSignal1("Modbus_0", {0, 1, 2})
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"RegisterControl.modbusSetOutputSignal1","params":["Modbus_0",0,1,2],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":0}
```

8.5.2.44 modbusSetOutputSignalPulse()

```
int arcs::common_interface::RegisterControl::modbusSetOutputSignalPulse (
    const std::string & signal_name,
    uint16_t value,
    double duration)
```

Set modbus signal output pulse (only supports coil output type)

Parameters

signal_name	A string identifying an output register signal that has been added in advance.
value	An integer which must be a valid word (0-65535)
duration	Duration of the signal, in seconds

Returns

Python interface prototype

```
modbusSetOutputSignalPulse(self: pyaubo_sdk.RegisterControl, arg0: str, arg1: int, arg2: double)
-> int
```

Lua interface prototype

```
modbusSetOutputSignalPulse(signal_name: string, value: number, duration: number) -> nil
```

Lua example

```
modbusSetOutputSignalPulse("Modbus_0",1,0.5)
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"RegisterControl.modbusSetOutputSignalPulse","params":["Modbus_0",1,0.5],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":0}
```

8.5.2.45 modbusSetSignalUpdateFrequency()

```
int arcs::common_interface::RegisterControl::modbusSetSignalUpdateFrequency (
    const std::string & signal_name,
    int update_frequency)
```

Sets the frequency with which the robot will send requests to the Modbus controller to either read or write the signal value.

Parameters

signal_name	A string identifying an output digital signal that in advance has been added.
update_frequency	An integer in the range 0-125 specifying the update frequency in Hz.

Returns

Python interface prototype

```
modbusSetSignalUpdateFrequency(self: pyaubo_sdk.RegisterControl, arg0: str, arg1: int) -> int
```

Lua interface prototype

```
modbusSetSignalUpdateFrequency(signal_name: string, update_frequency: number) -> nil
```

Lua example

```
modbusSetSignalUpdateFrequency("Modbus_0",1)
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"RegisterControl.modbusSetSignalUpdateFrequency","params":["Modbus_0",1],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":0}
```

8.5.2.46 setBool()

```
int arcs::common_interface::RegisterControl::setBool (
    const std::string & key,
    bool value)
```

Set or update the variable value

Parameters

key	
value	

Returns

Returns 0 on success, otherwise error code

Python interface prototype

```
setBool(self: pyaubo_sdk.RegisterControl, arg0: str, arg1: bool) -> int
```

Lua interface prototype

```
setBool(key: string, value: boolean) -> nil
```

Lua example

```
setBool("custom",true)
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"RegisterControl.setBool","params":["custom",true],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":0}
```

8.5.2.47 setBoolInput()

```
int arcs::common_interface::RegisterControl::setBoolInput (  
    uint32_t address,  
    bool value)
```

Parameters

address	Address of the register (0:127)
value	Boolean value to set (true or false)

Returns

Returns 0 on success, or an error code

Note

Only used when implementing RTDE/Modbus Slave/PLC server

Python interface prototype

```
setBoolInput(self: pyaubo_sdk.RegisterControl, arg0: int, arg1: bool) -> int
```

Lua interface prototype

```
setBoolInput(address: number, value: boolean) -> nil
```

Lua example

```
setBoolInput(0)
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"RegisterControl.setBoolInput","params":[0,true],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":0}
```

8.5.2.48 setBoolOutput()

```
int arcs::common_interface::RegisterControl::setBoolOutput (
    uint32_t address,
    bool value)
```

Parameters

address	Address of the register (0:127)
value	Boolean value to set (true or false)

Returns

Returns 0 on success, or an error code

Python interface prototype

```
setBoolOutput(self: pyaubo_sdk.RegisterControl, arg0: int, arg1: bool) -> int
```

Lua interface prototype

```
setBoolOutput(address: number, value: boolean) -> nil
```

Lua example

```
setBoolOutput(0)
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"RegisterControl.setBoolOutput","params":[0,false],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":0}
```


8.5.2.49 setDouble()

```
int arcs::common_interface::RegisterControl::setDouble (
    const std::string & key,
    double value)
```

Set or update the variable value

Parameters

key	
value	

Returns

Python interface prototype

```
setDouble(self: pyaubo_sdk.RegisterControl, arg0: str, arg1: float) -> int
```

Lua interface prototype

```
setDouble(key: string, value: number) -> nil
```

Lua example

```
setDouble("custom",6.6)
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"RegisterControl.setDouble","params":["custom",6.6],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":0}
```

8.5.2.50 setDoubleInput()

```
int arcs::common_interface::RegisterControl::setDoubleInput (
    uint32_t address,
    double value)
```

Parameters

address	
value	

Returns

Note

Only used when implementing RTDE/Modbus Slave/PLC server

Python interface prototype

```
setDoubleInput(self: pyaubo_sdk.RegisterControl, arg0: int, arg1: float) -> int
```

Lua interface prototype

```
setDoubleInput(address: number, value: number) -> nil
```

Lua example

```
setDoubleInput(0, 3.3)
```

JSON-RPC request example

```
{"jsonrpc": "2.0", "method": "RegisterControl.setDoubleInput", "params": [0, 6.6], "id": 1}
```

JSON-RPC response example

```
{"id": 1, "jsonrpc": "2.0", "result": 0}
```

8.5.2.51 setDoubleOutput()

```
int arcs::common_interface::RegisterControl::setDoubleOutput (
    uint32_t address,
    double value)
```

Parameters

address	Address of the register
---------	-------------------------

value	Double value to set
-------	---------------------

Returns

Returns 0 on success, or an error code

Python interface prototype

```
setDoubleOutput(self: pyaubo_sdk.RegisterControl, arg0: int, arg1: float) -> int
```

Lua interface prototype

```
setDoubleOutput(address: number, value: number) -> nil
```

Lua example

```
setDoubleOutput(0,4.4)
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"RegisterControl.setDoubleOutput","params":[0,4.4],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":0}
```

8.5.2.52 setFloat()

```
int arcs::common_interface::RegisterControl::setFloat (
    const std::string & key,
    float value)
```

Set or update the variable value

Parameters

key	
value	

Returns

Python interface prototype

```
setFloat(self: pyaubo_sdk.RegisterControl, arg0: str, arg1: float) -> int
```

Lua interface prototype

```
setFloat(key: string, value: number) -> nil
```

Lua example

```
setFloat("custom",4.4)
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"RegisterControl.setFloat","params":["custom",4.4],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":0}
```

8.5.2.53 setFloatInput()

```
int arcs::common_interface::RegisterControl::setFloatInput (  
    uint32_t address,  
    float value)
```

Parameters

address	Address of the register (0:47)
value	Float value to set

Returns

Returns 0 on success, or an error code

Note

Only used when implementing RTDE/Modbus Slave/PLC server

Python interface prototype

```
setFloatInput(self: pyaubo_sdk.RegisterControl, arg0: int, arg1: float) -> int
```

Lua interface prototype

```
setFloatInput(address: number, value: number) -> nil
```

Lua example

```
setFloatInput(0, 3.3)
```

JSON-RPC request example

```
{"jsonrpc": "2.0", "method": "RegisterControl.setFloatInput", "params": [0, 3.3], "id": 1}
```

JSON-RPC response example

```
{"id": 1, "jsonrpc": "2.0", "result": 0}
```

8.5.2.54 setFloatOutput()

```
int arcs::common_interface::RegisterControl::setFloatOutput (
    uint32_t address,
    float value)
```

Parameters

address	Address of the register (0:47)
value	Float value to set

Returns

Returns 0 on success, or an error code

Python interface prototype

```
setFloatOutput(self: pyaubo_sdk.RegisterControl, arg0: int, arg1: float) -> int
```

Lua interface prototype

```
setFloatOutput(address: number, value: number) -> nil
```

Lua example

```
setFloatOutput(0, 5.5)
```

JSON-RPC request example

```
{"jsonrpc": "2.0", "method": "RegisterControl.setFloatOutput", "params": [0, 5.5], "id": 1}
```

JSON-RPC response example

```
{"id": 1, "jsonrpc": "2.0", "result": 0}
```

8.5.2.55 setInt16Register()

```
int arcs::common_interface::RegisterControl::setInt16Register (
    uint32_t address,
    int16_t value)
```

Parameters

address	Register address
value	Value to set

Returns

Returns 0 on success, otherwise error code

Python interface prototype

```
setInt16Register(self: pyaubo_sdk.RegisterControl, arg0: int, arg1: int) -> int
```

Lua interface prototype

```
setInt16Register(address: number, value: number) -> nil
```

Lua example

```
setInt16Register(0,4.4)
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"RegisterControl.setInt16Register","params":[0,0],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":0}
```

8.5.2.56 setInt16RegisterBit()

```
int arcs::common_interface::RegisterControl::setInt16RegisterBit (
    uint32_t address,
    uint8_t bit_offset,
    bool value)
```

Set the status of a specific bit in an Int16 register

Parameters

address	Int16 register address
bit_offset	Bit offset in the register, 0 ~ 15
value	Value to set, true to set to 1, false to clear to 0

Returns

Returns 0 on success, error code on failure

Python interface prototype

```
setInt16RegisterBit(self: pyaubo_sdk.RegisterControl, address: int, bit_offset: int, value: bool)
-> int
```

Lua interface prototype

```
setInt16RegisterBit(address: number, bit_offset: number, value: boolean) -> number
```

JSON-RPC request example

```
{"jsonrpc": "2.0", "method": "RegisterControl.setInt16RegisterBit", "params": [1, 5, true], "id": 1}
```

JSON-RPC response example

```
{"id": 1, "jsonrpc": "2.0", "result": 0}
```

8.5.2.57 setInt32()

```
int arcs::common_interface::RegisterControl::setInt32 (
    const std::string & key,
    int value)
```

Set or update the variable value

Parameters

key	
value	

Returns

Python interface prototype

```
setInt32(self: pyaubo_sdk.RegisterControl, arg0: str, arg1: int) -> int
```

Lua interface prototype

```
setInt32(key: string, value: number) -> nil
```

Lua example

```
setInt32("custom",6)
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"RegisterControl.setInt32","params":["custom",6],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":0}
```

8.5.2.58 setInt32Input()

```
int arcs::common_interface::RegisterControl::setInt32Input (  
    uint32_t address,  
    int value)
```

Parameters

address	Address of the register (0:47)
value	Integer value to set

Returns

Returns 0 on success, or an error code

Note

Only used when implementing RTDE/Modbus Slave/PLC server

Python interface prototype

```
setInt32Input(self: pyaubo_sdk.RegisterControl, arg0: int, arg1: int) -> int
```

Lua interface prototype

```
setInt32Input(address: number, value: number) -> nil
```

Lua example

```
setInt32Input(0)
```

JSON-RPC request example

```
{"jsonrpc": "2.0", "method": "RegisterControl.setInt32Input", "params": [0, 33], "id": 1}
```

JSON-RPC response example

```
{"id": 1, "jsonrpc": "2.0", "result": 0}
```

8.5.2.59 setInt32Output()

```
int arcs::common_interface::RegisterControl::setInt32Output (
    uint32_t address,
    int value)
```

Parameters

address	Address of the register (0:47)
value	Integer value to set

Returns

Returns 0 on success, or an error code

Python interface prototype

```
setInt32Output(self: pyaubo_sdk.RegisterControl, arg0: int, arg1: int) -> int
```

Lua interface prototype

```
setInt32Output(address: number, value: number) -> nil
```

Lua example

```
setInt32Output(0, 100)
```

JSON-RPC request example

```
{"jsonrpc": "2.0", "method": "RegisterControl.setInt32Output", "params": [0, 100], "id": 1}
```

JSON-RPC response example

```
{"id": 1, "jsonrpc": "2.0", "result": 0}
```

8.5.2.60 setString()

```
int arcs::common_interface::RegisterControl::setString (
    const std::string & key,
    const std::string & value)
```

Set or update the variable value

Parameters

key	
value	

Returns

Returns 0 on success, otherwise error code

Python interface prototype

```
setString(self: pyaubo_sdk.RegisterControl, arg0: str, arg1: str) -> int
```

Lua interface prototype

```
setString(key: string, value: string) -> nil
```

Lua example

```
setString("custom", "test")
```

JSON-RPC request example

```
{"jsonrpc": "2.0", "method": "RegisterControl.setString", "params": ["custom", "test"], "id": 1}
```

JSON-RPC response example

```
{"id": 1, "jsonrpc": "2.0", "result": 0}
```

8.5.2.61 setVecChar()

```
int arcs::common_interface::RegisterControl::setVecChar (
    const std::string & key,
    const std::vector< char > & value)
```

Set or update the variable value

Parameters

key	
value	

Returns

Returns 0 on success, otherwise error code

Python interface prototype

```
setVecChar(self: pyaubo_sdk.RegisterControl, arg0: str, arg1: List[str]) -> int
```

Lua interface prototype

```
setVecChar(key: string, value: table) -> nil
```

Lua example

```
setVecChar("custom",{0,1,0})
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"RegisterControl.setVecChar","params":["custom",[0,1,0]],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":0}
```

8.5.2.62 setVecDouble()

```
int arcs::common_interface::RegisterControl::setVecDouble (
    const std::string & key,
    const std::vector< double > & value)
```

Set or update the variable value

Parameters

key	
value	

Returns

Python interface prototype

```
setVecDouble(self: pyaubo_sdk.RegisterControl, arg0: str, arg1: List[float]) -> int
```

Lua interface prototype

```
setVecDouble(key: string, value: table) -> nil
```

Lua example

```
setVecDouble("custom",{0.1,0.2,0.3})
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"RegisterControl.setVecDouble","params":["custom",[0.1,0.2,0.3]],  
"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":0}
```

8.5.2.63 setVecFloat()

```
int arcs::common_interface::RegisterControl::setVecFloat (  
    const std::string & key,  
    const std::vector< float > & value)
```

Set or update the variable value

Parameters

key	
value	

Returns

Python interface prototype

```
setVecFloat(self: pyaubo_sdk.RegisterControl, arg0: str, arg1: List[float]) -> int
```

Lua interface prototype

```
setVecFloat(key: string, value: table) -> nil
```

Lua example

```
setVecFloat("custom", {0.0,0.1,3.3})
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"RegisterControl.setVecFloat","params":["custom",[0.0,0.1,3.3]],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":0}
```

8.5.2.64 setVecInt32()

```
int arcs::common_interface::RegisterControl::setVecInt32 (
    const std::string & key,
    const std::vector< int32_t > & value)
```

Set or update the variable value

Parameters

key	
value	

Returns

Python interface prototype

```
setVecInt32(self: pyaubo_sdk.RegisterControl, arg0: str, arg1: List[int]) -> int
```

Lua interface prototype

```
setVecInt32(key: string, value: table) -> nil
```

Lua example

```
setVecInt32("custom",{1,2,3,4})
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"RegisterControl.setVecInt32","params":["custom",[1,2,3,4]],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":0}
```

8.5.2.65 setWatchDog()

```
int arcs::common_interface::RegisterControl::setWatchDog (
    const std::string & key,
    double timeout,
    int action)
```

Set the watchdog

After the watchdog is triggered, the controller will perform the corresponding action and automatically delete the watchdog.

Parameters

key	
timeout	Timeout in seconds (s), minimum timeout is 0.1s
action	NONE (0): No action PAUSE(1): Pause runtime STOP (2): Stop runtime/stop robot motion PROTECTIVE_STOP (3): Trigger protective stop

Returns

8.5.2.66 variableUpdated()

```
bool arcs::common_interface::RegisterControl::variableUpdated (
    const std::string & key,
    uint64_t since)
```

Whether the named variable has been updated

Parameters

key	
since	

Returns

Python interface prototype

```
variableUpdated(self: pyaubo_sdk.RegisterControl, arg0: str, arg1: int) -> bool
```

Lua interface prototype

```
variableUpdated(key: string, since: number) -> boolean
```

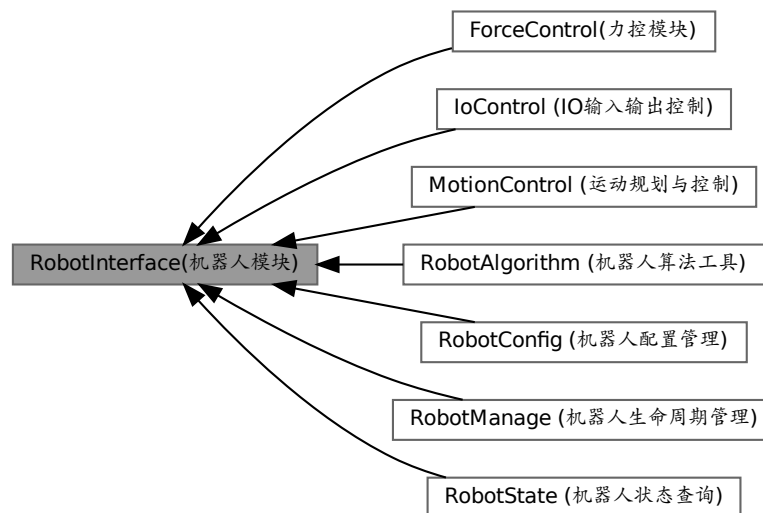
Lua example

```
Variable_Updated = variableUpdated("custom" , 0)
```

8.6 RobotInterface()

[RobotInterface](#).

Collaboration diagram for RobotInterface():



Topics

- [ForceControl\(\)](#)
Abstract class for force control interface
- [IoControl \(IO \)](#)
The [IoControl](#) class provides a series of interfaces for configuring and reading the robot's standard digital and analog IO, as well as setting output states.
- [MotionControl \(\)](#)
[MotionControl](#).
- [RobotAlgorithm \(\)](#)
Interfaces related to robot algorithms
- [RobotConfig \(\)](#)
[RobotConfig](#).
- [RobotManage \(\)](#)
[RobotManage](#).
- [RobotState \(\)](#)
[RobotState](#).

Functions

- [RobotConfigPtr arcs::common_interface::RobotInterface::getRobotConfig \(\)](#)
[RobotConfig \(\)](#) Get [RobotConfig](#) interface
- [MotionControlPtr arcs::common_interface::RobotInterface::getMotionControl \(\)](#)
[MotionControl \(\)](#) Get motion planning interface

- [ForceControlPtr arcs::common_interface::RobotInterface::getForceControl \(\)](#)
[ForceControl\(\)](#) Get force control interface
- [IoControlPtr arcs::common_interface::RobotInterface::getIoControl \(\)](#)
[IoControl \(IO \)](#) Get IO control interface
- [SyncMovePtr arcs::common_interface::RobotInterface::getSyncMove \(\)](#)
[SyncMove \(\)](#) Get synchronized motion interface
- [RobotAlgorithmPtr arcs::common_interface::RobotInterface::getRobotAlgorithm \(\)](#)
[RobotAlgorithm \(\)](#) Get robot utility algorithm interface
- [RobotManagePtr arcs::common_interface::RobotInterface::getRobotManage \(\)](#)
[RobotManage \(\)](#) Get robot management interface (power on, start, stop, etc.)
- [RobotStatePtr arcs::common_interface::RobotInterface::getRobotState \(\)](#)
[RobotState \(\)](#) Get robot state interface
- [TracePtr arcs::common_interface::RobotInterface::getTrace \(\)](#)
[Trace \(\)](#) Get alarm information interface

8.6.1 Detailed Description

[RobotInterface](#).

8.6.2 Function Documentation

8.6.2.1 getForceControl()

[ForceControlPtr arcs::common_interface::RobotInterface::getForceControl \(\)](#)

[ForceControl\(\)](#) Get force control interface

Returns

Pointer to [ForceControl](#) object

Python function prototype

```
getForceControl(self: pyaubo_sdk.RobotInterface) -> arcs::common\_interface::ForceControl
```

C++ example

```
auto rpc_cli = std::make_shared<RpcClient>();
auto robot_name = rpc_cli->getRobotNames().front();
ForceControlPtr ptr =
rpc_cli->getRobotInterface(robot_name)->getForceControl();
```


8.6.2.2 getIoControl()

`IoControlPtr` arcs::common_interface::RobotInterface::getIoControl ()

`IoControl` (`IO`) Get IO control interface

Returns

Pointer to `IoControl` object

Python function prototype

`getIoControl(self: pyaubo_sdk.RobotInterface) -> arcs::common_interface::IoControl`

C++ example

```
auto rpc_cli = std::make_shared<RpcClient>();
auto robot_name = rpc_cli->getRobotNames().front();
IoControlPtr ptr =
rpc_cli->getRobotInterface(robot_name)->getIoControl();
```

8.6.2.3 getMotionControl()

`MotionControlPtr` arcs::common_interface::RobotInterface::getMotionControl ()

`MotionControl` () Get motion planning interface

Returns

Pointer to `MotionControl` object

Python function prototype

`getMotionControl(self: pyaubo_sdk.RobotInterface) -> arcs::common_interface::MotionControl`

C++ example

```
auto rpc_cli = std::make_shared<RpcClient>();
auto robot_name = rpc_cli->getRobotNames().front();
MotionControlPtr ptr =
rpc_cli->getRobotInterface(robot_name)->getMotionControl();
```

8.6.2.4 getRobotAlgorithm()

`RobotAlgorithmPtr` arcs::common_interface::RobotInterface::getRobotAlgorithm ()

`RobotAlgorithm` () Get robot utility algorithm interface

Returns

Pointer to `RobotAlgorithm` object

Python function prototype

`getRobotAlgorithm(self: pyaubo_sdk.RobotInterface) -> arcs::common_interface::RobotAlgorithm`

C++ example

```
auto rpc_cli = std::make_shared<RpcClient>();
auto robot_name = rpc_cli->getRobotNames().front();
RobotAlgorithmPtr ptr =
rpc_cli->getRobotInterface(robot_name)->getRobotAlgorithm();
```

8.6.2.5 getRobotConfig()

[RobotConfigPtr](#) arcs::common_interface::RobotInterface::getRobotConfig ()

[RobotConfig](#) () Get [RobotConfig](#) interface

Returns

Pointer to [RobotConfig](#) object

Python function prototype

getRobotConfig(self: pyaubo_sdk.RobotInterface) -> [arcs::common_interface::RobotConfig](#)

C++ example

```
auto rpc_cli = std::make_shared<RpcClient>();
auto robot_name = rpc_cli->getRobotNames().front();
RobotConfigPtr ptr =
rpc_cli->getRobotInterface(robot_name)->getRobotConfig();
```

8.6.2.6 getRobotManage()

[RobotManagePtr](#) arcs::common_interface::RobotInterface::getRobotManage ()

[RobotManage](#) () Get robot management interface (power on, start, stop, etc.)

Returns

Pointer to [RobotManage](#) object

Python function prototype

getRobotManage(self: pyaubo_sdk.RobotInterface) -> [arcs::common_interface::RobotManage](#)

C++ example

```
auto rpc_cli = std::make_shared<RpcClient>();
auto robot_name = rpc_cli->getRobotNames().front();
RobotManagePtr ptr =
rpc_cli->getRobotInterface(robot_name)->getRobotManage();
```

8.6.2.7 getRobotState()

[RobotStatePtr](#) arcs::common_interface::RobotInterface::getRobotState ()

[RobotState](#) () Get robot state interface

Returns

Pointer to [RobotState](#) object

Python function prototype

getRobotState(self: pyaubo_sdk.RobotInterface) -> [arcs::common_interface::RobotState](#)

C++ example

```
auto rpc_cli = std::make_shared<RpcClient>();
auto robot_name = rpc_cli->getRobotNames().front();
RobotStatePtr ptr =
rpc_cli->getRobotInterface(robot_name)->getRobotState();
```

8.6.2.8 getSyncMove()

`SyncMovePtr` arcs::common_interface::RobotInterface::getSyncMove ()

`SyncMove` () Get synchronized motion interface

Returns

Pointer to `SyncMove` object

Python function prototype

`getSyncMove(self: pyaubo_sdk.RobotInterface) -> arcs::common_interface::SyncMove`

C++ example

```
auto rpc_cli = std::make_shared<RpcClient>();
auto robot_name = rpc_cli->getRobotNames().front();
SyncMovePtr ptr = rpc_cli->getRobotInterface(robot_name)->getSyncMove();
```

8.6.2.9 getTrace()

`TracePtr` arcs::common_interface::RobotInterface::getTrace ()

`Trace` () Get alarm information interface

Returns

Pointer to `Trace` object

Python function prototype

`getTrace(self: pyaubo_sdk.RobotInterface) -> arcs::common_interface::Trace`

C++ example

```
auto rpc_cli = std::make_shared<RpcClient>();
auto robot_name = rpc_cli->getRobotNames().front();
TracePtr ptr = rpc_cli->getRobotInterface(robot_name)->getTrace();
```

8.6.3 ForceControl()

Abstract class for force control interface

Collaboration diagram for ForceControl():



Functions

- int [arcs::common_interface::ForceControl::fcEnable](#) ()
Start force control
- int [arcs::common_interface::ForceControl::fcDisable](#) ()
End force control
- bool [arcs::common_interface::ForceControl::isFcEnabled](#) ()
Check if force control is enabled
- int [arcs::common_interface::ForceControl::setTargetForce](#) (const std::vector< double > &feature, const std::vector< bool > &compliance, const std::vector< double > &wrench, const std::vector< double > &limits, [TaskFrameType](#) type=TaskFrameType::FRAME_FORCE)
Set force control reference (target) value
- int [arcs::common_interface::ForceControl::setDynamicModel1](#) (const std::vector< double > &env_stiff, const std::vector< double > &damp_scale, const std::vector< double > &stiff_scale)
Set force control dynamics model
- [DynamicsModel](#) [arcs::common_interface::ForceControl::fcCalDynamicModel](#) (const std::vector< double > &env_stiff, const std::vector< double > &damp_scale, const std::vector< double > &stiff_scale)
Calculate force control dynamics model
- int [arcs::common_interface::ForceControl::setDynamicModelSearch](#) (const std::vector< double > &damp_scale, const std::vector< double > &stiff_scale)
Set force control dynamics model for hole searching scenario
- int [arcs::common_interface::ForceControl::setDynamicModelInsert](#) (const std::vector< double > &damp_scale, const std::vector< double > &stiff_scale)
Set force control dynamics model for insertion/extraction scenario
- int [arcs::common_interface::ForceControl::setDynamicModelContact](#) (const std::vector< double > &env_stiff, const std::vector< double > &damp_scale, const std::vector< double > &stiff_scale)
Set force control dynamics model for contact scenario
- int [arcs::common_interface::ForceControl::setDynamicModel](#) (const std::vector< double > &m, const std::vector< double > &d, const std::vector< double > &k)
Set force control dynamics model
- int [arcs::common_interface::ForceControl::fcSetSensorThresholds](#) (const std::vector< double > &thresholds)
Set force control thresholds
- int [arcs::common_interface::ForceControl::fcSetSensorLimits](#) (const std::vector< double > &limits)
Set force control maximum force limits
- std::vector< double > [arcs::common_interface::ForceControl::getFcSensorThresholds](#) ()
Get force control thresholds
- std::vector< double > [arcs::common_interface::ForceControl::getFcSensorLimits](#) ()
Get maximum force limits
- [DynamicsModel](#) [arcs::common_interface::ForceControl::getDynamicModel](#) ()
Get force control dynamics model
- int [arcs::common_interface::ForceControl::setCondForce](#) (const std::vector< double > &min, const std::vector< double > &max, bool outside, double timeout)
Set force control termination condition: Force.
- int [arcs::common_interface::ForceControl::setCondOrient](#) (const std::vector< double > &frame, double max_angle, double max_rot, bool outside, double timeout)
setCondOrient is used to set up an end condition for the tool orientation.
- int [arcs::common_interface::ForceControl::setCondPlane](#) (const std::vector< double > &plane, double timeout)
Specify a valid force control plane, x-y plane, z direction is valid
- int [arcs::common_interface::ForceControl::setCondCylinder](#) (const std::vector< double > &axis, double radius, bool outside, double timeout)

- Specify a valid force control cylinder by providing the central axis and cylinder radius, and specify whether the inside or outside of the cylinder is valid.
- int [arcs::common_interface::ForceControl::setCondSphere](#) (const std::vector< double > ¢er, double radius, bool outside, double timeout)
 - Specify a valid force control sphere by providing the center and radius, and specify whether the inside or outside of the sphere is valid.
- int [arcs::common_interface::ForceControl::setCondTcpSpeed](#) (const std::vector< double > &min, const std::vector< double > &max, bool outside, double timeout)
 - setCondTcpSpeed is used to setup an end condition for the TCP speed.
- int [arcs::common_interface::ForceControl::setCondDistance](#) (double distance, double timeout)
 - Force control termination condition - distance
- int [arcs::common_interface::ForceControl::setCondAdvanced](#) (const std::string &type, const std::vector< double > &args, double timeout)
 - Advanced force control termination condition
- int [arcs::common_interface::ForceControl::setCondActive](#) ()
 - Activate force control termination condition
- bool [arcs::common_interface::ForceControl::isCondFullfiled](#) ()
 - Check if the force control termination condition has been fulfilled
- int [arcs::common_interface::ForceControl::setSupvForce](#) (const std::vector< double > &min, const std::vector< double > &max)
 - setSupvForce is used to set up force supervision in Force Control.
- int [arcs::common_interface::ForceControl::setSupvOrient](#) (const std::vector< double > &frame, double max_angle, double max_rot, bool outside)
 - setSupvOrient is used to set up an supervision for the tool orientation.
- int [arcs::common_interface::ForceControl::setSupvPosBox](#) (const std::vector< double > &frame, const [Box](#) &box)
 - setSupvPosBox is used to set up position supervision in Force Control.
- int [arcs::common_interface::ForceControl::setSupvPosCylinder](#) (const std::vector< double > &frame, const [Cylinder](#) &cylinder)
- int [arcs::common_interface::ForceControl::setSupvPosSphere](#) (const std::vector< double > &frame, const [Sphere](#) &sphere)
- int [arcs::common_interface::ForceControl::setSupvReoriSpeed](#) (const std::vector< double > &speed_limit, bool outside, double timeout)
 - setSupvReoriSpeed is used to set up reorientation speed supervision in Force Control.
- int [arcs::common_interface::ForceControl::setSupvTcpSpeed](#) (const std::vector< double > &speed_limit, bool outside, double timeout)
 - setSupvTcpSpeed is used to set up TCP speed supervision in Force Control.
- int [arcs::common_interface::ForceControl::setLpFilter](#) (const std::vector< double > &cutoff_freq)
 - Set low-pass filter
- int [arcs::common_interface::ForceControl::resetLpFilter](#) ()
 - Reset low-pass filter
- int [arcs::common_interface::ForceControl::speedChangeEnable](#) (double ref_force)
 - The speedChangeEnable is used to activate FC SpeedChange function with desired reference and recover behavior.
- int [arcs::common_interface::ForceControl::speedChangeDisable](#) ()
 - Deactivate FC SpeedChange function.
- int [arcs::common_interface::ForceControl::speedChangeTune](#) (int speed_levels, double speed_ratio_min)
 - speedChangeTune is used to set FC SpeedChange system parameters to a new value.
- int [arcs::common_interface::ForceControl::setDamping](#) (const std::vector< double > &damping, double ramp_time)
 - setDamping is used to tune the damping in the force control coordinate systems.
- int [arcs::common_interface::ForceControl::resetDamping](#) ()

- Reset damping parameters
- int [arcs::common_interface::ForceControl::softFloatEnable](#) ()
Enable soft float function.
- int [arcs::common_interface::ForceControl::softFloatDisable](#) ()
Disable soft float function.
- bool [arcs::common_interface::ForceControl::isSoftFloatEnabled](#) ()
Returns whether soft float is enabled
- int [arcs::common_interface::ForceControl::setSoftFloatParams](#) (bool joint_space, const std::vector< bool > &select, const std::vector< double > &stiff_percent, const std::vector< double > &stiff_damp_ratio, const std::vector< double > &force_threshold, const std::vector< double > &force_limit)
Set soft float parameters
- int [arcs::common_interface::ForceControl::toolContact](#) (const std::vector< bool > &direction)
Detect contact between the tool and external objects
- std::vector< double > [arcs::common_interface::ForceControl::getActualJointPositionsHistory](#) (int steps)

8.6.3.1 Detailed Description

Abstract class for force control interface

8.6.3.2 Function Documentation

8.6.3.2.1 fcCalDynamicModel()

[DynamicsModel](#) [arcs::common_interface::ForceControl::fcCalDynamicModel](#) (
const std::vector< double > & env_stiff,
const std::vector< double > & damp_scale,
const std::vector< double > & stiff_scale)

Calculate force control dynamics model

Parameters

env_stiff	Environment stiffness, representing the workpiece stiffness in the contact axis direction, range [0, 1], default 0
damp_scale	Parameter representing damping level, range [0, 1], default 0.5
stiff_scale	Parameter representing stiffness level, range [0, 1], default 0.5

Returns

Force control dynamics model MDK AUBO_BUSY AUBO_BAD_STATE -AUBO_INVL_↵
ARGUMENT -AUBO_BAD_STATE

Exceptions

arcs::common_interface::AuboException	
---	--

Python Function Prototype

```
fcCalDynamicModel(self: pyaubo_sdk.ForceControl, arg0: List[float], arg1: List[float], arg2: List[
float]) -> Tuple[List[float], List[float], List[float]]
```

Lua Function Prototype

```
fcCalDynamicModel(env_stiff: table, damp_scale: table, stiff_scale: table) -> table
```

Lua example

```
fc_table = fcCalDynamicModel({0,0,0,0,0,0},{0.5,0.5,0.5,0.5,0.5,0.5},{0.5,0.5,0.5,0.5,0.5,0.5})
```

8.6.3.2.2 fcDisable()

```
int arcs::common_interface::ForceControl::fcDisable ()
```

End force control

fcDisable is used to disable Force Control. After a successful deactivation the robot is back in position control.

Returns

Return 0 if succeeded; return error code if failed AUBO_BUSY AUBO_BAD_STATE -AUBO_↵
BAD_STATE

Exceptions

arcs::common_interface::AuboException	
---	--

Python Function Prototype

```
fcDisable(self: pyaubo_sdk.ForceControl) -> int
```

Lua Function Prototype

```
fcDisable\(\) -> nil
```

Lua example

```
fcDisable\(\)
```

JSON-RPC Request example

```
{"jsonrpc":"2.0","method":"rob1.ForceControl.fcDisable","params":[],"id":1}
```

JSON-RPC Response example

```
{"id":1,"jsonrpc":"2.0","result":0}
```

8.6.3.2.3 fcEnable()

```
int arcs::common_interface::ForceControl::fcEnable ()
```

Start force control

fcEnable is used to enable Force Control. At the same time as Force Control is enabled, fcEnable is used to define the coordinate system for Force Control, and tune the force and torque damping. If a coordinate system is not specified in fcEnable a default force control coordinate system is created with the same orientation as the work object coordinate system. All Force Control supervisions are activated by fcEnable.

Returns

Return 0 if succeeded; return error code if failed AUBO_BUSY AUBO_BAD_STATE -AUBO_↵
BAD_STATE

Exceptions

arcs::common_interface::AuboException	
---	--

Python Function Prototype

```
fcEnable(self: pyaubo_sdk.ForceControl) -> int
```

Lua Function Prototype

```
fcEnable() -> nil
```

Lua example

```
fcEnable()
```

JSON-RPC Request example

```
{"jsonrpc":"2.0","method":"rob1.ForceControl.fcEnable","params":[],"id":1}
```

JSON-RPC Response example

```
{"id":1,"jsonrpc":"2.0","result":0}
```

8.6.3.2.4 fcSetSensorLimits()

```
int arcs::common_interface::ForceControl::fcSetSensorLimits (  
    const std::vector< double > & limits)
```

Set force control maximum force limits

Parameters

limits	Force limits
--------	--------------

Returns

Return 0 if succeeded; return error code if failed AUBO_BUSY AUBO_BAD_STATE -AUBO_↵
INVL_ARGUMENT -AUBO_BAD_STATE

Exceptions

arcs::common_interface::AuboException	
---	--

Python Function Prototype

```
fcSetSensorLimits(self: pyaubo_sdk.ForceControl, arg0: List[float]) -> int
```

Lua Function Prototype

```
fcSetSensorLimits(limits: table) -> nil
```

Lua example

```
fcSetSensorLimits({200.0,200.0,200.0,50.0,50.0,50.0})
```

8.6.3.2.5 fcSetSensorThresholds()

```
int arcs::common_interface::ForceControl::fcSetSensorThresholds (
    const std::vector< double > & thresholds)
```

Set force control thresholds

Parameters

thresholds	Force control thresholds
------------	--------------------------

Returns

Return 0 if succeeded; return error code if failed AUBO_BUSY AUBO_BAD_STATE -AUBO_↵
INVL_ARGUMENT -AUBO_BAD_STATE

Exceptions

arcs::common_interface::AuboException	
---	--

Python Function Prototype

```
fcSetSensorThresholds(self: pyaubo_sdk.ForceControl, arg0: List[float]) -> int
```

Lua Function Prototype

```
fcSetSensorThresholds(thresholds: table) -> nil
```

Lua example

```
fcSetSensorThresholds({1.0,1.0,1.0,0.5,0.5,0.5})
```

8.6.3.2.6 getActualJointPositionsHistory()

```
std::vector< double > arcs::common_interface::ForceControl::getActualJointPositionsHistory (
    int steps)
```

Get historical joint positions

According to the given number of cycle steps, go back the specified number of cycles from the joint state history to obtain the joint position data at that time.

Parameters

steps	Number of cycles to go back (unit: control cycles), the larger the value, the earlier the historical data is obtained
-------	---

Returns

```
std::vector<double> Joint positions (unit: radians) at the corresponding time point
```

8.6.3.2.7 getDynamicModel()

[DynamicsModel](#) arcs::common_interface::ForceControl::getDynamicModel ()

Get force control dynamics model

Returns

Force control dynamics model

Exceptions

arcs::common_interface::AuboException	
---	--

Python Function Prototype

```
getDynamicModel(self: pyaubo_sdk.ForceControl) -> Tuple[List[float], List[float], List[float]]
```

Lua Function Prototype

```
getDynamicModel() -> table
```

Lua example

```
Fc_Model = getDynamicModel()
```

JSON-RPC Request example

```
{"jsonrpc":"2.0","method":"rob1.ForceControl.getDynamicModel","params":[],"id":1}
```

JSON-RPC Response example

```
{"id":1,"jsonrpc":"2.0","result":[[],[20.0,20.0,20.0,5.0,5.0,5.0],[]]}
```

8.6.3.2.8 getFcSensorLimits()

```
std::vector< double > arcs::common_interface::ForceControl::getFcSensorLimits ()
```

Get maximum force limits

Returns

Force control maximum force limits

Exceptions

arcs::common_interface::AuboException	
---	--

Python Function Prototype

```
getFcSensorLimits(self: pyaubo_sdk.ForceControl) -> list
```

Lua Function Prototype

```
getFcSensorLimits() -> table
```

Lua example

```
Fc_Limits = getFcSensorLimits()
```

8.6.3.2.9 getFcSensorThresholds()

`std::vector< double > arcs::common_interface::ForceControl::getFcSensorThresholds ()`

Get force control thresholds

Returns

Minimum force control thresholds

Exceptions

arcs::common_interface::AuboException	
---	--

Python Function Prototype

`getFcSensorThresholds(self: pyaubo_sdk.ForceControl) -> list`

Lua Function Prototype

`getFcSensorThresholds() -> table`

Lua example

`Fc_Thresholds = getFcSensorThresholds\(\)`

8.6.3.2.10 isCondFullfiled()

`bool arcs::common_interface::ForceControl::isCondFullfiled ()`

Check if the force control termination condition has been fulfilled

Returns

Exceptions

arcs::common_interface::AuboException	
---	--

Python Function Prototype

```
isCondFullfiled(self: pyaubo_sdk.ForceControl) -> bool
```

Lua Function Prototype

```
isCondFullfiled() -> boolean
```

Lua example

```
status = isCondFullfiled()
```

JSON-RPC Request example

```
{"jsonrpc":"2.0","method":"rob1.ForceControl.isCondFullfiled","params":[],"id":1}
```

JSON-RPC Response example

```
{"id":1,"jsonrpc":"2.0","result":false}
```

8.6.3.2.11 isFcEnabled()

bool arcs::common_interface::ForceControl::isFcEnabled ()

Check if force control is enabled

Returns

Returns true if enabled, false if disabled

Exceptions

arcs::common_interface::AuboException	
---------------------------------------	--

Python Function Prototype

```
isFcEnabled(self: pyaubo_sdk.ForceControl) -> bool
```

Lua Function Prototype

```
isFcEnabled() -> boolean
```

Lua example

```
Fc_status = isFcEnabled()
```

JSON-RPC Request example

```
{"jsonrpc":"2.0","method":"rob1.ForceControl.isFcEnabled","params":[],"id":1}
```

JSON-RPC Response example

```
{"id":1,"jsonrpc":"2.0","result":false}
```

8.6.3.2.12 isSoftFloatEnabled()

bool arcs::common_interface::ForceControl::isSoftFloatEnabled ()

Returns whether soft float is enabled

Returns

Exceptions

arcs::common_interface::AuboException	
---------------------------------------	--

JSON-RPC Request example

```
{"jsonrpc": "2.0", "method": "rob1.ForceControl.isSoftFloatEnabled", "params": [], "id": 1}
```

JSON-RPC Response example

```
{"id": 1, "jsonrpc": "2.0", "result": false}
```

8.6.3.2.13 resetDamping()

int arcs::common_interface::ForceControl::resetDamping ()

Reset damping parameters

Returns

Return 0 if succeeded; return error code if failed AUBO_BUSY AUBO_BAD_STATE -AUBO_↵
BAD_STATE

Exceptions

arcs::common_interface::AuboException	
---------------------------------------	--

Python Function Prototype

```
resetDamping(self: pyaubo_sdk.ForceControl) -> int
```

Lua Function Prototype

```
resetDamping() -> nil
```

JSON-RPC Request example

```
{"jsonrpc": "2.0", "method": "rob1.ForceControl.resetDamping", "params": [], "id": 1}
```

JSON-RPC Response example

```
{"id": 1, "jsonrpc": "2.0", "result": 0}
```

8.6.3.2.14 resetLpFilter()

```
int arcs::common_interface::ForceControl::resetLpFilter ()
```

Reset low-pass filter

Returns

Return 0 if succeeded; return error code if failed AUBO_BUSY AUBO_BAD_STATE -AUBO_↵
BAD_STATE

Exceptions

arcs::common_interface::AuboException	
---	--

Python Function Prototype

```
resetLpFilter(self: pyaubo_sdk.ForceControl) -> int
```

Lua Function Prototype

```
resetLpFilter\(\) -> nil
```

JSON-RPC Request example

```
{"jsonrpc":"2.0","method":"rob1.ForceControl.resetLpFilter","params":[],"id":1}
```

JSON-RPC Response example

```
{"id":1,"jsonrpc":"2.0","result":0}
```

8.6.3.2.15 setCondActive()

```
int arcs::common_interface::ForceControl::setCondActive ()
```

Activate force control termination condition

Returns

Return 0 if succeeded; return error code if failed AUBO_BUSY AUBO_BAD_STATE -AUBO_↵
BAD_STATE

Exceptions

arcs::common_interface::AuboException	
---	--

Python Function Prototype

```
setCondActive(self: pyaubo_sdk.ForceControl) -> int
```

Lua Function Prototype

```
setCondActive() -> nil
```

Lua example

```
setCondActive()
```

JSON-RPC Request example

```
{"jsonrpc":"2.0","method":"rob1.ForceControl.setCondActive","params":[],"id":1}
```

JSON-RPC Response example

```
{"id":1,"jsonrpc":"2.0","result":0}
```

8.6.3.2.16 setCondAdvanced()

```
int arcs::common_interface::ForceControl::setCondAdvanced (
    const std::string & type,
    const std::vector< double > & args,
    double timeout)
```

Advanced force control termination condition

Parameters

type	Type
args	Arguments
timeout	Timeout

Returns

Return 0 if succeeded; return error code if failed AUBO_BUSY AUBO_BAD_STATE -AUBO_↔
~~INVL_ARGUMENT AUBO_BAD_STATE~~

Exceptions

arcs::common_interface::AuboException	
---	--

Lua Function Prototype

```
setCondAdvanced(type: number, args: table, timeout: number)
```

Lua example

```
setCondAdvanced(1, {0,1,0,0,0,0},10)
```

8.6.3.2.17 setCondCylinder()

```
int arcs::common_interface::ForceControl::setCondCylinder (
    const std::vector< double > & axis,
    double radius,
    bool outside,
    double timeout)
```

Specify a valid force control cylinder by providing the central axis and cylinder radius, and specify whether the inside or outside of the cylinder is valid.

Parameters

axis	
radius	
outside	
timeout	

Returns

Return 0 if succeeded; return error code if failed AUBO_BUSY AUBO_BAD_STATE -AUBO_↵
INVL_ARGUMENT -AUBO_BAD_STATE

Exceptions

arcs::common_interface::AuboException	
---	--

Python Function Prototype

```
setCondCylinder(self: pyaubo_sdk.ForceControl, arg0: List[float], arg1: float, arg2: bool, arg3: float) -> int
```

Lua Function Prototype

```
setCondCylinder(axis: table, radius: number, outside: boolean, timeout: number) -> nil
```

Lua example

```
setCondCylinder({0,1,0},10.0,true,5,0)
```

8.6.3.2.18 setCondDistance()

```
int arcs::common_interface::ForceControl::setCondDistance (
    double distance,
    double timeout)
```

Force control termination condition - distance

Parameters

distance	Distance
timeout	Timeout

Returns

Return 0 if succeeded; return error code if failed AUBO_BUSY AUBO_BAD_STATE -AUBO_↵
INVL_ARGUMENT -AUBO_BAD_STATE

Exceptions

arcs::common_interface::AuboException	
---	--

Lua Function Prototype

```
setCondDistance(distance: number, timeout: number)
```

Lua example

```
setCondDistance(0.2, 10.0)
```

8.6.3.2.19 setCondForce()

```
int arcs::common_interface::ForceControl::setCondForce (
    const std::vector< double > & min,
    const std::vector< double > & max,
    bool outside,
    double timeout)
```

Set force control termination condition: Force.

When the measured force is within the specified range, the force control algorithm will continue to run until the set condition is not met, at which point force control will exit.

The condition is later activated by calling the instruction FCCondWaitWhile, which will wait and hold the program execution while the specified condition is true. This allows the reference force, torque and movement to continue until the force is outside the specified limits.

A force condition is set up by defining minimum and maximum limits for the force in the directions of the force control coordinate system. Once activated with FCCondWaitWhile, the program execution will continue to wait while the measured force is within its specified limits.

It is possible to specify that the condition is fulfilled when the force is outside the specified limits instead. This is done by using the switch argument Outside. The condition on force is specified in the force control coordinate system. This coordinate system is setup by the user in the instruction FCAct.

Parameters

min	Minimum force/torque in each direction
max	Maximum force/torque in each direction
outside	false: valid within the specified range true: valid outside the specified range
timeout	Time limit in seconds; when this time is reached from the start of force control, force control will terminate regardless of whether the end condition is met

Returns

Return 0 if succeeded; return error code if failed AUBO_BUSY AUBO_BAD_STATE -AUBO_↵
INVL_ARGUMENT -AUBO_BAD_STATE

Exceptions

arcs::common_interface::AuboException

Python Function Prototype

```
setCondForce(self: pyaubo_sdk.ForceControl, arg0: List[float], arg1: List[float], arg2: bool, arg3: float) -> int
```

Lua Function Prototype

```
setCondForce(min: table, max: table, outside: boolean, timeout: number) -> nil
```

Lua example

```
setCondForce({0.1,0.1,0.1,0.1,0.1,0.1},{50,50,50,5,5,5},{true,true,true,true,true,true},10)
```

8.6.3.2.20 setCondOrient()

```
int arcs::common_interface::ForceControl::setCondOrient (
    const std::vector< double > & frame,
    double max_angle,
    double max_rot,
    bool outside,
    double timeout)
```

setCondOrient is used to set up an end condition for the tool orientation.

The condition will wait and hold the program execution while the specified condition is true. This allows the reference force, torque and movement to continue until the orientation is outside the specified limits.

An orientation condition is set up by defining a maximum angle and a maximum rotation from a reference orientation. The reference orientation is either defined by the current z direction of the tool, or by specifying an orientation in relation to the z direction of the work object.

Once activated, the tool orientation must be within the limits (or outside, if the argument Outside is used).

Parameters

frame	
max_angle	
max_rot	
outside	
timeout	

Returns

Return 0 if succeeded; return error code if failed AUBO_BUSY AUBO_BAD_STATE -AUBO_↵
INVL_ARGUMENT -AUBO_BAD_STATE

Exceptions

arcs::common_interface::AuboException	
---	--

Python Function Prototype

```
setCondOrient(self: pyaubo_sdk.ForceControl, arg0: List[float], arg1: float, arg2: float, arg3: bool,
arg4: float) -> int
```

Lua Function Prototype

```
setCondOrient(frame: table, max_angle: number, max_rot: number, outside: boolean, timeout:
number) -> nil
```

Lua example

```
setCondOrient({0.1, 0.3, 0.1, 0.3142, 0.0, 1.571},30.0,20.0,true,10.0)
```

8.6.3.2.21 setCondPlane()

```
int arcs::common_interface::ForceControl::setCondPlane (
    const std::vector< double > & plane,
    double timeout)
```

Specify a valid force control plane, x-y plane, z direction is valid

Parameters

plane={A,B,C,D}	Plane equation: $Ax + By + Cz + D = 0$ where $n = (A, B, C)$ is the normal vector of the plane, D is the distance required to move the plane to the origin (so D=0 means the plane passes through the origin)
timeout	

Returns

Return 0 if succeeded; return error code if failed AUBO_BUSY AUBO_BAD_STATE -AUBO_↔
INVL_ARGUMENT -AUBO_BAD_STATE

Exceptions

arcs::common_interface::AuboException	
---	--

Python Function Prototype

```
setCondPlane(self: pyaubo_sdk.ForceControl, arg0: List[float], arg1: float) -> int
```

Lua Function Prototype

```
setCondPlane(plane: table, timeout: number) -> nil
```

Lua example

```
setCondPlane({0.1, 0.3, 0.1, 0.3},10.0)
```

8.6.3.2.22 setCondSphere()

```
int arcs::common_interface::ForceControl::setCondSphere (
    const std::vector< double > & center,
    double radius,
    bool outside,
    double timeout)
```

Generated by Doxygen

Specify a valid force control sphere by providing the center and radius, and specify whether the inside or outside of the sphere is valid.

Parameters

center	
radius	
outside	
timeout	

Returns

Return 0 if succeeded; return error code if failed AUBO_BUSY AUBO_BAD_STATE -AUBO_↵
INVL_ARGUMENT -AUBO_BAD_STATE

Exceptions

arcs::common_interface::AuboException	
---	--

Python Function Prototype

```
setCondSphere(self: pyaubo_sdk.ForceControl, arg0: List[float], arg1: float, arg2: bool, arg3: float)
-> int
```

Lua Function Prototype

```
setCondSphere(center: table, radius: number, outside: boolean, timeout: number) -> nil
```

Lua example

```
setCondSphere({0.2, 0.5, 0.1, 1.57, 0, 0},10.0,true,5,0)
```

8.6.3.2.23 setCondTcpSpeed()

```
int arcs::common_interface::ForceControl::setCondTcpSpeed (
    const std::vector< double > & min,
    const std::vector< double > & max,
    bool outside,
    double timeout)
```

setCondTcpSpeed is used to setup an end condition for the TCP speed.

The condition is later activated by calling the instruction FCCondWaitWhile, which will wait and hold the program execution while the specified condition is true. This allows the reference force, torque and movement to continue until the speed is outside the specified limits.

A TCP speed condition is set up by defining minimum and maximum limits for the TCP speed in all directions of the work object. Once activated with FCCondWaitWhile, the program execution will continue to wait while the measured speed is within its specified limits.

It is possible to specify that the condition is fulfilled when the speed is outside the specified limits instead. This is done by setting the outside parameter to Outside. The condition TCP condition is fulfilled when the

Parameters

min	
max	
outside	
timeout	

Returns

Return 0 if succeeded; return error code if failed AUBO_BUSY AUBO_BAD_STATE -AUBO_↵
INVL_ARGUMENT -AUBO_BAD_STATE

Exceptions

arcs::common_interface::AuboException	
---	--

Python Function Prototype

```
setCondTcpSpeed(self: pyaubo_sdk.ForceControl, arg0: List[float], arg1: List[float], arg2: bool,
arg3: float) -> int
```

Lua Function Prototype

```
setCondTcpSpeed(min: table, max: table, outside: boolean, timeout: number) -> nil
```

Lua example

```
setCondTcpSpeed({0.2, 0.2, 0.2, 0.2, 0.2, 0.2},{1.2, 1.2, 1.2, 1.2, 1.2, 1.2},true,5,0)
```

8.6.3.2.24 setDamping()

```
int arcs::common_interface::ForceControl::setDamping (
    const std::vector< double > & damping,
    double ramp_time)
```

setDamping is used to tune the damping in the force control coordinate systems.

The parameters tuned are those described in Damping in Torque x Direction - Damping in Torque z Direction on page 255 and Damping in Force x Direction - Damping in Force z Direction on page 254.

Damping can be set in the configuration file or by the instruction FCAct. The difference is that this instruction can be used when force control is active. FCSetDampingTune tunes the actual values set by the instruction FCAct, not the value in the configuration file.

Parameters

damping	
ramp_time	

Returns

Return 0 if succeeded; return error code if failed AUBO_BUSY AUBO_BAD_STATE -AUBO_↵
INVL_ARGUMENT -AUBO_BAD_STATE

Exceptions

arcs::common_interface::AuboException	
---	--

Python Function Prototype

```
setDamping(self: pyaubo_sdk.ForceControl, arg0: List[float], arg1: float) -> int
```

Lua Function Prototype

```
setDamping(damping: table, ramp_time: number) -> nil
```

8.6.3.2.25 setDynamicModel()

```
int arcs::common_interface::ForceControl::setDynamicModel (
    const std::vector< double > & m,
    const std::vector< double > & d,
    const std::vector< double > & k)
```

Set force control dynamics model

Parameters

m	Mass parameters
d	Damping parameters
k	Stiffness parameters

Parameter unit description:

- Mass:

- Cartesian space: unit is kg, length is 6, order is [x, y, z, Rx, Ry, Rz]
- Joint space: unit is $\text{kg} \cdot \text{m}^2$, length is 6, order is [J1, J2, J3, J4, J5, J6]
- Damping:
 - Cartesian space: unit is $\text{N} \cdot \text{s}/\text{m}$, order is [x, y, z, Rx, Ry, Rz]
 - Joint space: unit is $\text{N} \cdot \text{m} \cdot \text{s}/\text{rad}$, order is [J1, J2, J3, J4, J5, J6]
- Stiffness:
 - Cartesian space: unit is N/m , order is [x, y, z, Rx, Ry, Rz]
 - Joint space: unit is $\text{N} \cdot \text{m}/\text{rad}$, order is [J1, J2, J3, J4, J5, J6]

Returns

Return 0 if succeeded; return error code if failed AUBO_BUSY AUBO_BAD_STATE -AUBO_↵
INVL_ARGUMENT -AUBO_BAD_STATE

Exceptions

arcs::common_interface::AuboException	
---	--

Python Function Prototype

```
setDynamicModel(self: pyaubo_sdk.ForceControl, arg0: List[float], arg1: List[float], arg2: List[↵
[float]) -> int
```

Lua Function Prototype

```
setDynamicModel(m: table, d: table, k: table) -> nil
```

Lua example

```
setDynamicModel({20.0, 20.0, 20.0, 10.0, 10.0, 10.0},{2000, 0, 0, 0, 0, 0},{0, 0, 0, 0, 0, 0})
```

8.6.3.2.26 setDynamicModel1()

```
int arcs::common_interface::ForceControl::setDynamicModel1 (
    const std::vector< double > & env_stiff,
    const std::vector< double > & damp_scale,
    const std::vector< double > & stiff_scale)
```

Set force control dynamics model

Parameters

env_stiff	Environment stiffness, representing the workpiece stiffness in the contact axis direction, range [0, 1], default 0
damp_scale	Parameter representing damping level, range [0, 1], default 0.5
stiff_scale	Parameter representing stiffness level, range [0, 1], default 0.5

Returns

Return 0 if succeeded; return error code if failed AUBO_BUSY AUBO_BAD_STATE -AUBO_↵
INVL_ARGUMENT -AUBO_BAD_STATE

Exceptions

arcs::common_interface::AuboException	
---	--

Python Function Prototype

```
setDynamicModel1(self: pyaubo_sdk.ForceControl, arg0: List[float], arg1: List[float], arg2: List↵  
[float]) -> int
```

Lua Function Prototype

```
setDynamicModel1(env_stiff: table, damp_scale: table, stiff_scale: table) -> nil
```

Lua example

```
setDynamicModel1({0,0,0,0,0,0},{0.5,0.5,0.5,0.5,0.5,0.5},{0.5,0.5,0.5,0.5,0.5,0.5})
```

8.6.3.2.27 setDynamicModelContact()

```
int arcs::common_interface::ForceControl::setDynamicModelContact (
    const std::vector< double > & env_stiff,
    const std::vector< double > & damp_scale,
    const std::vector< double > & stiff_scale)
```

Set force control dynamics model for contact scenario

Parameters

env_stiff	Parameter representing environment stiffness, range [0, 1], default 0
damp_scale	Parameter representing damping level, range [0, 1], default 0.5
stiff_scale	Parameter representing stiffness level, range [0, 1], default 0.5

Returns

Return 0 if succeeded; return error code if failed AUBO_BUSY AUBO_BAD_STATE -AUBO_↵
INVL_ARGUMENT -AUBO_BAD_STATE

Exceptions

arcs::common_interface::AuboException	
---	--

Python Function Prototype

```
setDynamicModelContact(self: pyaubo_sdk.ForceControl, arg0: List[float], arg1: List[float], arg2:
List[float]) -> int
```

Lua Function Prototype

```
setDynamicModelContact(env_stiff: table, damp_scale: table, stiff_scale: table) -> nil
```

Lua example

```
setDynamicModelContact({0,0,0,0,0,0},{0.5,0.5,0.5,0.5,0.5,0.5},{0.5,0.5,0.5,0.5,0.5,0.5})
```

8.6.3.2.28 setDynamicModelInsert()

```
int arcs::common_interface::ForceControl::setDynamicModelInsert (
    const std::vector< double > & damp_scale,
    const std::vector< double > & stiff_scale)
```

Set force control dynamics model for insertion/extraction scenario

Parameters

damp_scale	Parameter representing damping level, range [0, 1], default 0.5
stiff_scale	Parameter representing stiffness level, range [0, 1], default 0.5

Returns

Return 0 if succeeded; return error code if failed AUBO_BUSY AUBO_BAD_STATE AUBO_

Exceptions

arcs::common_interface::AuboException	
---	--

Python Function Prototype

```
setDynamicModelInsert(self: pyaubo_sdk.ForceControl, arg0: List[float], arg1: List[float]) -> int
```

Lua Function Prototype

```
setDynamicModelInsert(damp_scale: table, stiff_scale: table) -> nil
```

Lua example

```
setDynamicModelInsert({0.5,0.5,0.5,0.5,0.5,0.5},{0.5,0.5,0.5,0.5,0.5,0.5})
```

8.6.3.2.29 setDynamicModelSearch()

```
int arcs::common_interface::ForceControl::setDynamicModelSearch (
    const std::vector< double > & damp_scale,
    const std::vector< double > & stiff_scale)
```

Set force control dynamics model for hole searching scenario

Parameters

damp_scale	Parameter representing damping level, range [0, 1], default 0.5
stiff_scale	Parameter representing stiffness level, range [0, 1], default 0.5

Returns

Return 0 if succeeded; return error code if failed AUBO_BUSY AUBO_BAD_STATE -AUBO_↵
INVL_ARGUMENT -AUBO_BAD_STATE

Exceptions

arcs::common_interface::AuboException	
---	--

Python Function Prototype

```
setDynamicModelSearch(self: pyaubo_sdk.ForceControl, arg0: List[float], arg1: List[float]) -> int
```

Lua Function Prototype

```
setDynamicModelSearch(damp_scale: table, stiff_scale: table) -> nil
```

Lua example

```
setDynamicModelSearch({0.5,0.5,0.5,0.5,0.5,0.5},{0.5,0.5,0.5,0.5,0.5,0.5})
```

8.6.3.2.30 setLpFilter()

```
int arcs::common_interface::ForceControl::setLpFilter (
    const std::vector< double > & cutoff_freq)
```

Set low-pass filter

— force frame filter: filter measured force/torque +++ force loop filter: filter for force control output reference speed

FCSetLPFilterTune is used to change the response of force loop according to the description in Damping and LP-filter on page 103.

Parameters

cutoff_freq	Cutoff frequency
-------------	------------------

Returns

Return 0 if succeeded; return error code if failed AUBO_BUSY AUBO_BAD_STATE -AUBO_↔
INVL_ARGUMENT -AUBO_BAD_STATE

Exceptions

arcs::common_interface::AuboException	
---	--

Python Function Prototype

```
setLpFilter(self: pyaubo_sdk.ForceControl, arg0: List[float]) -> int
```

Lua Function Prototype

```
setLpFilter(cutoff_freq: table) -> nil
```

8.6.3.2.31 setSoftFloatParams()

```
int arcs::common_interface::ForceControl::setSoftFloatParams (
    bool joint_space,
    const std::vector< bool > & select,
    const std::vector< double > & stiff_percent,
    const std::vector< double > & stiff_damp_ratio,
    const std::vector< double > & force_threshold,
    const std::vector< double > & force_limit)
```

Parameters

joint_softfloat	Whether to enable soft float in joint space
select	Select which degrees of freedom to enable soft float
stiff_percent	Stiffness percentage
stiff_damp_ratio	Stiffness damping ratio
force_threshold	Force threshold
force_limit	Force limit

Returns

Return 0 if succeeded, otherwise error code

8.6.3.2.32 setSupvForce()

```
int arcs::common_interface::ForceControl::setSupvForce (
    const std::vector< double > & min,
    const std::vector< double > & max)
```

setSupvForce is used to set up force supervision in Force Control.

The supervision is activated when Force Control is activated with the instruction FCAct.

The force supervision is set up by defining minimum and maximum limits for the force in the directions of the force control coordinate system. Once activated, the supervision will stop the execution if the force is outside the allowed values. The force supervision is specified in the force control coordinate system. This coordinate system is setup by the user with the instruction FCAct.

Parameters

min	
max	

Returns

Return 0 if succeeded; return error code if failed AUBO_BUSY AUBO_BAD_STATE -AUBO_↵
INVL_ARGUMENT -AUBO_BAD_STATE

Exceptions

arcs::common_interface::AuboException	
---	--

Python Function Prototype

```
setSupvForce(self: pyaubo_sdk.ForceControl, arg0: List[float], arg1: List[float]) -> int
```

Lua Function Prototype

```
setSupvForce(min: table, max: table) -> nil
```

Lua example

```
setSupvForce({0.1 ,0.1 ,0.1 ,0.1 ,0.1 ,0.1}, {10.0 ,10.0 ,10.0 ,10.0 ,10.0 ,10.0})
```

8.6.3.2.33 setSupvOrient()

```
int arcs::common_interface::ForceControl::setSupvOrient (
    const std::vector< double > & frame,
    double max_angle,
    double max_rot,
    bool outside)
```

setSupvOrient is used to set up an supervision for the tool orientation.

The supervision is activated when Force Control is activated with the instruction FCAct.

An orientation supervision is set up by defining a maximum angle and a maximum rotation from a reference orientation. The reference orientation is either defined by the current z direction of the tool, or by specifying an orientation in relation to the z direction of the work object.

Once activated, the tool orientation must be within the limits otherwise the supervision will stop the execution.

Parameters

frame	
max_angle	
max_rot	
outside	

Returns

Return 0 if succeeded; return error code if failed AUBO_BUSY AUBO_BAD_STATE -AUBO_↵
INVL_ARGUMENT -AUBO_BAD_STATE

Exceptions

arcs::common_interface::AuboException	
---	--

Python Function Prototype

```
setSupvOrient(self: pyaubo_sdk.ForceControl, arg0: List[float], arg1: float, arg2: float, arg3: bool)
-> int
```

Lua Function Prototype

```
setSupvOrient(frame: table, max_angle: number, max_rot: number, outside: boolean) -> nil
```

8.6.3.2.34 setSupvPosBox()

```
int arcs::common_interface::ForceControl::setSupvPosBox (
    const std::vector< double > & frame,
    const Box & box)
```

setSupvPosBox is used to set up position supervision in Force Control.

Supervision is activated when Force Control is activated with the instruction FCAct. Position supervision is set up by defining a volume in space for the TCP. Once activated, the supervision will stop the execution if the TCP is outside this volume.

Parameters

frame	
box	

Returns

Return 0 if succeeded; return error code if failed AUBO_BUSY AUBO_BAD_STATE -AUBO_↵
INVL_ARGUMENT -AUBO_BAD_STATE

Exceptions

arcs::common_interface::AuboException	
---	--

Python Function Prototype

```
setSupvPosBox(self: pyaubo_sdk.ForceControl, arg0: List[float], arg1: List[float[6]]) -> int
```

Lua Function Prototype

```
setSupvPosBox(frame: table, box: table) -> nil
```


8.6.3.2.35 setSupvPosCylinder()

```
int arcs::common_interface::ForceControl::setSupvPosCylinder (
    const std::vector< double > & frame,
    const Cylinder & cylinder)
```

Parameters

frame	
cylinder	

Returns

Return 0 if succeeded; return error code if failed AUBO_BUSY AUBO_BAD_STATE -AUBO_↵
INVL_ARGUMENT -AUBO_BAD_STATE

Exceptions

arcs::common_interface::AuboException	
---	--

Python Function Prototype

```
setSupvPosCylinder(self: pyaubo_sdk.ForceControl, arg0: List[float], arg1: List[float[5]]) -> int
```

Lua Function Prototype

```
setSupvPosCylinder(frame: table, cylinder: table) -> nil
```

8.6.3.2.36 setSupvPosSphere()

```
int arcs::common_interface::ForceControl::setSupvPosSphere (
    const std::vector< double > & frame,
    const Sphere & sphere)
```

Parameters

frame	
sphere	

Return 0 if succeeded; return error code if failed AUBO_BUSY AUBO_BAD_STATE -AUBO_↵
INVL_ARGUMENT -AUBO_BAD_STATE

Exceptions

arcs::common_interface::AuboException	
---	--

Python Function Prototype

```
setSupvPosSphere(self: pyaubo_sdk.ForceControl, arg0: List[float], arg1: List[float[3]]) -> int
```

Lua Function Prototype

```
setSupvPosSphere(frame: table, sphere: table) -> nil
```

8.6.3.2.37 setSupvReoriSpeed()

```
int arcs::common_interface::ForceControl::setSupvReoriSpeed (
    const std::vector< double > & speed_limit,
    bool outside,
    double timeout)
```

setSupvReoriSpeed is used to set up reorientation speed supervision in Force Control.

The supervision is activated when Force Control is activated with the instruction FCAct.

The reorientation speed supervision is set up by defining minimum and maximum limits for the reorientation speed around the axis of the work object coordinate system. Once activated, the supervision will stop the execution if the values of the reorientation speed are too high.

There are two speed supervisions: FCSupvReoriSpeed and FCSupvTCPSpeed, which is described in section FCSupvTCPSpeed on page 199. Both supervisions may be required because:

- A robot axis can rotate with high speed while the TCP is stationary.
- The TCP can be far from the rotating axis and a small axis rotation may result in a high speed movement of the TCP.

Parameters

speed_limit	
outside	
timeout	

Returns

Return 0 if succeeded; return error code if failed AUBO_BUSY AUBO_BAD_STATE -AUBO_↵
INVL_ARGUMENT -AUBO_BAD_STATE

Exceptions

<code>arcs::common_interface::AuboException</code>	
--	--

Python Function Prototype

```
setSupvReoriSpeed(self: pyaubo_sdk.ForceControl, arg0: List[float], arg1: bool, arg2: float) -> int
```

Lua Function Prototype

```
setSupvReoriSpeed(speed_limit: table, outside: boolean, timeout: number) -> nil
```

8.6.3.2.38 setSupvTcpSpeed()

```
int arcs::common_interface::ForceControl::setSupvTcpSpeed (
    const std::vector< double > & speed_limit,
    bool outside,
    double timeout)
```

setSupvTcpSpeed is used to set up TCP speed supervision in Force Control.

The supervision is activated when Force Control is activated with the instruction FCAct. The TCP speed supervision is set up by defining minimum and maximum limits for the TCP speed in the directions of the work object coordinate system. Once activated, the supervision will stop the execution if too high TCP speed values are detected.

There are two speed supervisions: FCSupvTCPSpeed and FCSupvReoriSpeed, which is described in section FCSupvReoriSpeed on page 197.

Both supervisions may be required because:

- A robot axis can rotate with high speed while the TCP is stationary.
- The TCP can be far from the rotating axis and a small axis rotation may result in a high speed movement of the TCP.

Parameters

speed_limit	
outside	
timeout	

Returns

Return 0 if succeeded; return error code if failed AUBO_BUSY AUBO_BAD_STATE -AUBO_↵
 INVL_ARGUMENT -AUBO_BAD_STATE

Exceptions

arcs::common_interface::AuboException	
---	--

Python Function Prototype

```
setSupvTcpSpeed(self: pyaubo_sdk.ForceControl, arg0: List[float], arg1: bool, arg2: float) -> int
```

Lua Function Prototype

```
setSupvTcpSpeed(speed_limit: table, outside: boolean, timeout: number) -> nil
```

8.6.3.2.39 setTargetForce()

```
int arcs::common_interface::ForceControl::setTargetForce (
    const std::vector< double > & feature,
    const std::vector< bool > & compliance,
    const std::vector< double > & wrench,
    const std::vector< double > & limits,
    TaskFrameType type = TaskFrameType::FRAME_FORCE)
```

Set force control reference (target) value

Parameters

feature	Reference geometric feature for generating force control reference frame
compliance	Compliance axis (direction) selection
wrench	Target force/torque
limits	Velocity limits
type	Force control reference frame type

Usage Examples:

1. Base Coordinate Frame: feature = {0,0,0,0,0,0} type = TaskFrameType::NONE
2. Flange Coordinate Frame: feature = {0,0,0,0,0,0} type = TaskFrameType::TOOL_FORCE
3. TCP Coordinate Frame: feature = tcp_offset type = TaskFrameType::TOOL_FORCE
4. User Coordinate Frame (FRAME_FORCE): type = TaskFrameType::FRAME_FORCE feature should be the user-defined reference frame, for example, getTcpPose() means setting the force control frame to current TCP pose.

Returns

Return 0 if succeeded; return error code if failed AUBO_BUSY AUBO_BAD_STATE AUBO_INVL_ARGUMENT -AUBO_BAD_STATE

Exceptions

<code>arcs::common_interface::AuboException</code>	
--	--

Python Function Prototype

```
setTargetForce(self: pyaubo_sdk.ForceControl, arg0: List[float], arg1: List[bool], arg2: List[float],
arg3: List[float], arg4: arcs::common_interface::TaskFrameType) -> int
```

Lua Function Prototype

```
setTargetForce(feature: table, compliance: table, wrench: table, limits: table, type: number) -> nil
```

Lua example

```
setTargetForce({0,0,0,0,0,0},{true,false,false,false,false,false},{10,0,0,0,0,0},{0,0,0,0,0,0},4)
```

8.6.3.2.40 softFloatDisable()

```
int arcs::common_interface::ForceControl::softFloatDisable ()
```

Disable soft float function.

Returns

Return 0 if succeeded; return error code if failed AUBO_BUSY AUBO_BAD_STATE -AUBO_↔
BAD_STATE

Exceptions

<code>arcs::common_interface::AuboException</code>	
--	--

JSON-RPC Request example

```
{"jsonrpc":"2.0","method":"rob1.ForceControl.softFloatDisable","params":[],"id":1}
```

JSON-RPC Response example

```
{"id":1,"jsonrpc":"2.0","result":-1}
```

8.6.3.2.41 softFloatEnable()

Generated by Doxygen

```
int arcs::common_interface::ForceControl::softFloatEnable ()
```

Enable soft float function.

Exceptions

arcs::common_interface::AuboException	
---	--

JSON-RPC Request example

```
{"jsonrpc": "2.0", "method": "rob1.ForceControl.softFloatEnable", "params": [], "id": 1}
```

JSON-RPC Response example

```
{"id": 1, "jsonrpc": "2.0", "result": 0}
```

8.6.3.2.42 speedChangeDisable()

```
int arcs::common_interface::ForceControl::speedChangeDisable ()
```

Deactivate FC SpeedChange function.

Returns

Return 0 if succeeded; return error code if failed AUBO_BUSY AUBO_BAD_STATE -AUBO_↵
BAD_STATE

Exceptions

arcs::common_interface::AuboException	
---	--

Python Function Prototype

```
speedChangeDisable(self: pyaubo_sdk.ForceControl) -> int
```

Lua Function Prototype

```
speedChangeDisable\(\) -> nil
```

JSON-RPC Request example

```
{"jsonrpc": "2.0", "method": "rob1.ForceControl.speedChangeDisable", "params": [], "id": 1}
```

JSON-RPC Response example

```
{"id": 1, "jsonrpc": "2.0", "result": 0}
```

8.6.3.2.43 speedChangeEnable()

Parameters

ref_force	
-----------	--

Returns

Return 0 if succeeded; return error code if failed AUBO_BUSY AUBO_BAD_STATE -AUBO_↵
INVL_ARGUMENT -AUBO_BAD_STATE

Exceptions

arcs::common_interface::AuboException	
---	--

Python Function Prototype

speedChangeEnable(self: pyaubo_sdk.ForceControl, arg0: float) -> int

Lua Function Prototype

speedChangeEnable(ref_force: number) -> nil

8.6.3.2.44 speedChangeTune()

```
int arcs::common_interface::ForceControl::speedChangeTune (
    int speed_levels,
    double speed_ratio_min)
```

speedChangeTune is used to set FC SpeedChange system parameters to a new value.

Parameters

speed_levels	
speed_ratio_min	

Returns

Return 0 if succeeded; return error code if failed AUBO_BUSY AUBO_BAD_STATE -AUBO_↵
INVL_ARGUMENT -AUBO_BAD_STATE

Exceptions

arcs::common_interface::AuboException	
---	--

Python Function Prototype

```
speedChangeTune(self: pyaubo_sdk.ForceControl, arg0: int, arg1: float) -> int
```

Lua Function Prototype

```
speedChangeTune(speed_levels: number, speed_ratio_min: number) -> nil
```

8.6.3.2.45 toolContact()

```
int arcs::common_interface::ForceControl::toolContact (
    const std::vector< bool > & direction)
```

Detect contact between the tool and external objects

Parameters

direction	Expected contact direction. If all elements are 0, detect contact in all directions.
-----------	--

Returns

Returns the number of cycle steps back from the current point to the collision start point. If the return value is 0, no contact is detected.

8.6.4 IoControl (IO)

The [IoControl](#) class provides a series of interfaces for configuring and reading the robot's standard digital and analog IO, as well as setting output states.

Collaboration diagram for IoControl (IO):



Functions

- `int arcs::common_interface::IoControl::getStandardDigitalInputNum ()`
Get the number of standard digital inputs.
- `int arcs::common_interface::IoControl::getToolDigitalInputNum ()`
Get the number of tool digital IOs (including digital inputs and outputs).
- `int arcs::common_interface::IoControl::getConfigurableDigitalInputNum ()`
Get the number of configurable digital inputs.
- `int arcs::common_interface::IoControl::getStandardDigitalOutputNum ()`
Get the number of standard digital outputs.
- `int arcs::common_interface::IoControl::getToolDigitalOutputNum ()`
Get the number of tool digital IOs (including digital inputs and outputs).
- `int arcs::common_interface::IoControl::setToolIoInput (int index, bool input)`
Set the specified tool digital IO as input or output.
- `bool arcs::common_interface::IoControl::isToolIoInput (int index)`
Determine whether the specified tool digital IO is configured as input.
- `int arcs::common_interface::IoControl::getConfigurableDigitalOutputNum ()`
Get the number of configurable digital outputs.
- `int arcs::common_interface::IoControl::getStandardAnalogInputNum ()`
Get the number of standard analog inputs.
- `int arcs::common_interface::IoControl::getToolAnalogInputNum ()`
Get the number of tool analog inputs.
- `int arcs::common_interface::IoControl::getStandardAnalogOutputNum ()`
Get the number of standard analog outputs.
- `int arcs::common_interface::IoControl::getToolAnalogOutputNum ()`
Get the number of tool analog outputs.
- `int arcs::common_interface::IoControl::setDigitalInputActionDefault ()`
Set all digital input actions to no trigger.
- `int arcs::common_interface::IoControl::setStandardDigitalInputAction (int index, StandardInputAction action)`
Set the trigger action for standard digital input.
- `int arcs::common_interface::IoControl::setToolDigitalInputAction (int index, StandardInputAction action)`
Set the trigger action for tool digital input.
- `int arcs::common_interface::IoControl::setConfigurableDigitalInputAction (int index, StandardInputAction action)`
Set the trigger action for configurable digital input.
- `StandardInputAction arcs::common_interface::IoControl::getStandardDigitalInputAction (int index)`
Get the trigger action for standard digital input.
- `StandardInputAction arcs::common_interface::IoControl::getToolDigitalInputAction (int index)`
Get the trigger action for tool digital input.
- `StandardInputAction arcs::common_interface::IoControl::getConfigurableDigitalInputAction (int index)`
Get the trigger action for configurable digital input.
- `int arcs::common_interface::IoControl::setDigitalOutputRunstateDefault ()`
Set all digital output runstates to None.
- `int arcs::common_interface::IoControl::setStandardDigitalOutputRunstate (int index, StandardOutputRunState runstate)`
Set the runstate for standard digital output.
- `int arcs::common_interface::IoControl::setToolDigitalOutputRunstate (int index, StandardOutputRunState runstate)`

- Set the runstate for tool digital output.

 - `int arcs::common_interface::IoControl::setConfigurableDigitalOutputRunstate (int index, StandardOutputRunState runstate)`

Set the runstate for configurable digital output.
- `StandardOutputRunState arcs::common_interface::IoControl::getStandardDigitalOutputRunstate (int index)`

Get the runstate for standard digital output.
- `StandardOutputRunState arcs::common_interface::IoControl::getToolDigitalOutputRunstate (int index)`

Get the runstate for tool digital output.
- `StandardOutputRunState arcs::common_interface::IoControl::getConfigurableDigitalOutputRunstate (int index)`

Get the runstate for configurable digital output.
- `int arcs::common_interface::IoControl::setStandardAnalogOutputRunstate (int index, StandardOutputRunState runstate)`

Set the runstate for standard analog output.
- `int arcs::common_interface::IoControl::setToolAnalogOutputRunstate (int index, StandardOutputRunState runstate)`

Set the runstate for tool analog output.
- `StandardOutputRunState arcs::common_interface::IoControl::getStandardAnalogOutputRunstate (int index)`

Get the runstate for standard analog output.
- `StandardOutputRunState arcs::common_interface::IoControl::getToolAnalogOutputRunstate (int index)`

Get the runstate for tool analog output.
- `int arcs::common_interface::IoControl::setDigitalOutputAfterEStopDefault ()`

Set all digital output states after emergency stop to default(no change)
- `int arcs::common_interface::IoControl::setAnalogOutputAfterEStopDefault ()`

Set all analog output states after emergency stop to default(no change)
- `int arcs::common_interface::IoControl::setStandardDigitalOutputAfterEStop (int index, bool value)`

Set the value of a standard digital output after emergency stop
- `int arcs::common_interface::IoControl::setConfigurableDigitalOutputAfterEStop (int index, bool value)`

Set the value of a configurable digital output after emergency stop
- `int arcs::common_interface::IoControl::setStandardAnalogOutputAfterEStop (int index, double value)`

Set the value of standard analog output after emergency stop
- `int arcs::common_interface::IoControl::setStandardAnalogInputDomain (int index, int domain)`

Set the range of standard analog input.
- `int arcs::common_interface::IoControl::setToolAnalogInputDomain (int index, int domain)`

Set the range of tool analog input.
- `int arcs::common_interface::IoControl::getStandardAnalogInputDomain (int index)`

Get the domain of standard analog input.
- `int arcs::common_interface::IoControl::getToolAnalogInputDomain (int index)`

Get the domain of tool analog input.
- `int arcs::common_interface::IoControl::setStandardAnalogOutputDomain (int index, int domain)`

Set the range of standard analog output.
- `int arcs::common_interface::IoControl::setToolAnalogOutputDomain (int index, int domain)`

Set the range of tool analog output.
- `int arcs::common_interface::IoControl::getStandardAnalogOutputDomain (int index)`

Get the domain of standard analog output.

- int [arcs::common_interface::IoControl::getToolAnalogOutputDomain](#) (int index)
Get the domain of tool analog output.
- int [arcs::common_interface::IoControl::setToolVoltageOutputDomain](#) (int domain)
Set the tool voltage output value (unit: V)
- int [arcs::common_interface::IoControl::getToolVoltageOutputDomain](#) ()
Get the tool voltage output value (unit: V)
- int [arcs::common_interface::IoControl::setStandardDigitalOutput](#) (int index, bool value)
Set the value of a standard digital output.
- int [arcs::common_interface::IoControl::setStandardDigitalOutputPulse](#) (int index, bool value, double duration)
Set digital output pulse.
- int [arcs::common_interface::IoControl::setToolDigitalOutput](#) (int index, bool value)
Set the value of tool digital output.
- int [arcs::common_interface::IoControl::setToolDigitalOutputPulse](#) (int index, bool value, double duration)
Set tool digital output pulse.
- int [arcs::common_interface::IoControl::setConfigurableDigitalOutput](#) (int index, bool value)
Set the value of configurable digital output.
- int [arcs::common_interface::IoControl::setConfigurableDigitalOutputPulse](#) (int index, bool value, double duration)
Set configurable digital output pulse.
- int [arcs::common_interface::IoControl::setStandardAnalogOutput](#) (int index, double value)
Set the value of standard analog output.
- int [arcs::common_interface::IoControl::setToolAnalogOutput](#) (int index, double value)
Set the value of tool analog output.
- bool [arcs::common_interface::IoControl::getStandardDigitalInput](#) (int index)
Get the value of a standard digital input.
- uint32_t [arcs::common_interface::IoControl::getStandardDigitalInputs](#) ()
Get all standard digital input values.
- bool [arcs::common_interface::IoControl::getToolDigitalInput](#) (int index)
Get the value of tool digital input.
- uint32_t [arcs::common_interface::IoControl::getToolDigitalInputs](#) ()
Get all tool digital input values.
- bool [arcs::common_interface::IoControl::getConfigurableDigitalInput](#) (int index)
Get the value of configurable digital input.
- uint32_t [arcs::common_interface::IoControl::getConfigurableDigitalInputs](#) ()
Get all configurable digital input values.
- bool [arcs::common_interface::IoControl::getStandardDigitalOutput](#) (int index)
Get the value of a standard digital output.
- uint32_t [arcs::common_interface::IoControl::getStandardDigitalOutputs](#) ()
Get all standard digital output values.
- bool [arcs::common_interface::IoControl::getToolDigitalOutput](#) (int index)
Get the value of tool digital output.
- uint32_t [arcs::common_interface::IoControl::getToolDigitalOutputs](#) ()
Get all tool digital output values.
- bool [arcs::common_interface::IoControl::getConfigurableDigitalOutput](#) (int index)
Get the value of configurable digital output.
- uint32_t [arcs::common_interface::IoControl::getConfigurableDigitalOutputs](#) ()
Get all configurable digital output values.
- double [arcs::common_interface::IoControl::getStandardAnalogInput](#) (int index)
Get the value of standard analog input.

- double `arcs::common_interface::IoControl::getToolAnalogInput` (int index)
Get the value of tool analog input.
- double `arcs::common_interface::IoControl::getStandardAnalogOutput` (int index)
Get the value of standard analog output.
- double `arcs::common_interface::IoControl::getToolAnalogOutput` (int index)
Get the value of tool analog output.
- int `arcs::common_interface::IoControl::getStaticLinkInputNum` ()
Get the number of static link inputs.
- int `arcs::common_interface::IoControl::getStaticLinkOutputNum` ()
Get the number of static link outputs.
- uint32_t `arcs::common_interface::IoControl::getStaticLinkInputs` ()
Get all static link input values.
- uint32_t `arcs::common_interface::IoControl::getStaticLinkOutputs` ()
Get all static link output values.
- bool `arcs::common_interface::IoControl::hasEncoderSensor` ()
Whether the robot is equipped with an encoder.
- int `arcs::common_interface::IoControl::setEncDecoderType` (int type, int range_id)
Set the decoding method of the integrated encoder.
- int `arcs::common_interface::IoControl::setEncTickCount` (int tick)
Set the tick count of the integrated encoder.
- int `arcs::common_interface::IoControl::getEncDecoderType` ()
Get the decoder type of the encoder.
- int `arcs::common_interface::IoControl::getEncTickCount` ()
Get the tick count
- int `arcs::common_interface::IoControl::unwindEncDeltaTickCount` (int delta_count)
Prevent counting errors when the count exceeds the range
- bool `arcs::common_interface::IoControl::getToolButtonStatus` ()
Get the status of the tool button.
- uint32_t `arcs::common_interface::IoControl::getHandleIoStatus` ()
Get the status of handle buttons.
- int `arcs::common_interface::IoControl::getHandleType` ()
Get the handle type.

8.6.4.1 Detailed Description

The `IoControl` class provides a series of interfaces for configuring and reading the robot's standard digital and analog IO, as well as setting output states.

1. Get the number of various IOs
2. Configure IO input/output functions
3. Configuration of configurable IOs
4. Set and read the input/output range of analog IOs

Standard digital input/output: Standard IOs on the control cabinet IO panel

Tool digital input/output: Digital IOs exposed via the tool-end connector

Configurable input/output: Can be configured as safety IO or general digital IO

8.6.4.2 Function Documentation

8.6.4.2.1 getConfigurableDigitalInput()

```
bool arcs::common_interface::IoControl::getConfigurableDigitalInput (
    int index)
```

Get the value of configurable digital input.

Note

Can be used to get the value of safety IO.

Parameters

index	Indicates the IO pin, starting from 0. For example, 0 means the first pin.
-------	--

Returns

Returns true for high level; false for low level.

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
getConfigurableDigitalInput(self: pyaubo_sdk.IoControl, arg0: int) -> bool
```

Lua function prototype

```
getConfigurableDigitalInput(index: number) -> boolean
```

Lua example

```
status = getConfigurableDigitalInput(0)
```

JSON-RPC request example

```
{"jsonrpc": "2.0", "method": "rob1.IoControl.getConfigurableDigitalInput", "params": [0], "id": 1}
```

JSON-RPC response example

```
{"id": 1, "jsonrpc": "2.0", "result": false}
```

Parameters

index	Indicates the IO pin, starting from 0. For example, 0 means the first pin.
-------	--

Returns

Returns the input trigger action.

Python function prototype

```
getConfigurableDigitalInputAction(self: pyaubo_sdk.IoControl, arg0: int) -> arcs::common_interface::StandardInp
```

Lua function prototype

```
getConfigurableDigitalInputAction(index: number) -> number
```

Lua example

```
num = getConfigurableDigitalInputAction(0)
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.IoControl.getConfigurableDigitalInputAction","params":[0],  
,"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":"Default"}
```

8.6.4.2.3 getConfigurableDigitalInputNum()

```
int arcs::common_interface::IoControl::getConfigurableDigitalInputNum ()
```

Get the number of configurable digital inputs.

Returns

Number of configurable digital inputs.

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
getConfigurableDigitalInputNum(self: pyaubo_sdk.IoControl) -> int
```

Lua function prototype

```
getConfigurableDigitalInputNum() -> number
```

Lua example

```
num = getConfigurableDigitalInputNum()
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.IoControl.getConfigurableDigitalInputNum","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":16}
```

8.6.4.2.4 getConfigurableDigitalInputs()

```
uint32_t arcs::common_interface::IoControl::getConfigurableDigitalInputs ()
```

Get all configurable digital input values.

Note

Can be used to get the value of safety IO.

Returns

All configurable digital input values.

For example, if the return value is 2863267846, its binary representation is 1010101010101000000000000000110.

The lower 16 bits represent the status of all input pins, the least significant bit indicates the input status of pin 0, the second least significant bit indicates pin 1, and so on.

1 means high level, 0 means low level.

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
getConfigurableDigitalInputs(self: pyaubo_sdk.IoControl) -> int
```

Lua function prototype

```
getConfigurableDigitalInputs() -> number
```

Lua example

```
num = getConfigurableDigitalInputs()
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.IoControl.getConfigurableDigitalInputs","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":0}
```

8.6.4.2.5 getConfigurableDigitalOutput()

```
bool arcs::common_interface::IoControl::getConfigurableDigitalOutput (
    int index)
```

Get the value of configurable digital output.

Note

Can be used to get the value of safety IO.

Parameters

index	Indicates the IO pin, starting from 0. For example, 0 means the first pin.
-------	--

Returns

Returns true for high level; false for low level.

Exceptions

arcs::common_interface::AuboException	
---------------------------------------	--

Python function prototype

```
getConfigurableDigitalOutput(self: pyaubo_sdk.IoControl, arg0: int) -> bool
```

Lua function prototype

```
getConfigurableDigitalOutput(index: number) -> boolean
```

Lua example

```
status = getConfigurableDigitalOutput(0)
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.IoControl.getConfigurableDigitalOutput","params":[0],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":true}
```

8.6.4.2.6 getConfigurableDigitalOutputNum()

```
int arcs::common_interface::IoControl::getConfigurableDigitalOutputNum ()
```

Get the number of configurable digital outputs.

Returns

Number of configurable digital outputs.

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
getConfigurableDigitalOutputNum(self: pyaubo_sdk.IoControl) -> int
```

Lua function prototype

```
getConfigurableDigitalOutputNum\(\) -> number
```

Lua example

```
num = getConfigurableDigitalOutputNum\(\)
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.IoControl.getConfigurableDigitalOutputNum","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":16}
```

8.6.4.2.7 getConfigurableDigitalOutputRunstate()

[StandardOutputRunState](#) arcs::common_interface::IoControl::getConfigurableDigitalOutputRunstate (
int index)

Get the runstate for configurable digital output.

Parameters

index	Indicates the IO pin, starting from 0. For example, 0 means the first pin.
-------	--

Returns

Output runstate selection

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
getConfigurableDigitalOutputRunstate(self: pyaubo_sdk.IoControl, arg0: int) -> arcs::common_interface::StandardOutputRunState
```

Lua function prototype

```
getConfigurableDigitalOutputRunstate(index: number) -> number
```

Lua example

```
num = getConfigurableDigitalOutputRunstate(0)
```

JSON-RPC request example

```
{ "jsonrpc": "2.0", "method": "rob1.IoControl.getConfigurableDigitalOutputRunstate", "params": [0], "id": 1 }
```

JSON-RPC response example

```
{ "id": 1, "jsonrpc": "2.0", "result": "None" }
```

8.6.4.2.8 getConfigurableDigitalOutputs()

```
uint32_t arcs::common_interface::IoControl::getConfigurableDigitalOutputs ()
```

Get all configurable digital output values.

Note

Can be used to get the value of safety IO.

Returns

All configurable digital output values.

For example, if the return value is 2863267846, its binary representation is 1010101010101000000000000000110.

The lower 16 bits represent the status of all output pins, the least significant bit indicates the output status of pin 0, the second least significant bit indicates pin 1, and so on.

1 means high level, 0 means low level.

Exceptions

<code>arcs::common_interface::AuboException</code>	
--	--

Python function prototype

```
getConfigurableDigitalOutputs(self: pyaubo_sdk.IoControl) -> int
```

Lua function prototype

```
getConfigurableDigitalOutputs() -> number
```

Lua example

```
num = getConfigurableDigitalOutputs()
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.IoControl.getConfigurableDigitalOutputs","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":1}
```

8.6.4.2.9 getEncDecoderType()

```
arcs::common_interface::IoControl::getEncDecoderType ()
```

Get the decoder type of the encoder.

Exceptions

arcs::common_interface::AuboException	
---	--

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.IoControl.getEncDecoderType","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":0}
```

8.6.4.2.10 getEncTickCount()

```
int arcs::common_interface::IoControl::getEncTickCount ()
```

Get the tick count

Returns

Returns 0 on success; error code on failure. AUBO_NO_ACCESS AUBO_BUSY AUBO_BAD_↵
_STATE -AUBO_BAD_STATE

Exceptions

arcs::common_interface::AuboException	
---	--

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.IoControl.getEncTickCount","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":0}
```

8.6.4.2.11 getHandleIoStatus()

```
uint32_t arcs::common_interface::IoControl::getHandleIoStatus ()
```

Get the status of handle buttons.

Note

Get the status of handle buttons.

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
getHandleIoStatus(self: pyaubo_sdk.IoControl) -> int
```

Lua function prototype

```
getHandleIoStatus() -> number
```

Lua example

```
num = getHandleIoStatus()
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.IoControl.getHandleIoStatus","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":0}
```

8.6.4.2.12 getHandleType()

```
int arcs::common_interface::IoControl::getHandleType ()
```

Get the handle type.

Returns

```
type
```

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
getHandleType() -> int
```

Lua function prototype

```
getHandleType() -> int
```

Lua example

```
int_num = getHandleType()
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.IoControl.getHandleType","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":0}
```

8.6.4.2.13 getStandardAnalogInput()

```
double arcs::common_interface::IoControl::getStandardAnalogInput (
    int index)
```

Get the value of standard analog input.

Parameters

index	Indicates the IO pin, starting from 0. For example, 0 means the first pin.
-------	--

Returns

Standard analog input value.

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
getStandardAnalogInput(self: pyaubo_sdk.IoControl, arg0: int) -> float
```

Lua function prototype

```
getStandardAnalogInput(index: number) -> number
```

Lua example

```
getStandardAnalogInput(0)
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.IoControl.getStandardAnalogInput","params":[0],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":0.0}
```

8.6.4.2.14 getStandardAnalogInputDomain()

```
int arcs::common_interface::IoControl::getStandardAnalogInputDomain (
    int index)
```

Get the domain of standard analog input.

Parameters

index	Indicates the IO pin, starting from 0. For example, 0 means the first pin.
-------	--

Returns

Standard analog input domain

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
getStandardAnalogInputDomain(self: pyaubo_sdk.IoControl, arg0: int) -> int
```

Lua function prototype

```
getStandardAnalogInputDomain(index: number) -> number
```

Lua example

```
num = getStandardAnalogInputDomain(0)
```

JSON-RPC request example

```
{"jsonrpc": "2.0", "method": "rob1.IoControl.getStandardAnalogInputDomain", "params": [0, "id": 1]}
```

JSON-RPC response example

```
{"id": 1, "jsonrpc": "2.0", "result": 0}
```

8.6.4.2.15 getStandardAnalogInputNum()

Generated by Doxygen

```
int arcs::common_interface::IoControl::getStandardAnalogInputNum ()
```

Get the number of standard analog inputs.

Exceptions

<code>arcs::common_interface::AuboException</code>	
--	--

Python function prototype

```
getStandardAnalogInputNum(self: pyaubo_sdk.IoControl) -> int
```

Lua function prototype

```
getStandardAnalogInputNum() -> number
```

Lua example

```
num = getStandardAnalogInputNum()
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.IoControl.getStandardAnalogInputNum","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":2}
```

8.6.4.2.16 getStandardAnalogOutput()

```
double arcs::common_interface::IoControl::getStandardAnalogOutput (
    int index)
```

Get the value of standard analog output.

Parameters

index	Indicates the IO pin, starting from 0. For example, 0 means the first pin.
-------	--

Returns

Standard analog output value.

Exceptions

<code>arcs::common_interface::AuboException</code>	
--	--

Python function prototype

```
getStandardAnalogOutput(self: pyaubo_sdk.IoControl, arg0: int) -> float
```

Lua function prototype

```
getStandardAnalogOutput(index: number) -> number
```

Lua example

```
num = getStandardAnalogOutput(0)
```

JSON-RPC request example

```
{"jsonrpc": "2.0", "method": "rob1.IoControl.getStandardAnalogOutput", "params": [0], "id": 1}
```

JSON-RPC response example

```
{"id": 1, "jsonrpc": "2.0", "result": 0.0}
```

8.6.4.2.17 getStandardAnalogOutputDomain()

```
int arcs::common_interface::IoControl::getStandardAnalogOutputDomain (
    int index)
```

Get the domain of standard analog output.

Parameters

index	Indicates the IO pin, starting from 0. For example, 0 means the first pin.
-------	--

Returns

Standard analog output domain

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
getStandardAnalogOutputDomain(self: pyaubo_sdk.IoControl, arg0: int) -> int
```

Lua function prototype

```
getStandardAnalogOutputDomain(index: number) -> number
```

Lua example

```
num = getStandardAnalogOutputDomain(0)
```

JSON-RPC request example

```
{ "jsonrpc": "2.0", "method": "rob1.IoControl.getStandardAnalogOutputDomain", "params": [0],  
  "id": 1 }
```

JSON-RPC response example

```
{ "id": 1, "jsonrpc": "2.0", "result": 0 }
```

8.6.4.2.18 getStandardAnalogOutputNum()

```
int arcs::common_interface::IoControl::getStandardAnalogOutputNum ()
```

Get the number of standard analog outputs.

Returns

Number of standard analog outputs.

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
getStandardAnalogOutputNum(self: pyaubo_sdk.IoControl) -> int
```

Lua function prototype

```
getStandardAnalogOutputNum() -> number
```

Lua example

```
num = getStandardAnalogOutputNum()
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.IoControl.getStandardAnalogOutputNum","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":2}
```

8.6.4.2.19 getStandardAnalogOutputRunstate()

[StandardOutputRunState](#) arcs::common_interface::IoControl::getStandardAnalogOutputRunstate (int index)

Get the runstate for standard analog output.

Parameters

index	Indicates the IO pin, starting from 0. For example, 0 means the first pin.
-------	--

Returns

Standard analog output runstate selection

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
getStandardAnalogOutputRunstate(self: pyaubo_sdk.IoControl, arg0: int) -> arcs::common\_interface::StandardOutputRunState
```

Lua function prototype

```
getStandardAnalogOutputRunstate(index: number) -> number
```

Lua example

```
num = getStandardAnalogOutputRunstate(0)
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.IoControl.getStandardAnalogOutputRunstate","params":[0],  
"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":"None"}
```

8.6.4.2.20 getStandardDigitalInput()

```
bool arcs::common_interface::IoControl::getStandardDigitalInput (  
    int index)
```

Get the value of a standard digital input.

Parameters

index	Indicates the IO pin, starting from 0. For example, 0 means the first pin.
-------	--

Returns

Returns true for high level; false for low level.

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
getStandardDigitalInput(self: pyaubo_sdk.IoControl, arg0: int) -> bool
```

Lua function prototype

```
getStandardDigitalInput(index: number) -> boolean
```

Lua example

```
status = getStandardDigitalInput(0)
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.IoControl.getStandardDigitalInput","params":[0],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":false}
```

8.6.4.2.21 getStandardDigitalInputAction()

[StandardInputAction](#) arcs::common_interface::IoControl::getStandardDigitalInputAction (int index)

Get the trigger action for standard digital input.

Parameters

index	Indicates the IO pin, starting from 0. For example, 0 means the first pin.
-------	--

Returns

Standard digital input trigger action

Python function prototype

```
getStandardDigitalInputAction(self: pyaubo_sdk.IoControl, arg0: int) -> arcs::common\_interface::StandardInputA
```

Lua function prototype

```
getStandardDigitalInputAction(index: number) -> number
```

Lua example

```
num = getStandardDigitalInputAction(0)
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.IoControl.getStandardDigitalInputAction","params":[0],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":"Default"}
```

8.6.4.2.22 getStandardDigitalInputNum()

```
int arcs::common_interface::IoControl::getStandardDigitalInputNum ()
```

Get the number of standard digital inputs.

Returns

Number of standard digital inputs.

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
getStandardDigitalInputNum(self: pyaubo_sdk.IoControl) -> int
```

Lua function prototype

```
getStandardDigitalInputNum\(\) -> number
```

Lua example

```
num = getStandardDigitalInputNum\(\)
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.IoControl.getStandardDigitalInputNum","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":16}
```

8.6.4.2.23 getStandardDigitalInputs()

```
uint32_t arcs::common_interface::IoControl::getStandardDigitalInputs ()
```

Get all standard digital input values.

Returns

All standard digital input values.

For example, if the return value is 2863267846, its binary representation is 101010101010101000000000000000110.

— The lower 16 bits represent the status of all standard digital inputs, the least significant bit indicates the input status of DI00, the second least significant bit indicates DI01, and so on. 1 means high level, 0 means low level.

Exceptions

<code>arcs::common_interface::AuboException</code>	
--	--

Python function prototype

```
getStandardDigitalInputs(self: pyaubo_sdk.IoControl) -> int
```

Lua function prototype

```
getStandardDigitalInputs() -> number
```

Lua example

```
num = getStandardDigitalInputs()
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.IoControl.getStandardDigitalInputs","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":0}
```

8.6.4.2.24 getStandardDigitalOutput()

```
bool arcs::common_interface::IoControl::getStandardDigitalOutput (
    int index)
```

Get the value of a standard digital output.

Parameters

index	Indicates the IO pin, starting from 0. For example, 0 means the first pin.
-------	--

Returns

Returns true for high level; false for low level.

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
getStandardDigitalOutput(self: pyaubo_sdk.IoControl, arg0: int) -> bool
```

Lua function prototype

```
getStandardDigitalOutput(index: number) -> boolean
```

Lua example

```
status = getStandardDigitalOutput(0)
```

JSON-RPC request example

```
{ "jsonrpc": "2.0", "method": "rob1.IoControl.getStandardDigitalOutput", "params": [0], "id": 1 }
```

JSON-RPC response example

```
{ "id": 1, "jsonrpc": "2.0", "result": true }
```

8.6.4.2.25 getStandardDigitalOutputNum()

```
int arcs::common_interface::IoControl::getStandardDigitalOutputNum ()
```

Get the number of standard digital outputs.

Returns

Number of standard digital outputs.

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
getStandardDigitalOutputNum(self: pyaubo_sdk.IoControl) -> int
```


Lua function prototype

```
getStandardDigitalOutputNum() -> number
```

Lua example

```
num = getStandardDigitalOutputNum()
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.IoControl.getStandardDigitalOutputNum","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":8}
```

8.6.4.2.26 getStandardDigitalOutputRunstate()

[StandardOutputRunState](#) arcs::common_interface::IoControl::getStandardDigitalOutputRunstate (int index)

Get the runstate for standard digital output.

Parameters

index	Indicates the IO pin, starting from 0. For example, 0 means the first pin.
-------	--

Returns

Output runstate selection

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
getStandardDigitalOutputRunstate(self: pyaubo_sdk.IoControl, arg0: int) -> arcs::common\_interface::StandardOutputRunState
```

Lua function prototype

```
getStandardDigitalOutputRunstate(index: number) -> number
```

Lua example

```
num = getStandardDigitalOutputRunstate(0)
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.IoControl.getStandardDigitalOutputRunstate","params":[0],  
"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":"None"}
```

8.6.4.2.27 getStandardDigitalOutputs()

uint32_t arcs::common_interface::IoControl::getStandardDigitalOutputs ()

Get all standard digital output values.

Returns

All standard digital output values.

For example, if the return value is 2863267846, its binary representation is 1010101010101000000000000000110.

The lower 16 bits represent the status of all standard digital outputs, the least significant bit indicates the output status of DO00, the second least significant bit indicates DO01, and so on.

1 means high level, 0 means low level.

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
getStandardDigitalOutputs(self: pyaubo_sdk.IoControl) -> int
```

Lua function prototype

```
getStandardDigitalOutputs() -> number
```

Lua example

```
num = getStandardDigitalOutputs()
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.IoControl.getStandardDigitalOutputs","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":69}
```

8.6.4.2.28 getStaticLinkInputNum()

```
int arcs::common_interface::IoControl::getStaticLinkInputNum ()
```

Get the number of static link inputs.

Returns

Number of static link inputs.

Exceptions

<code>arcs::common_interface::AuboException</code>	
--	--

Python function prototype

```
getStaticLinkInputNum(self: pyaubo_sdk.IoControl) -> int
```

Lua function prototype

```
getStaticLinkInputNum() -> number
```

Lua example

```
num = getStaticLinkInputNum()
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.IoControl.getStaticLinkInputNum","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":8}
```

8.6.4.2.29 getStaticLinkInputs()

```
uint32_t arcs::common_interface::IoControl::getStaticLinkInputs ()
```

Get all static link input values.

Returns

All static link input values.

For example, if the return value is 2863267846, its binary representation is 1010101010101000000000000000110.

The lower 16 bits represent the status of all static link inputs, the least significant bit indicates the input status of pin 0, the second least significant bit indicates pin 1, and so on.

1 means high level, 0 means low level.

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
getStaticLinkInputs(self: pyaubo_sdk.IoControl) -> int
```

Lua function prototype

```
getStaticLinkInputs() -> number
```

Lua example

```
num = getStaticLinkInputs()
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.IoControl.getStaticLinkInputs","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":0}
```

8.6.4.2.30 getStaticLinkOutputNum()

```
int arcs::common_interface::IoControl::getStaticLinkOutputNum ()
```

Get the number of static link outputs.

Returns

Number of static link outputs.

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
getStaticLinkOutputNum(self: pyaubo_sdk.IoControl) -> int
```

Lua function prototype

```
getStaticLinkOutputNum() -> number
```

Lua example

```
num = getStaticLinkOutputNum()
```

JSON-RPC request example

```
{"jsonrpc": "2.0", "method": "rob1.IoControl.getStaticLinkOutputNum", "params": [], "id": 1}
```

JSON-RPC response example

```
{"id": 1, "jsonrpc": "2.0", "result": 0}
```

8.6.4.2.31 getStaticLinkOutputs()

```
uint32_t arcs::common_interface::IoControl::getStaticLinkOutputs ()
```

Get all static link output values.

Returns

Returns all static link output values.

For example, if the return value is 2863267846, its binary representation is 1010101010101000000000000000110.

The lower 16 bits represent the status of all static link outputs, the least significant bit indicates the output status of pin 0, the second least significant bit indicates pin 1, and so on.

1 means high level, 0 means low level.

Exceptions

arcs::common_interface::AuboException	
---------------------------------------	--

Python function prototype

```
getStaticLinkOutputs(self: pyaubo_sdk.IoControl) -> int
```

Lua function prototype

```
getStaticLinkOutputs() -> number
```

Lua example

```
num = getStaticLinkOutputs()
```

JSON-RPC request example

```
{"jsonrpc": "2.0", "method": "rob1.IoControl.getStaticLinkOutputs", "params": [], "id": 1}
```

JSON-RPC response example

```
{"id": 1, "jsonrpc": "2.0", "result": 0}
```

8.6.4.2.32 `getToolAnalogInput()`

```
double arcs::common_interface::IoControl::getToolAnalogInput (
    int index)
```

Get the value of tool analog input.

Parameters

index	Indicates the IO pin, starting from 0. For example, 0 means the first pin.
-------	--

Returns

Tool analog input value.

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
getToolAnalogInput(self: pyaubo_sdk.IoControl, arg0: int) -> float
```

Lua function prototype

```
getToolAnalogInput(index: number) -> number
```

Lua example

```
num = getToolAnalogInput(0)
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.IoControl.getToolAnalogInput","params":[0],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":0.0}
```

8.6.4.2.33 `getToolAnalogInputDomain()`

```
int arcs::common_interface::IoControl::getToolAnalogInputDomain (
    int index)
```

Get the domain of tool analog input.

Parameters

index	Indicates the IO pin, starting from 0. For example, 0 means the first pin.
-------	--

Returns

Tool analog input domain

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
getToolAnalogInputDomain(self: pyaubo_sdk.IoControl, arg0: int) -> int
```

Lua function prototype

```
getToolAnalogInputDomain(index: number) -> number
```

Lua example

```
num = getToolAnalogInputDomain(0)
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.IoControl.getToolAnalogInputDomain","params":[0],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":10}
```

8.6.4.2.34 getToolAnalogInputNum()

```
int arcs::common_interface::IoControl::getToolAnalogInputNum ()
```

Get the number of tool analog inputs.

Returns

Number of tool analog inputs.

Exceptions

<code>arcs::common_interface::AuboException</code>	
--	--

Python function prototype

```
getToolAnalogInputNum(self: pyaubo_sdk.IoControl) -> int
```

Lua function prototype

```
getToolAnalogInputNum() -> number
```

Lua example

```
num = getToolAnalogInputNum()
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.IoControl.getToolAnalogInputNum","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":2}
```

8.6.4.2.35 getToolAnalogOutput()

```
double arcs::common_interface::IoControl::getToolAnalogOutput (
    int index)
```

Get the value of tool analog output.

Parameters

index	Indicates the IO pin, starting from 0. For example, 0 means the first pin.
-------	--

Returns

Tool analog output value.

Exceptions

<code>arcs::common_interface::AuboException</code>	
--	--

Python function prototype

```
getToolAnalogOutput(self: pyaubo_sdk.IoControl, arg0: int) -> float
```

Lua function prototype

```
getToolAnalogOutput(index: number) -> number
```

Lua example

```
num = getToolAnalogOutput(0)
```

JSON-RPC request example

```
{"jsonrpc": "2.0", "method": "rob1.IoControl.getToolAnalogOutput", "params": [0], "id": 1}
```

JSON-RPC response example

```
{"id": 1, "jsonrpc": "2.0", "result": 0.0}
```

8.6.4.2.36 getToolAnalogOutputDomain()

```
int arcs::common_interface::IoControl::getToolAnalogOutputDomain (  
    int index)
```

Get the domain of tool analog output.

Parameters

index	Indicates the IO pin, starting from 0. For example, 0 means the first pin.
-------	--

Returns

Tool analog output domain

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
getToolAnalogOutputDomain(self: pyaubo_sdk.IoControl, arg0: int) -> int
```

Lua function prototype

```
getToolAnalogOutputDomain(index: number) -> number
```

Lua example

```
num = getToolAnalogOutputDomain(0)
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.IoControl.getToolAnalogOutputDomain","params":[0,"id":1]}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":0}
```

8.6.4.2.37 getToolAnalogOutputNum()

```
int arcs::common_interface::IoControl::getToolAnalogOutputNum ()
```

Get the number of tool analog outputs.

Returns

Number of tool analog outputs.

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
getToolAnalogOutputNum(self: pyaubo_sdk.IoControl) -> int
```

Lua function prototype

```
getToolAnalogOutputNum() -> number
```

Lua example

```
num = getToolAnalogOutputNum()
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.IoControl.getToolAnalogOutputNum","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":0}
```

8.6.4.2.38 getToolAnalogOutputRunstate()

[StandardOutputRunState](#) arcs::common_interface::IoControl::getToolAnalogOutputRunstate (int index)

Get the runstate for tool analog output.

Parameters

index	Indicates the IO pin, starting from 0. For example, 0 means the first pin.
-------	--

Returns

Tool analog output runstate selection

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
getToolAnalogOutputRunstate(self: pyaubo_sdk.IoControl, arg0: int) -> arcs::common\_interface::StandardOutputRunState
```

Lua function prototype

```
getToolAnalogOutputRunstate(index: number) -> number
```

Lua example

```
num = getToolAnalogOutputRunstate(0)
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.IoControl.getToolAnalogOutputRunstate","params":[0],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":"None"}
```

8.6.4.2.39 `getToolButtonStatus()`

```
bool arcs::common_interface::IoControl::getToolButtonStatus ()
```

Get the status of the tool button.

Returns

Returns true if pressed; otherwise false.

Exceptions

<code>arcs::common_interface::AuboException</code>	
--	--

Python function prototype

```
getToolButtonStatus() -> bool
```

Lua function prototype

```
getToolButtonStatus() -> boolean
```

Lua example

```
status = getToolButtonStatus()
```

JSON-RPC request example

```
{ "jsonrpc": "2.0", "method": "rob1.IoControl.getToolButtonStatus", "params": [], "id": 1 }
```

JSON-RPC response example

```
{ "id": 1, "jsonrpc": "2.0", "result": false }
```

8.6.4.2.40 `getToolDigitalInput()`

```
bool arcs::common_interface::IoControl::getToolDigitalInput (
    int index)
```

Get the value of tool digital input.

Parameters

index	Indicates the IO pin, starting from 0. For example, 0 means the first pin.
-------	--

Returns

Returns true for high level; false for low level.

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
getToolDigitalInput(self: pyaubo_sdk.IoControl, arg0: int) -> bool
```

Lua function prototype

```
getToolDigitalInput(index: number) -> boolean
```

Lua example

```
status = getToolDigitalInput(0)
```

JSON-RPC request example

```
{ "jsonrpc": "2.0", "method": "rob1.IoControl.getToolDigitalInput", "params": [0], "id": 1 }
```

JSON-RPC response example

```
{ "id": 1, "jsonrpc": "2.0", "result": false }
```

8.6.4.2.41 getToolDigitalInputAction()

[StandardInputAction](#) arcs::common_interface::IoControl::getToolDigitalInputAction (
 int index)

Get the trigger action for tool digital input.

Parameters

index	Indicates the IO pin, starting from 0. For example, 0 means the first pin.
-------	--

Returns

Tool digital input trigger action

Python function prototype

```
getToolDigitalInputAction(self: pyaubo_sdk.IoControl, arg0: int) -> arcs::common\_interface::StandardInputAction
```

Lua function prototype

```
getToolDigitalInputAction(index: number) -> number
```

Lua example

```
getToolDigitalInputAction(0)
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.IoControl.getToolDigitalInputAction","params":[0],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":"Default"}
```

8.6.4.2.42 getToolDigitalInputNum()

```
int arcs::common_interface::IoControl::getToolDigitalInputNum ()
```

Get the number of tool digital IOs (including digital inputs and outputs).

Returns

Number of tool digital IOs (including digital inputs and outputs).

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
getToolDigitalInputNum(self: pyaubo_sdk.IoControl) -> int
```

Lua function prototype

```
getToolDigitalInputNum() -> number
```

Lua example

```
num = getStandardDigitalInputNum()
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.IoControl.getToolDigitalInputNum","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":4}
```

8.6.4.2.43 getToolDigitalInputs()

uint32_t arcs::common_interface::IoControl::getToolDigitalInputs ()

Get all tool digital input values.

Returns

Returns all tool digital input values.

Exceptions

arcs::common_interface::AuboException	
---------------------------------------	--

Python function prototype

```
getToolDigitalInputs(self: pyaubo_sdk.IoControl) -> int
```

Lua function prototype

```
getToolDigitalInputs() -> number
```

Lua example

```
num = getToolDigitalInputs()
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.IoControl.getToolDigitalInputs","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":0}
```

8.6.4.2.44 `getToolDigitalOutput()`

```
bool arcs::common_interface::IoControl::getToolDigitalOutput (
    int index)
```

Get the value of tool digital output.

Parameters

index	Indicates the IO pin, starting from 0. For example, 0 means the first pin.
-------	--

Returns

Returns true for high level; false for low level.

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
getToolDigitalOutput(self: pyaubo_sdk.IoControl, arg0: int) -> bool
```

Lua function prototype

```
getToolDigitalOutput(index: number) -> boolean
```

Lua example

```
status = getToolDigitalOutput(0)
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.IoControl.getToolDigitalOutput","params":[0,"id":1]}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":false}
```

8.6.4.2.45 `getToolDigitalOutputNum()`

```
int arcs::common_interface::IoControl::getToolDigitalOutputNum ()
```

Get the number of tool digital IOs (including digital inputs and outputs).

Exceptions

<code>arcs::common_interface::AuboException</code>	
--	--

Python function prototype

```
getToolDigitalOutputNum(self: pyaubo_sdk.IoControl) -> int
```

Lua function prototype

```
getToolDigitalOutputNum() -> number
```

Lua example

```
num = getToolDigitalOutputNum()
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.IoControl.getToolDigitalOutputNum","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":4}
```

8.6.4.2.46 getToolDigitalOutputRunstate()

`StandardOutputRunState` `arcs::common_interface::IoControl::getToolDigitalOutputRunstate (`
`int index)`

Get the runstate for tool digital output.

Parameters

index	Indicates the IO pin, starting from 0. For example, 0 means the first pin.
-------	--

Returns

Output runstate selection

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
getToolDigitalOutputRunstate(self: pyaubo_sdk.IoControl, arg0: int) -> arcs::common_interface::StandardOutput
```

Lua function prototype

```
getToolDigitalOutputRunstate(index: number) -> number
```

Lua example

```
num = getToolDigitalOutputRunstate(0)
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.IoControl.getToolDigitalOutputRunstate","params":[0],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":"None"}
```

8.6.4.2.47 getToolDigitalOutputs()

```
uint32_t arcs::common_interface::IoControl::getToolDigitalOutputs ()
```

Get all tool digital output values.

Returns

All tool digital output values.

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
getToolDigitalOutputs(self: pyaubo_sdk.IoControl) -> int
```

Lua function prototype

```
getToolDigitalOutputs() -> number
```

Lua example

```
num = getToolDigitalOutputs()
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.IoControl.getToolDigitalOutputs","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":9}
```

8.6.4.2.48 getToolVoltageOutputDomain()

```
int arcs::common_interface::IoControl::getToolVoltageOutputDomain ()
```

Get the tool voltage output value (unit: V)

Returns

Tool voltage output value (unit: V)

Exceptions

arcs::common_interface::AuboException	
---------------------------------------	--

Python function prototype

```
getToolVoltageOutputDomain(self: pyaubo_sdk.IoControl) -> int
```

Lua function prototype

```
getToolVoltageOutputDomain() -> number
```

Lua example

```
num = getToolVoltageOutputDomain()
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.IoControl.getToolVoltageOutputDomain","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":0}
```

8.6.4.2.49 hasEncoderSensor()

```
bool arcs::common_interface::IoControl::hasEncoderSensor ()
```

Whether the robot is equipped with an encoder.

The integrated encoder number is 0.

Returns

Returns true if the robot is equipped with an encoder, otherwise false.

Exceptions

arcs::common_interface::AuboException	
---	--

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.IoControl.hasEncoderSensor","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":true}
```

8.6.4.2.50 isToolIoInput()

```
bool arcs::common_interface::IoControl::isToolIoInput (
    int index)
```

Determine whether the specified tool digital IO is configured as input.

Parameters

index	Indicates the IO pin, starting from 0. For example, 0 means the first pin.
-------	--

Returns

Returns true if the specified IO is input, otherwise false.

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
isToolIoInput(self: pyaubo_sdk.IoControl, arg0: int) -> bool
```

Lua function prototype

```
isToolIoInput(index: number) -> boolean
```

Lua example

```
status = isToolIoInput(0)
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.IoControl.isToolIoInput","params":[0],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":true}
```

8.6.4.2.51 setAnalogOutputAfterEStopDefault()

```
int arcs::common_interface::IoControl::setAnalogOutputAfterEStopDefault ()
```

Set all analog output states after emergency stop to default(no change)

Returns

Returns 0 on success; error code on failure. AUBO_BUSY AUBO_BAD_STATE -AUBO_BAD↵
_STATE

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
setAnalogOutputAfterEStopDefault(self: pyaubo_sdk.IoControl) -> int
```

Lua function prototype

```
setAnalogOutputAfterEStopDefault() -> nil
```

Lua example

```
setAnalogOutputAfterEStopDefault()
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.IoControl.setAnalogOutputAfterEStopDefault","params":[↵
,"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":0}
```

8.6.4.2.52 setConfigurableDigitalInputAction()

```
int arcs::common_interface::IoControl::setConfigurableDigitalInputAction (
    int index,
    StandardInputAction action)
```

Set the trigger action for configurable digital input.

Parameters

index	Indicates the IO pin, starting from 0. For example, 0 means the first pin.
action	Trigger action

Returns

Returns 0 on success; error code on failure. AUBO_REQUEST_IGNORE AUBO_BUSY AUBO↵
_BAD_STATE -AUBO_INVL_ARGUMENT -AUBO_BAD_STATE

Exceptions

arcs::common_interface::AuboException	
---------------------------------------	--

Note

This function takes effect only when the safety input action of the configurable input is set to SafetyInputAction::Unassigned.

Python function prototype

```
setConfigurableDigitalInputAction(self: pyaubo_sdk.IoControl, arg0: int, arg1: arcs::common\_interface::StandardI
-> int
```

Lua function prototype

```
setConfigurableDigitalInputAction(index: number, action: number) -> nil
```

Lua example

```
setConfigurableDigitalInputAction(0,1)
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.IoControl.setConfigurableDigitalInputAction","params":[0,"↵
Handguide"],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":0}
```

8.6.4.2.53 setConfigurableDigitalOutput()

```
int arcs::common_interface::IoControl::setConfigurableDigitalOutput (
    int index,
    bool value)
```

Set the value of configurable digital output.

Parameters

index	Indicates the IO pin, starting from 0. For example, 0 means the first pin.
value	Output value.

Returns

Returns 0 on success; error code on failure. AUBO_REQUEST_IGNORE AUBO_BUSY AUBO↵
_BAD_STATE -AUBO_INVL_ARGUMENT -AUBO_BAD_STATE

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
setConfigurableDigitalOutput(self: pyaubo_sdk.IoControl, arg0: int, arg1: bool) -> int
```

Lua function prototype

```
setConfigurableDigitalOutput(index: number, value: boolean) -> nil
```

Lua example

```
setConfigurableDigitalOutput(0,true)
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.IoControl.setConfigurableDigitalOutput","params":[0,true],
"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":0}
```

8.6.4.2.54 setConfigurableDigitalOutputAfterEStop()

```
int arcs::common_interface::IoControl::setConfigurableDigitalOutputAfterEStop (
    int index,
    bool value)
```

Set the value of a configurable digital output after emergency stop

Parameters

index	Indicates the IO pin.
value	Output value.

Returns

Returns 0 on success; error code on failure. AUBO_REQUEST_IGNORE AUBO_BUSY AUBO_BAD_STATE -AUBO_INVL_ARGUMENT -AUBO_BAD_STATE

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
setConfigurableDigitalOutputAfterEStop(self: pyaubo_sdk.IoControl, arg0: int, arg1: bool) -> int
```

Lua function prototype

```
setConfigurableDigitalOutputAfterEStop(index: number, value: boolean) -> nil
```

Lua example

```
setConfigurableDigitalOutputAfterEStop(0,true)
```

JSON-RPC request example

```
{ "jsonrpc": "2.0", "method": "rob1.IoControl.setConfigurableDigitalOutputAfterEStop", "params": [0,true], "id": 1 }
```

JSON-RPC response example

```
{ "id": 1, "jsonrpc": "2.0", "result": 0 }
```

8.6.4.2.55 setConfigurableDigitalOutputPulse()

```
int arcs::common_interface::IoControl::setConfigurableDigitalOutputPulse (
    int index,
    bool value,
    double duration)
```

Set configurable digital output pulse.

Parameters

index	
value	
duration	

Returns

Returns 0 on success; error code on failure. AUBO_REQUEST_IGNORE AUBO_BUSY AUBO_BAD_STATE -AUBO_INVL_ARGUMENT -AUBO_BAD_STATE

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
setConfigurableDigitalOutputPulse(self: pyaubo_sdk.IoControl, arg0: int, arg1: bool, arg2: float)
-> int
```

Lua function prototype

```
setConfigurableDigitalOutputPulse(index: number, value: boolean, duration: number) -> nil
```

Lua example

```
setConfigurableDigitalOutputPulse(0,true,2.3)
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.IoControl.setConfigurableDigitalOutputPulse","params":↵
[0,true,0.5],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":0}
```

8.6.4.2.56 setConfigurableDigitalOutputRunstate()

```
int arcs::common_interface::IoControl::setConfigurableDigitalOutputRunstate (
    int index,
    StandardOutputRunState runstate)
```

Set the runstate for configurable digital output.

Parameters

index	Indicates the IO pin, starting from 0. For example, 0 means the first pin.
runstate	Output runstate selection

Returns

Returns 0 on success; error code on failure. AUBO_REQUEST_IGNORE AUBO_BUSY AUBO↵
BAD_STATE -AUBO_INVL_ARGUMENT -AUBO_BAD_STATE

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
setConfigurableDigitalOutputRunstate(self: pyaubo_sdk.IoControl, arg0: int, arg1: arcs::common_interface::Stand
-> int
```

Lua function prototype

```
setConfigurableDigitalOutputRunstate(index: number, runstate: number) -> nil
```

Lua example

```
setConfigurableDigitalOutputRunstate(0,1)
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.IoControl.setConfigurableDigitalOutputRunstate","params":↵
[0,"None"],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":0}
```

8.6.4.2.57 setDigitalInputActionDefault()

```
int arcs::common_interface::IoControl::setDigitalInputActionDefault ()
```

Set all digital input actions to no trigger.

Note

When the input action is set to no trigger, setting the digital input value to high will not trigger any robot action.

Returns

Returns 0 on success; error code on failure. AUBO_BUSY AUBO_BAD_STATE -AUBO_BAD↵
_STATE

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
setDigitalInputActionDefault(self: pyaubo_sdk.IoControl) -> int
```

Lua function prototype

```
setDigitalInputActionDefault() -> nil
```

Lua example

```
setDigitalInputActionDefault()
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.IoControl.setDigitalInputActionDefault","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":0}
```

8.6.4.2.58 setDigitalOutputAfterEStopDefault()

```
int arcs::common_interface::IoControl::setDigitalOutputAfterEStopDefault ()
```

Set all digital output states after emergency stop to default(no change)

Returns

Returns 0 on success; error code on failure. AUBO_BUSY AUBO_BAD_STATE -AUBO_BAD↵
_STATE

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
setDigitalOutputAfterEStopDefault(self: pyaubo_sdk.IoControl) -> int
```

Lua function prototype

```
setDigitalOutputAfterEStopDefault() -> nil
```

Lua example

```
setDigitalOutputAfterEStopDefault()
```

JSON-RPC request example

```
{ "jsonrpc": "2.0", "method": "rob1.IoControl.setDigitalOutputAfterEStopDefault", "params": [], "id": 1 }
```

JSON-RPC response example

```
{ "id": 1, "jsonrpc": "2.0", "result": 0 }
```

8.6.4.2.59 setDigitalOutputRunstateDefault()

```
int arcs::common_interface::IoControl::setDigitalOutputRunstateDefault ( )
```

Set all digital output runstates to None.

Note

When the output runstate is set to None (StandardOutputRunState::None), users can set the digital output value.

When the output runstate is set, users cannot set the digital output value, and the controller will set it automatically.

For example, when DO0's output runstate is set to indicate hand-guiding (StandardOutputRunState::Handguiding), the robot enters hand-guiding mode and DO0 will automatically become high. When the robot exits hand-guiding, DO0 will automatically become low.

Returns

Returns 0 on success; error code on failure. AUBO_BUSY AUBO_BAD_STATE -AUBO_BAD_STATE

Exceptions

arcs::common_interface::AuboException	
---------------------------------------	--

Python function prototype

```
setDigitalOutputRunstateDefault(self: pyaubo_sdk.IoControl) -> int
```

Lua function prototype

```
setDigitalOutputRunstateDefault() -> nil
```

Lua example

```
setDigitalOutputRunstateDefault()
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.IoControl.setDigitalOutputRunstateDefault","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":0}
```

8.6.4.2.60 setEncDecoderType()

```
int arcs::common_interface::IoControl::setEncDecoderType (
    int type,
    int range_id)
```

Set the decoding method of the integrated encoder.

Parameters

type	0 - Disable encoder 1 - AB quadrature 2 - AB quadrature + Z 3 - AB differential quadrature 4 - AB differential quadrature + Z differential
range_id	0 is a 32-bit signed encoder, range [-2147483648, 2147483647] 1 is an 8-bit unsigned encoder, range [0, 255] 2 is a 16-bit unsigned encoder, range [0, 65535] 3 is a 24-bit unsigned encoder, range [0, 16777215] 4 is a 32-bit unsigned encoder, range [0, 4294967295]

Returns

Returns 0 on success; error code on failure. AUBO_NO_ACCESS AUBO_BUSY AUBO_BAD↵
_STATE -AUBO_INVL_ARGUMENT -AUBO_BAD_STATE

Exceptions

arcs::common_interface::AuboException

8.6.4.2.61 setEncTickCount()

Generated by Doxygen

```
int arcs::common_interface::IoControl::setEncTickCount (
    int tick)
```

Parameters

tick	Tick count
------	------------

Returns

Returns 0 on success; error code on failure. AUBO_NO_ACCESS AUBO_BUSY AUBO_BAD↵
_STATE -AUBO_INVL_ARGUMENT -AUBO_BAD_STATE

Exceptions

arcs::common_interface::AuboException	
---	--

8.6.4.2.62 setStandardAnalogInputDomain()

```
int arcs::common_interface::IoControl::setStandardAnalogInputDomain (
    int index,
    int domain)
```

Set the range of standard analog input.

Parameters

index	Indicates the IO pin, starting from 0. For example, 0 means the first pin.
domain	Input range

Returns

Returns 0 on success; error code on failure. AUBO_REQUEST_IGNORE AUBO_BUSY AUBO↵
_BAD_STATE -AUBO_INVL_ARGUMENT -AUBO_BAD_STATE

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
setStandardAnalogInputDomain(self: pyaubo_sdk.IoControl, arg0: int, arg1: int) -> int
```

Lua function prototype

```
setStandardAnalogInputDomain(index: number, domain: number) -> nil
```

Lua example

```
setStandardAnalogInputDomain(0,1)
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.IoControl.setStandardAnalogInputDomain","params":[0,8],  
"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":0}
```

8.6.4.2.63 setStandardAnalogOutput()

```
int arcs::common_interface::IoControl::setStandardAnalogOutput (  
    int index,  
    double value)
```

Set the value of standard analog output.

Parameters

index	Indicates the IO pin, starting from 0. For example, 0 means the first pin.
value	Output value.

Returns

Returns 0 on success; error code on failure. AUBO_REQUEST_IGNORE AUBO_BUSY AUBO_↵
_BAD_STATE -AUBO_INVL_ARGUMENT -AUBO_BAD_STATE

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
setStandardAnalogOutput(self: pyaubo_sdk.IoControl, arg0: int, arg1: float) -> int
```

Lua function prototype

```
setStandardAnalogOutput(index: number, value: number) -> nil
```

Lua example

```
setStandardAnalogOutput(0,5.4)
```


8.6.4.2.64 setStandardAnalogOutputAfterEStop()

```
int arcs::common_interface::IoControl::setStandardAnalogOutputAfterEStop (
    int index,
    double value)
```

Set the value of standard analog output after emergency stop

Parameters

index	Indicates the IO pin, starting from 0. For example, 0 means the first pin.
value	Output value.

Returns

Returns 0 on success; error code on failure. AUBO_REQUEST_IGNORE AUBO_BUSY AUBO↵
_BAD_STATE -AUBO_INVL_ARGUMENT -AUBO_BAD_STATE

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
setStandardAnalogOutputAfterEStop(self: pyaubo_sdk.IoControl, arg0: int, arg1: float) -> int
```

Lua function prototype

```
setStandardAnalogOutputAfterEStop(index: number, value: number) -> nil
```

Lua example

```
setStandardAnalogOutputAfterEStop(0,5.4)
```

8.6.4.2.65 setStandardAnalogOutputDomain()

```
int arcs::common_interface::IoControl::setStandardAnalogOutputDomain (
    int index,
    int domain)
```

Set the range of standard analog output.

Parameters

index	Indicates the IO pin, starting from 0. For example, 0 means the first pin.
domain	Output range

Returns

Returns 0 on success; error code on failure. AUBO_REQUEST_IGNORE AUBO_BUSY AUBO_BAD_STATE -AUBO_INVL_ARGUMENT -AUBO_BAD_STATE

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
setStandardAnalogOutputDomain(self: pyaubo_sdk.IoControl, arg0: int, arg1: int) -> int
```

Lua function prototype

```
setStandardAnalogOutputDomain(index: number, domain: number) -> nil
```

Lua example

```
setStandardAnalogOutputDomain(0,1)
```

JSON-RPC request example

```
{"jsonrpc": "2.0", "method": "rob1.IoControl.setStandardAnalogOutputDomain", "params": [0,8], "id": 1}
```

JSON-RPC response example

```
{"id": 1, "jsonrpc": "2.0", "result": 0}
```

8.6.4.2.66 setStandardAnalogOutputRunstate()

```
int arcs::common_interface::IoControl::setStandardAnalogOutputRunstate (
    int index,
    StandardOutputRunState runstate)
```

Set the runstate for standard analog output.

Note

When the output runstate is set to None (StandardOutputRunState::None), users can set the analog

Parameters

index	Indicates the IO pin, starting from 0. For example, 0 means the first pin.
runstate	Output runstate selection

Returns

Returns 0 on success; error code on failure. AUBO_REQUEST_IGNORE AUBO_BUSY AUBO_BAD_STATE -AUBO_INVL_ARGUMENT -AUBO_BAD_STATE

Exceptions

arcs::common_interface::AuboException

Python function prototype

```
setStandardAnalogOutputRunstate(self: pyaubo_sdk.IoControl, arg0: int, arg1: arcs::common\_interface::StandardAnalogOutputRunstate
-> int
```

Lua function prototype

```
setStandardAnalogOutputRunstate(index: number, runstate: number) -> nil
```

Lua example

```
setStandardAnalogOutputRunstate(0,6)
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.IoControl.setStandardAnalogOutputRunstate","params":[0,"None"],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":0}
```

8.6.4.2.67 setStandardDigitalInputAction()

```
int arcs::common_interface::IoControl::setStandardDigitalInputAction (
    int index,
    StandardInputAction action)
```

Set the trigger action for standard digital input.

Note

When the input is set to no trigger action ([StandardInputAction::Default](#)), setting the digital input

Parameters

index	Indicates the IO pin, starting from 0. For example, 0 means the first pin.
action	Trigger action

Returns

Returns 0 on success; error code on failure. AUBO_REQUEST_IGNORE AUBO_BUSY AUBO_BAD_STATE -AUBO_INVL_ARGUMENT -AUBO_BAD_STATE

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
setStandardDigitalInputAction(self: pyaubo_sdk.IoControl, arg0: int, arg1: arcs::common\_interface::StandardInput
-> int
```

Lua function prototype

```
setStandardDigitalInputAction(index: number, action: number) -> nil
```

Lua example

```
setStandardDigitalInputAction(0,1)
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.IoControl.setStandardDigitalInputAction","params":[0,"Handguide"]↵
,"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":0}
```

8.6.4.2.68 setStandardDigitalOutput()

```
int arcs::common_interface::IoControl::setStandardDigitalOutput (
    int index,
    bool value)
```

Set the value of a standard digital output.

Parameters

index	Indicates the IO pin.
value	Output value.

Returns

Returns 0 on success; error code on failure. AUBO_REQUEST_IGNORE AUBO_BUSY AUBO_↵
_BAD_STATE -AUBO_INVL_ARGUMENT -AUBO_BAD_STATE

Exceptions

<code>arcs::common_interface::AuboException</code>	
--	--

Python function prototype

```
setStandardDigitalOutput(self: pyaubo_sdk.IoControl, arg0: int, arg1: bool) -> int
```

Lua function prototype

```
setStandardDigitalOutput(index: number, value: boolean) -> nil
```

Lua example

```
setStandardDigitalOutput(0,true)
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.IoControl.setStandardDigitalOutput","params":[0,true],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":0}
```

8.6.4.2.69 setStandardDigitalOutputAfterEStop()

```
int arcs::common_interface::IoControl::setStandardDigitalOutputAfterEStop (
    int index,
    bool value)
```

Set the value of a standard digital output after emergency stop

Parameters

index	Indicates the IO pin.
value	Output value.

Returns

Returns 0 on success; error code on failure. AUBO_REQUEST_IGNORE AUBO_BUSY AUBO_BAD_STATE -AUBO_INVL_ARGUMENT -AUBO_BAD_STATE

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
setStandardDigitalOutputAfterEStop(self: pyaubo_sdk.IoControl, arg0: int, arg1: bool) -> int
```

Lua function prototype

```
setStandardDigitalOutputAfterEStop(index: number, value: boolean) -> nil
```

Lua example

```
setStandardDigitalOutputAfterEStop(0,true)
```

JSON-RPC request example

```
{ "jsonrpc": "2.0", "method": "rob1.IoControl.setStandardDigitalOutputAfterEStop", "params": [0,true], "id": 1 }
```

JSON-RPC response example

```
{ "id": 1, "jsonrpc": "2.0", "result": 0 }
```

8.6.4.2.70 setStandardDigitalOutputPulse()

```
int arcs::common_interface::IoControl::setStandardDigitalOutputPulse (
    int index,
    bool value,
    double duration)
```

Parameters

index	
value	
duration	

Returns

Returns 0 on success; error code on failure. AUBO_REQUEST_IGNORE AUBO_BUSY AUBO_BAD_STATE -AUBO_INVL_ARGUMENT -AUBO_BAD_STATE

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
setStandardDigitalOutputPulse(self: pyaubo_sdk.IoControl, arg0: int, arg1: bool, arg2: float) ->
int
```

Lua function prototype

```
setStandardDigitalOutputPulse(index: number, value: boolean, duration: number) -> nil
```

Lua example

```
setStandardDigitalOutputPulse(0,true,2.5)
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.IoControl.setStandardDigitalOutputPulse","params":[0,true,0.5],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":0}
```

8.6.4.2.71 setStandardDigitalOutputRunstate()

```
int arcs::common_interface::IoControl::setStandardDigitalOutputRunstate (
    int index,
    StandardOutputRunState runstate)
```

Generated by Doxygen

Set the runstate for standard digital output.

Note

Parameters

index	Indicates the IO pin, starting from 0. For example, 0 means the first pin.
runstate	Output runstate selection

Returns

Returns 0 on success; error code on failure. AUBO_REQUEST_IGNORE AUBO_BUSY AUBO_BAD_STATE -AUBO_INVL_ARGUMENT -AUBO_BAD_STATE

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
setStandardDigitalOutputRunstate(self: pyaubo_sdk.IoControl, arg0: int, arg1: arcs::common\_interface::StandardDigitalOutputRunstate
-> int
```

Lua function prototype

```
setStandardDigitalOutputRunstate(index: number, runstate: number) -> nil
```

Lua example

```
setStandardDigitalOutputRunstate(0,1)
```

JSON-RPC request example

```
{ "jsonrpc": "2.0", "method": "rob1.IoControl.setStandardDigitalOutputRunstate", "params": [0, "PowerOn"], "id": 1 }
```

JSON-RPC response example

```
{ "id": 1, "jsonrpc": "2.0", "result": 0 }
```

8.6.4.2.72 setToolAnalogInputDomain()

```
int arcs::common_interface::IoControl::setToolAnalogInputDomain (
    int index,
    int domain)
```

Set the range of tool analog input.

Parameters

index	Indicates the IO pin, starting from 0. For example, 0 means the first pin.
domain	Input range

Returns

Returns 0 on success; error code on failure. AUBO_REQUEST_IGNORE AUBO_BUSY AUBO_BAD_STATE -AUBO_INVL_ARGUMENT -AUBO_BAD_STATE

Exceptions

<code>arcs::common_interface::AuboException</code>	
--	--

Python function prototype

```
setToolAnalogInputDomain(self: pyaubo_sdk.IoControl, arg0: int, arg1: int) -> int
```

Lua function prototype

```
setToolAnalogInputDomain(index: number, domain: number) -> nil
```

Lua example

```
setToolAnalogInputDomain(0,1)
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.IoControl.setToolAnalogInputDomain","params":[0,8],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":0}
```

8.6.4.2.73 setToolAnalogOutput()

```
int arcs::common_interface::IoControl::setToolAnalogOutput (
    int index,
    double value)
```

Set the value of tool analog output.

Parameters

index	Indicates the IO pin, starting from 0. For example, 0 means the first pin.
value	Output value.

Returns

Returns 0 on success; error code on failure. AUBO_REQUEST_IGNORE AUBO_BUSY AUBO_↵
_BAD_STATE -AUBO_INVL_ARGUMENT -AUBO_BAD_STATE

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
setToolAnalogOutput(self: pyaubo_sdk.IoControl, arg0: int, arg1: float) -> int
```

Lua function prototype

```
setToolAnalogOutput(index: number, value: number) -> nil
```

Lua example

```
setToolAnalogOutput(0,1.2)
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.IoControl.setToolAnalogOutput","params":[0,0.5],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":13}
```

8.6.4.2.74 setToolAnalogOutputDomain()

```
int arcs::common_interface::IoControl::setToolAnalogOutputDomain (
    int index,
    int domain)
```

Set the range of tool analog output.

Parameters

index	Indicates the IO pin, starting from 0. For example, 0 means the first pin.
domain	Output range

Returns

Returns 0 on success; error code on failure. AUBO_REQUEST_IGNORE AUBO_BUSY AUBO_BAD_STATE -AUBO_INVL_ARGUMENT -AUBO_BAD_STATE

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
setToolAnalogOutputDomain(self: pyaubo_sdk.IoControl, arg0: int, arg1: int) -> int
```

Lua function prototype

```
setToolAnalogOutputDomain(index: number, domain: number) -> nil
```

Lua example

```
setToolAnalogOutputDomain(0,1)
```

8.6.4.2.75 setToolAnalogOutputRunstate()

```
int arcs::common_interface::IoControl::setToolAnalogOutputRunstate (
    int index,
    StandardOutputRunState runstate)
```

Set the runstate for tool analog output.

Parameters

index	Indicates the IO pin, starting from 0. For example, 0 means the first pin.
runstate	Output runstate selection

Returns 0 on success; error code on failure. AUBO_REQUEST_IGNORE AUBO_BUSY AUBO_BAD_STATE -AUBO_INVL_ARGUMENT -AUBO_BAD_STATE

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
setToolAnalogOutputRunstate(self: pyaubo_sdk.IoControl, arg0: int, arg1: arcs::common\_interface::StandardOutput
-> int
```

Lua function prototype

```
setToolAnalogOutputRunstate(index: number, runstate: number) -> nil
```

Lua example

```
setToolAnalogOutputRunstate(0,1)
```

JSON-RPC request example

```
{ "jsonrpc": "2.0", "method": "rob1.IoControl.setToolAnalogOutputRunstate", "params": [0, "None"],
  "id": 1 }
```

JSON-RPC response example

```
{ "id": 1, "jsonrpc": "2.0", "result": 0 }
```

8.6.4.2.76 setToolDigitalInputAction()

```
int arcs::common_interface::IoControl::setToolDigitalInputAction (
    int index,
    StandardInputAction action)
```

Set the trigger action for tool digital input.

Note

When the input is set to no trigger action (`StandardInputAction::Default`), setting the tool digital input value to high will not trigger any robot action.

When a trigger action is set, setting the tool digital input value to high will trigger the corresponding robot action.

For example, if `TOOL_IO[0]` is set as input and its trigger action is Handguide (`StandardInputAction::Handguide`), setting `TOOL_IO[0]` to high will enable hand-guiding mode. Setting `TOOL_IO[0]` to low will exit hand-guiding mode.

Parameters

index	Indicates the IO pin, starting from 0. For example, 0 means the first pin.
action	Trigger action

Returns

Returns 0 on success; error code on failure. AUBO_REQUEST_IGNORE AUBO_BUSY AUBO_BAD_STATE -AUBO_INVL_ARGUMENT -AUBO_BAD_STATE

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
setToolDigitalInputAction(self: pyaubo_sdk.IoControl, arg0: int, arg1: arcs::common\_interface::StandardInputAct
-> int
```

Lua function prototype

```
setToolDigitalInputAction(index: number, action: number) -> nil
```

Lua example

```
setToolDigitalInputAction(0,1)
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.IoControl.setToolDigitalInputAction","params":[0,"Handguide"]↵
,"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":0}
```

8.6.4.2.77 setToolDigitalOutput()

```
int arcs::common_interface::IoControl::setToolDigitalOutput (
    int index,
    bool value)
```

Set the value of tool digital output.

Parameters

index	Indicates the IO pin, starting from 0. For example, 0 means the first pin.
value	Output value.

Returns

Returns 0 on success; error code on failure. AUBO_REQUEST_IGNORE AUBO_BUSY AUBO_↵
_BAD_STATE -AUBO_INVL_ARGUMENT -AUBO_BAD_STATE

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
setToolDigitalOutput(self: pyaubo_sdk.IoControl, arg0: int, arg1: bool) -> int
```

Lua function prototype

```
setToolDigitalOutput(index: number, value: boolean) -> nil
```

Lua example

```
setToolDigitalOutput(0,true)
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.IoControl.setToolDigitalOutput","params":[0,true],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":0}
```

8.6.4.2.78 setToolDigitalOutputPulse()

```
int arcs::common_interface::IoControl::setToolDigitalOutputPulse (
    int index,
    bool value,
    double duration)
```

Set tool digital output pulse.

Parameters

index	
value	
duration	

Returns

Returns 0 on success; error code on failure. AUBO_REQUEST_IGNORE AUBO_BUSY AUBO_BAD_STATE -AUBO_INVL_ARGUMENT -AUBO_BAD_STATE

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
setToolDigitalOutputPulse(self: pyaubo_sdk.IoControl, arg0: int, arg1: bool, arg2: float) -> int
```

Lua function prototype

```
setToolDigitalOutputPulse(index: number, value: boolean, duration: number) -> nil
```

Lua example

```
setToolDigitalOutputPulse(0,true,2)
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.IoControl.setToolDigitalOutputPulse","params":[0,true,0.5],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":0}
```

8.6.4.2.79 setToolDigitalOutputRunstate()

```
int arcs::common_interface::IoControl::setToolDigitalOutputRunstate (
    int index,
    StandardOutputRunState runstate)
```

Generated by Doxygen
Set the runstate for tool digital output.

Parameters

index	Indicates the IO pin, starting from 0. For example, 0 means the first pin.
runstate	Output runstate selection

Returns

Returns 0 on success; error code on failure. AUBO_REQUEST_IGNORE AUBO_BUSY AUBO_↵
_BAD_STATE -AUBO_INVL_ARGUMENT -AUBO_BAD_STATE

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
setToolDigitalOutputRunstate(self: pyaubo_sdk.IoControl, arg0: int, arg1: arcs::common\_interface::StandardOutp
-> int
```

Lua function prototype

```
setToolDigitalOutputRunstate(index: number, runstate: number) -> nil
```

Lua example

```
setToolDigitalOutputRunstate(0,1)
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.IoControl.setToolDigitalOutputRunstate","params":[0,"None"]}↵  
,"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":0}
```

8.6.4.2.80 setToolIoInput()

```
int arcs::common_interface::IoControl::setToolIoInput (
    int index,
    bool input)
```

Set the specified tool digital IO as input or output.

Generated by Doxygen

Tool digital IOs are special and can be configured as input or output.

Parameters

index	Indicates the IO pin, starting from 0. For example, 0 means the first pin.
input	Indicates whether the specified IO is input. If input is true, set the IO as input; otherwise, set as output.

Returns

Returns 0 on success; error code on failure. AUBO_BUSY AUBO_BAD_STATE -AUBO_INVL↵
_ARGUMENT -AUBO_BAD_STATE

Exceptions

<code>arcs::common_interface::AuboException</code>	
--	--

Python function prototype

```
setToolIoInput(self: pyaubo_sdk.IoControl, arg0: int, arg1: bool) -> int
```

Lua function prototype

```
setToolIoInput(index: number, input: boolean) -> nil
```

Lua example

```
setToolIoInput(0,true)
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.IoControl.setToolIoInput","params":[0,true],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":0}
```

8.6.4.2.81 setToolVoltageOutputDomain()

```
int arcs::common_interface::IoControl::setToolVoltageOutputDomain (
    int domain)
```

Set the tool voltage output value (unit: V)

Parameters

domain	Tool voltage output value, can be 0, 12, or 24. 0 means 0V, 12 means 12V, 24 means 24V.
--------	--

Returns

Returns 0 on success; error code on failure. AUBO_REQUEST_IGNORE AUBO_BUSY AUBO_↵
_BAD_STATE -AUBO_INVL_ARGUMENT -AUBO_BAD_STATE

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
setToolVoltageOutputDomain(self: pyaubo_sdk.IoControl, arg0: int) -> int
```

Lua function prototype

```
setToolVoltageOutputDomain(domain: number) -> nil
```

Lua example

```
setToolVoltageOutputDomain(24)
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.IoControl.setToolVoltageOutputDomain","params":[24],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":0}
```

8.6.4.2.82 unwindEncDeltaTickCount()

```
int arcs::common_interface::IoControl::unwindEncDeltaTickCount (
    int delta_count)
```

Prevent counting errors when the count exceeds the range

Parameters

delta_count	
-------------	--

Returns

Returns 0 on success; error code on failure. AUBO_NO_ACCESS AUBO_BUSY AUBO_BAD↵
_STATE -AUBO_BAD_STATE

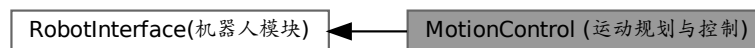
Exceptions

arcs::common_interface::AuboException	
---	--

8.6.5 MotionControl ()

[MotionControl](#).

Collaboration diagram for MotionControl ():



Functions

- double [arcs::common_interface::MotionControl::getEradius](#) ()
Get the equivalent radius, in meters.
- int [arcs::common_interface::MotionControl::setEradius](#) (double eradius)
Set the equivalent radius, in meters.
- int [arcs::common_interface::MotionControl::setSpeedFraction](#) (double fraction)
Dynamically adjust the robot's speed and acceleration ratio (0., 1.
- double [arcs::common_interface::MotionControl::getSpeedFraction](#) ()
Get the speed and acceleration ratio, default is 1.
- int [arcs::common_interface::MotionControl::speedFractionCritical](#) (bool enable)
Speed fraction critical section.
- bool [arcs::common_interface::MotionControl::isSpeedFractionCritical](#) ()
Whether it is in the speed fraction critical section
- bool [arcs::common_interface::MotionControl::isBlending](#) ()
Whether it is in the blending area
- int [arcs::common_interface::MotionControl::pathOffsetLimits](#) (double v, double a)
Set the maximum speed and maximum acceleration for offset.

- `int arcs::common_interface::MotionControl::pathOffsetCoordinate (int ref_coord)`
Set the reference coordinate system for offset.
- `int arcs::common_interface::MotionControl::pathOffsetEnable ()`
Enable path offset
- `int arcs::common_interface::MotionControl::pathOffsetSet (const std::vector< double > &offset, int type=0)`
Set path offset
- `int arcs::common_interface::MotionControl::pathOffsetDisable ()`
Disable path offset
- `int arcs::common_interface::MotionControl::pathOffsetSupv (const std::vector< double > &min, const std::vector< double > &max, int strategy)`
- `int arcs::common_interface::MotionControl::jointOffsetEnable ()`
Enable joint offset
- `int arcs::common_interface::MotionControl::jointOffsetSet (const std::vector< double > &offset, int type=1)`
Set joint offset
- `int arcs::common_interface::MotionControl::jointOffsetDisable ()`
Disable joint offset
- `int arcs::common_interface::MotionControl::getQueueSize ()`
Get the number of enqueued instruction segments (INST), including motion instructions such as move↵ Joint/moveLine/moveCircle and configuration instructions such as setPayload.
- `int arcs::common_interface::MotionControl::getTrajectoryQueueSize ()`
Get the number of enqueued motion planning interpolation points
- `int arcs::common_interface::MotionControl::getExecId ()`
Get the ID of the currently interpolating motion instruction segment.
- `double arcs::common_interface::MotionControl::getDuration (int id)`
Get the expected execution duration of the motion segment with the specified ID.
- `double arcs::common_interface::MotionControl::getMotionLeftTime (int id)`
Get the remaining execution time of the motion segment with the specified ID.
- `int arcs::common_interface::MotionControl::stopMove (bool quick, bool all_tasks)`
StopMove is used to stop robot and external axes movements and any belonging process temporarily.
- `int arcs::common_interface::MotionControl::startMove ()`
StartMove is used to resume robot, external axes movement and belonging process after the movement has been stopped
- `int arcs::common_interface::MotionControl::storePath (bool keep_sync)`
storePath
- `int arcs::common_interface::MotionControl::clearPath ()`
ClearPath clears the whole motion path on the current motion path level (base level or StorePath level).
- `int arcs::common_interface::MotionControl::restoPath ()`
restoPath
- `double arcs::common_interface::MotionControl::getProgress ()`
Get the execution progress of the current motion instruction segment.
- `int arcs::common_interface::MotionControl::setWorkObjectHold (const std::string &module_↵ name, const std::vector< double > &mounting_pose)`
Specify the name and mounting position when the workpiece is installed on the end of another robot or external axis.
- `std::tuple< std::string, std::vector< double > > arcs::common_interface::MotionControl::getWorkObjectHold ()`
getWorkObjectHold
- `std::vector< double > arcs::common_interface::MotionControl::getPauseJointPositions ()`
- `int arcs::common_interface::MotionControl::setResumeStartPoint (const std::vector< double > &q, int move_type, double blend_radius, const std::vector< double > &qdmax, const std::vector< double > &qddmax, const std::vector< double > &vmax, const std::vector< double > &amax)`

- Set resume motion parameters
- int [arcs::common_interface::MotionControl::getResumeMode](#) ()
Get resume motion mode
- ARCS_DEPRECATED int [arcs::common_interface::MotionControl::setServoMode](#) (bool enable)
Set servo mode Use setServoModeSelect instead
- ARCS_DEPRECATED bool [arcs::common_interface::MotionControl::isServoModeEnabled](#) ()
Determine whether the servo mode is enabled.
- int [arcs::common_interface::MotionControl::setServoModeSelect](#) (int mode)
Set servo motion mode
- int [arcs::common_interface::MotionControl::getServoModeSelect](#) ()
Get the servo motion mode
- int [arcs::common_interface::MotionControl::servoJoint](#) (const std::vector< double > &q, double a, double v, double t, double lookahead_time, double gain)
Joint space servo
- int [arcs::common_interface::MotionControl::servoCartesian](#) (const std::vector< double > &pose, double a, double v, double t, double lookahead_time, double gain)
Cartesian space servo
- int [arcs::common_interface::MotionControl::servoJointWithAxes](#) (const std::vector< double > &q, const std::vector< double > &extq, double a, double v, double t, double lookahead_time, double gain)
Servo motion (with external axes), used for executing offline trajectories, pass-through user planned trajectories, etc.
- int [arcs::common_interface::MotionControl::servoCartesianWithAxes](#) (const std::vector< double > &pose, const std::vector< double > &extq, double a, double v, double t, double lookahead_time, double gain)
Servo motion (with external axes), used for executing offline trajectories, pass-through user planned trajectories, etc.
- int [arcs::common_interface::MotionControl::trackJoint](#) (const std::vector< double > &q, double t, double smooth_scale, double delay_sacle)
Tracking motion, used for executing offline trajectories or passing through user-planned trajectories, etc.
- int [arcs::common_interface::MotionControl::trackCartesian](#) (const std::vector< double > &pose, double t, double smooth_scale, double delay_sacle)
Tracking motion, used for executing offline trajectories or passing through user-planned trajectories, etc.
- int [arcs::common_interface::MotionControl::followJoint](#) (const std::vector< double > &q)
Joint space following
- int [arcs::common_interface::MotionControl::followLine](#) (const std::vector< double > &pose)
Cartesian space following
- int [arcs::common_interface::MotionControl::speedJoint](#) (const std::vector< double > &qd, double a, double t)
Joint space velocity following
- int [arcs::common_interface::MotionControl::resumeSpeedJoint](#) (const std::vector< double > &qd, double a, double t)
Joint space velocity following (used to move to a safe position after a collision during process execution)
- int [arcs::common_interface::MotionControl::speedLine](#) (const std::vector< double > &xd, double a, double t)
Cartesian space velocity following
- int [arcs::common_interface::MotionControl::resumeSpeedLine](#) (const std::vector< double > &xd, double a, double t)
Cartesian space velocity following (used to move to a safe position after a collision during process execution)
- int [arcs::common_interface::MotionControl::moveSpline](#) (const std::vector< double > &q, double a, double v, double duration)

- Perform spline interpolation in joint space
- `int arcs::common_interface::MotionControl::moveJoint` (const std::vector< double > &q, double a, double v, double blend_radius, double duration)
 - Add joint motion
- `int arcs::common_interface::MotionControl::moveJointWithAxisGroup` (const std::vector< double > &q, double a, double v, double blend_radius, double duration, const std::string &group_name, const std::vector< double > &extq)
 - Synchronous motion of robot and external axes
- `int arcs::common_interface::MotionControl::resumeMoveJoint` (const std::vector< double > &q, double a, double v, double duration)
 - Move to the pause point using joint motion.
- `int arcs::common_interface::MotionControl::moveLine` (const std::vector< double > &pose, double a, double v, double blend_radius, double duration)
 - Add linear motion
- `int arcs::common_interface::MotionControl::moveLineWithAxisGroup` (const std::vector< double > &pose, double a, double v, double blend_radius, double duration, const std::string &group_name, const std::vector< double > &extq)
 - Linear motion synchronized with external axes
- `int arcs::common_interface::MotionControl::moveProcess` (const std::vector< double > &pose, double a, double v, double blend_radius)
 - Add process motion
- `int arcs::common_interface::MotionControl::resumeMoveLine` (const std::vector< double > &pose, double a, double v, double duration)
 - Move to the pause point using linear motion.
- `int arcs::common_interface::MotionControl::moveCircle` (const std::vector< double > &via_pose, const std::vector< double > &end_pose, double a, double v, double blend_radius, double duration)
 - Add circular arc motion
- `int arcs::common_interface::MotionControl::moveCircleWithAxisGroup` (const std::vector< double > &via_pose, const std::vector< double > &end_pose, double a, double v, double blend_radius, double duration, const std::string &group_name, const std::vector< double > &extq)
 - moveCircle with external axes synchronized motion
- `int arcs::common_interface::MotionControl::setCirclePathMode` (int mode)
 - Set circle path mode
- `int arcs::common_interface::MotionControl::moveCircle2` (const CircleParameters ¶m)
 - Advanced arc or circular motion
- `int arcs::common_interface::MotionControl::pathBufferAlloc` (const std::string &name, int type, int size)
 - Create a new path point buffer
- `int arcs::common_interface::MotionControl::pathBufferAppend` (const std::string &name, const std::vector< std::vector< double > > &waypoints)
 - Add waypoints to the path buffer
- `int arcs::common_interface::MotionControl::pathBufferEval` (const std::string &name, const std::vector< double > &a, const std::vector< double > &v, double t)
 - Perform computation and optimization (time-consuming operations).
- `bool arcs::common_interface::MotionControl::pathBufferValid` (const std::string &name)
 - Whether the buffer with the specified name is valid
- `int arcs::common_interface::MotionControl::pathBufferFree` (const std::string &name)
 - Release path buffer
- `int arcs::common_interface::MotionControl::pathBufferFilter` (const std::string &name, int order, double fd, double fs)
 - Joint space path filter
- `std::vector< std::string > arcs::common_interface::MotionControl::pathBufferList` ()
 - List all cached path names

- int [arcs::common_interface::MotionControl::movePathBuffer](#) (const std::string &name)
Execute the cached path
- int [arcs::common_interface::MotionControl::moveIntersection](#) (const std::vector< std::vector< double > > &poses, double a, double v, double main_pipe_radius, double sub_pipe_radius, double normal_distance, double normal_alpha)
Intersection interface
- int [arcs::common_interface::MotionControl::stopJoint](#) (double acc)
Stop motion in joint space
- int [arcs::common_interface::MotionControl::resumeStopJoint](#) (double acc)
Stop motion in joint space (used after moving to a safe position via resumeSpeedJoint following a collision during process execution)
- int [arcs::common_interface::MotionControl::stopLine](#) (double acc, double acc_rot)
Stop motions in Cartesian space such as moveLine/moveCircle.
- int [arcs::common_interface::MotionControl::resumeStopLine](#) (double acc, double acc_rot)
Stop motion in Cartesian space (used after moving to a safe position via resumeSpeedLine following a collision during process execution)
- int [arcs::common_interface::MotionControl::weaveStart](#) (const std::string ¶ms)
Start weaving: between weaveStart and weaveEnd, moveLine/moveCircle/moveProcess follows params.
- int [arcs::common_interface::MotionControl::weaveEnd](#) ()
End weaving
- int [arcs::common_interface::MotionControl::setFuturePointSamplePeriod](#) (double sample_time)
Set the sampling interval for points on the future path
- std::vector< std::vector< double > > [arcs::common_interface::MotionControl::getFuturePathPointsJoint](#) ()
Get trajectory points on the future path
- int [arcs::common_interface::MotionControl::setConveyorTrackEncoder](#) (int encoder_id, int tick_↵ per_meter)
Set conveyor encoder parameters
- int [arcs::common_interface::MotionControl::conveyorTrackCircle](#) (int encoder_id, const std::↵ vector< double > ¢er, bool rotate_tool)
Circular conveyor tracking
- int [arcs::common_interface::MotionControl::conveyorTrackLine](#) (int encoder_id, const std::vector< double > &direction)
Linear conveyor tracking
- int [arcs::common_interface::MotionControl::conveyorTrackStop](#) (int encoder_id, double a)
Stop conveyor tracking
- bool [arcs::common_interface::MotionControl::conveyorTrackSwitch](#) (int encoder_id)
Switch conveyor tracking item.
- bool [arcs::common_interface::MotionControl::hasItemOnConveyorToTrack](#) (int encoder_id)
Whether there is an item on the conveyor that can be tracked
- int [arcs::common_interface::MotionControl::conveyorTrackCreatItem](#) (int encoder_id, int item_id, const std::vector< double > &offset)
Add an item to the conveyor queue
- int [arcs::common_interface::MotionControl::setConveyorTrackCompensate](#) (int encoder_id, double comp)
Set the compensation value for conveyor tracking
- bool [arcs::common_interface::MotionControl::isConveyorTrackSync](#) (int encoder_id)
Determine whether the conveyor and the robot arm have reached relative rest
- int [arcs::common_interface::MotionControl::setConveyorTrackLimit](#) (int encoder_id, double limit)
Set the maximum distance limit for conveyor tracking
- int [arcs::common_interface::MotionControl::setConveyorTrackStartWindow](#) (int encoder_id, double window_min, double window_max)

- Set the start window for conveyor tracking
- `int arcs::common_interface::MotionControl::setConveyorTrackSensorOffset (int encoder_id, double offset)`
- Set the distance from the conveyor teaching position to the sync switch
- `int arcs::common_interface::MotionControl::setConveyorTrackSyncSeparation (int encoder_id, double distance, double time)`
- Set conveyor sync separation, used to filter out unwanted signals from the sync switch.
- `bool arcs::common_interface::MotionControl::isConveyorTrackExceed (int encoder_id)`
- Whether the workpiece on the conveyor has moved beyond the maximum limit
- `int arcs::common_interface::MotionControl::conveyorTrackClearItems (int encoder_id)`
- Clear all items in the conveyor queue
- `std::vector< int > arcs::common_interface::MotionControl::getConveyorTrackQueue (int encoder_id)`
- Get encoder values of the conveyor queue
- `int arcs::common_interface::MotionControl::moveSpiral (const SpiralParameters ¶m, double blend_radius, double v, double a, double t)`
- Spiral motion
- `int arcs::common_interface::MotionControl::getLookAheadSize ()`
- Get look-ahead segment size
- `int arcs::common_interface::MotionControl::setLookAheadSize (int size)`
- Set look-ahead segment size
- `int arcs::common_interface::MotionControl::weaveUpdateParameters (const std::string ¶ms)`
- Update frequency and amplitude during weaving process
- `int arcs::common_interface::MotionControl::enableJointSoftServo (const std::vector< double > &stiffness)`
- Set joint current loop stiffness coefficient
- `int arcs::common_interface::MotionControl::disableJointSoftServo ()`
- Disable joint current loop stiffness coefficient
- `bool arcs::common_interface::MotionControl::isJointSoftServoEnabled ()`
- Determine whether the joint current loop stiffness coefficient is enabled.
- `int arcs::common_interface::MotionControl::enableVibrationSuppress (const std::vector< double > &omega, const std::vector< double > &zeta, int level)`
- `int arcs::common_interface::MotionControl::disbaleVibrationSuppress ()`
- `int arcs::common_interface::MotionControl::setTimeOptimalEnable (bool enable)`
- `bool arcs::common_interface::MotionControl::isTimeOptimalEnabled ()`
- Get the time optimal algorithm state: true - enable false - disable
- `bool arcs::common_interface::MotionControl::isSupportedTimeOptimal ()`
- Check whether the time optimal algorithm is supported true - supported false - not supported
- `int arcs::common_interface::MotionControl::setTcpMaxLinearVelocity (double v)`
- Set TCP maximum linear velocity
- `double arcs::common_interface::MotionControl::getTcpMaxLinearVelocity ()`
- Get TCP maximum linear velocity
- `int arcs::common_interface::MotionControl::resetTcpMaxLinearVelocity ()`
- Reset TCP maximum linear velocity
- `int arcs::common_interface::MotionControl::setEndPath ()`
- Set end path (terminate blending at current trajectory segment)

8.6.5.1 Detailed Description

[MotionControl.](#)

8.6.5.2 Function Documentation

8.6.5.2.1 clearPath()

```
int arcs::common_interface::MotionControl::clearPath ()
```

ClearPath clears the whole motion path on the current motion path level (base level or StorePath level).

Returns

Returns 0 on success; otherwise returns an error code: AUBO_BAD_STATE AUBO_BUSY - AUBO_BAD_STATE

Exceptions

arcs::common_interface::AuboException	
---	--

Lua function prototype

```
clearPath() -> number
```

Lua example

```
num = clearPath()
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.MotionControl.clearPath","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":0}
```

8.6.5.2.2 conveyorTrackCircle()

```
int arcs::common_interface::MotionControl::conveyorTrackCircle (
    int encoder_id,
    const std::vector< double > & center,
    bool rotate_tool)
```

Circular conveyor tracking

Note

Not implemented yet

Parameters

encoder_id	0 - integrated sensor
rotate_tool	

Returns

Exceptions

arcs::common_interface::AuboException	
---	--

8.6.5.2.3 conveyorTrackClearItems()

```
int arcs::common_interface::MotionControl::conveyorTrackClearItems (
    int encoder_id)
```

Clear all items in the conveyor queue

Parameters

encoder_id	Reserved
------------	----------

Returns

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
conveyorTrackClearItems(self: pyaubo_sdk.MotionControl, arg0: int) -> int
```

Lua function prototype

```
conveyorTrackClearItems(encoder_id: number) -> int
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.MotionControl.conveyorTrackClearItems","params":[0],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":0}
```

8.6.5.2.4 conveyorTrackCreatItem()

```
int arcs::common_interface::MotionControl::conveyorTrackCreatItem (
    int encoder_id,
    int item_id,
    const std::vector< double > & offset)
```

Add an item to the conveyor queue

Parameters

encoder_id	Reserved
item_id	Item ID
offset	Offset of the current item position relative to the template item position

Returns

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
conveyorTrackCreatItem(self: pyaubo_sdk.MotionControl, arg0: int, arg1:int, arg2: List[float]) ->
int
```

Lua function prototype

```
conveyorTrackCreatItem(encoder_id: number, item_id: number, offset: table) -> number
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.MotionControl.conveyorTrackCreatItem","params":[0,2, {0.0,0.↵
0,0.0,0.0,0.0,0.0}], "id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":0.0}
```

8.6.5.2.5 conveyorTrackLine()

Generated by Doxygen

```
int arcs::common_interface::MotionControl::conveyorTrackLine (
    int encoder_id,
    const std::vector< double > & direction)
```

Parameters

encoder_id	Reserved
direction	The movement direction of the conveyor relative to the robot base coordinate system

Returns

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
conveyorTrackLine(self: pyaubo_sdk.MotionControl) -> int
```

Lua function prototype

```
conveyorTrackLine -> int
```

JSON-RPC request example

```
{ "jsonrpc": "2.0", "method": "rob1.MotionControl.conveyorTrackLine", "params": [1, [1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]], "id": 1 }
```

JSON-RPC response example

```
{ "id": 1, "jsonrpc": "2.0", "result": 0 }
```

8.6.5.2.6 conveyorTrackStop()

```
int arcs::common_interface::MotionControl::conveyorTrackStop (
    int encoder_id,
    double a)
```

Stop conveyor tracking

Parameters

encoder_id	Reserved
a	Reserved

Returns

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
conveyorTrackStop(self: pyaubo_sdk.MotionControl) -> int
```

Lua function prototype

```
conveyorTrackStop -> int
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.MotionControl.conveyorTrackStop","params":[0,1.0],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":0}
```

8.6.5.2.7 conveyorTrackSwitch()

```
bool arcs::common_interface::MotionControl::conveyorTrackSwitch (
    int encoder_id)
```

Switch conveyor tracking item.

If the current item is being tracked, it will be dequeued and no longer tracked, returning true. If no item is currently being tracked, returns false.

Returns

Exceptions

arcs::common_interface::AuboException	
---	--

Parameters

encoder_id	Reserved
------------	----------

Python function prototype

```
conveyorTrackSwitch(self: pyaubo_sdk.MotionControl) -> bool
```

Lua function prototype

```
conveyorTrackSwitch\(\) -> boolean
```

JSON-RPC request example

```
{"jsonrpc": "2.0", "method": "rob1.MotionControl.conveyorTrackSwitch", "params": [0], "id": 1}
```

JSON-RPC response example

```
{"id": 1, "jsonrpc": "2.0", "result": true}
```

8.6.5.2.8 disableJointSoftServo()

```
int arcs::common_interface::MotionControl::disableJointSoftServo ()
```

Disable joint current loop stiffness coefficient

Returns

Returns 0 on success, otherwise error code

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
disableJointSoftServo(self: pyaubo_sdk.MotionControl) -> int
```

Lua function prototype

```
disableJointSoftServo() -> number
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.MotionControl.disableJointSoftServo","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":1}
```

8.6.5.2.9 disbaleVibrationSuppress()

```
int arcs::common_interface::MotionControl::disbaleVibrationSuppress ()
```

8.6.5.2.10 enableJointSoftServo()

```
int arcs::common_interface::MotionControl::enableJointSoftServo (
    const std::vector< double > & stiffness)
```

Set joint current loop stiffness coefficient

Parameters

stiffness	Stiffness coefficient for each joint, as a percentage [0 -> 1]. The larger the value, the stiffer the joint.
-----------	--

Returns

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
enableJointSoftServo(self: pyaubo_sdk.RobotConfig, arg0: List[float]) -> int
```

Lua function prototype

```
enableJointSoftServo(stiffness: table) -> int
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.RobotConfig.enableJointSoftServo","params":[[0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0]],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":0}
```

8.6.5.2.11 enableVibrationSuppress()

```
int arcs::common_interface::MotionControl::enableVibrationSuppress (
    const std::vector< double > & omega,
    const std::vector< double > & zeta,
    int level)
```

8.6.5.2.12 followJoint()

```
int arcs::common_interface::MotionControl::followJoint (
    const std::vector< double > & q)
```

Joint space following

Note

Not implemented yet

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
followJoint(self: pyaubo_sdk.MotionControl, arg0: List[float]) -> int
```

Lua function prototype

```
followJoint(q: table) -> nil
```

Lua example

```
followJoint({0,0,0,0,0,0})
```


8.6.5.2.13 followLine()

```
int arcs::common_interface::MotionControl::followLine (
    const std::vector< double > & pose)
```

Cartesian space following

Note

Not implemented yet

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
followLine(self: pyaubo_sdk.MotionControl, arg0: List[float]) -> int
```

Lua function prototype

```
followLine(pose: table) -> nil
```

Lua example

```
followLine({0.58712,-0.15775,0.48703,2.76,0.344,1.432})
```

8.6.5.2.14 getConveyorTrackQueue()

```
std::vector< int > arcs::common_interface::MotionControl::getConveyorTrackQueue (
    int encoder_id)
```

Get encoder values of the conveyor queue

Parameters

encoder_id	Reserved
------------	----------

Returns

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
getConveyorTrackQueue(self: pyaubo_sdk.MotionControl, arg0: int) -> List[int]
```

Lua function prototype

```
getConveyorTrackQueue(encoder_id: number) -> table
```

JSON-RPC request example

```
{"jsonrpc": "2.0", "method": "rob1.MotionControl.getConveyorTrackQueue", "params": [0, "id": 1]}
```

JSON-RPC response example

```
{"id": 1, "jsonrpc": "2.0", "result": {[-500, -200, 150, -50]}}
```

8.6.5.2.15 getDuration()

```
double arcs::common_interface::MotionControl::getDuration (
    int id)
```

Get the expected execution duration of the motion segment with the specified ID.

Parameters

id	Motion segment ID
----	-------------------

Returns

Returns the expected execution duration

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
getDuration(self: pyaubo_sdk.MotionControl, arg0: int) -> float
```

Lua function prototype

```
getDuration(id: number) -> number
```

Lua example

```
num = getDuration(16)
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.MotionControl.getDuration","params":[16],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":0.0}
```

8.6.5.2.16 getEqradius()

double arcs::common_interface::MotionControl::getEqradius ()

Get the equivalent radius, in meters.

When using moveLine/moveCircle, the rotation angle of the end effector's pose is converted to an equivalent end position movement. Can be set via setEqradius, default is 1.

Returns

Returns the equivalent radius.

Exceptions

arcs::common_interface::AuboException	
---------------------------------------	--

Python function prototype

```
getEqradius(self: pyaubo_sdk.MotionControl) -> float
```

Lua function prototype

```
getEqradius() -> number
```

Lua example

```
num = getEqradius()
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.MotionControl.getEqradius","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":1.0}
```

8.6.5.2.17 `getExecId()`

```
int arcs::common_interface::MotionControl::getExecId ()
```

Get the ID of the currently interpolating motion instruction segment.

Returns

The ID of the currently interpolating motion instruction segment.

Return values

-1	Indicates the trajectory queue is empty. Both the buffer in <code>movePathBuffer</code> motion and the queue in the planner (such as <code>moveJoint</code> and <code>moveLine</code>) belong to the trajectory queue.
----	--

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
getExecId(self: pyaubo_sdk.MotionControl) -> int
```

Lua function prototype

```
getExecId\(\) -> number
```

Lua example

```
num = getExecId\(\)
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.MotionControl.getExecId","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":-1}
```

8.6.5.2.18 `getFuturePathPointsJoint()`

```
std::vector< std::vector< double > > arcs::common_interface::MotionControl::getFuturePathPointsJoint ()
```

Get trajectory points on the future path

Exceptions

arcs::common_interface::AuboException	
---	--

JSON-RPC request example

```
{"jsonrpc": "2.0", "method": "rob1.MotionControl.getFuturePathPointsJoint", "params": [], "id": 1}
```

JSON-RPC response example

```
{"id": 1, "jsonrpc": "2.0", "result": []}
```

8.6.5.2.19 getLookAheadSize()

```
int arcs::common_interface::MotionControl::getLookAheadSize ()
```

Get look-ahead segment size

Returns

Returns the look-ahead segment size

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
getLookAheadSize(self: pyaubo_sdk.MotionControl) -> int
```

Lua function prototype

```
getLookAheadSize\(\) -> number
```

JSON-RPC request example

```
{"jsonrpc": "2.0", "method": "rob1.MotionControl.getLookAheadSize", "params": [], "id": 1}
```

JSON-RPC response example

```
{"id": 1, "jsonrpc": "2.0", "result": 1}
```

8.6.5.2.20 getMotionLeftTime()

Parameters

id	Motion segment ID
----	-------------------

Returns

Returns the remaining execution time

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
getMotionLeftTime(self: pyaubo_sdk.MotionControl, arg0: int) -> float
```

Lua function prototype

```
getMotionLeftTime(id: number) -> number
```

Lua example

```
num = getMotionLeftTime(16)
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.MotionControl.getMotionLeftTime","params":[16],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":0.0}
```

8.6.5.2.21 getPauseJointPositions()

```
std::vector< double > arcs::common_interface::MotionControl::getPauseJointPositions ()
```

Note

Get the joint positions at the pause point.

Commonly used during process recovery after a collision occurs (first move to the pause position using `resumeMoveJoint` or `resumeMoveLine`, then resume the process).

Generated by Doxygen

Returns

Pause joint positions

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
getPauseJointPositions(self: pyaubo_sdk.MotionControl) -> List[float]
```

Lua function prototype

```
getPauseJointPositions() -> table
```

Lua example

```
JointPositions = getPauseJointPositions()
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.MotionControl.getPauseJointPositions","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":[8.2321e-13,-0.200999,1.33999,0.334999,1.206,-6.39383e-12]}
```

8.6.5.2.22 getProgress()

```
double arcs::common_interface::MotionControl::getProgress ()
```

Get the execution progress of the current motion instruction segment.

Returns

Returns the execution progress.

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
getProgress(self: pyaubo_sdk.MotionControl) -> float
```

Lua function prototype

```
getProgress() -> number
```

Lua example

```
num = getProgress()
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.MotionControl.getProgress","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":0.0}
```

8.6.5.2.23 getQueueSize()

```
int arcs::common_interface::MotionControl::getQueueSize ()
```

Get the number of enqueued instruction segments (INST), including motion instructions such as moveJoint/moveLine/moveCircle and configuration instructions such as setPayload.

Instructions are generally indicated with `_INST` in macro definitions, for example: `_INST(MotionControl, 5, moveJoint, q, a, v, blend_radius, duration)`

Returns

The number of enqueued instruction segments.

Exceptions

<code>arcs::common_interface::AuboException</code>	
--	--

Python function prototype

```
getQueueSize(self: pyaubo_sdk.MotionControl) -> int
```

Lua function prototype

```
getQueueSize() -> number
```

Lua example

```
num = getQueueSize()
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.MotionControl.getQueueSize","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":0}
```


8.6.5.2.24 getResumeMode()

```
int arcs::common_interface::MotionControl::getResumeMode ()
```

Get resume motion mode

Returns

0: Resume start point is the pause point; 1: Resume start point is the specified point

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
getResumeMode(self: pyaubo_sdk.MotionControl) -> int
```

Lua function prototype

```
getResumeMode\(\) -> int
```

Lua example

```
num = getResumeMode\(\)
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.MotionControl.getResumeMode","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":false}
```

8.6.5.2.25 getServoModeSelect()

```
int arcs::common_interface::MotionControl::getServoModeSelect ()
```

Get the servo motion mode

Returns

Lua function prototype

```
getServoModeSelect\(\) -> number
```

Lua example

```
num = getServoModeSelect\(\)
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.MotionControl.getServoModeSelect","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":0}
```

8.6.5.2.26 `getSpeedFraction()`

```
double arcs::common_interface::MotionControl::getSpeedFraction ()
```

Get the speed and acceleration ratio, default is 1.

Can be set via `setSpeedFraction` interface.

Returns

Returns the speed and acceleration ratio.

Exceptions

<code>arcs::common_interface::AuboException</code>	
--	--

Python function prototype

```
getSpeedFraction(self: pyaubo_sdk.MotionControl) -> float
```

Lua function prototype

```
getSpeedFraction() -> number
```

Lua example

```
num = getSpeedFraction()
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.MotionControl.getSpeedFraction","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":1.0}
```

8.6.5.2.27 `getTcpMaxLinearVelocity()`

```
double arcs::common_interface::MotionControl::getTcpMaxLinearVelocity ()
```

Get TCP maximum linear velocity

Returns

Current TCP maximum linear velocity value, unit is m/s

Exceptions

<code>arcs::common_interface::AuboException</code>	
--	--

Lua function prototype

```
getTcpMaxLinearVelocity() -> number
```

Lua example

```
local v = getTcpMaxLinearVelocity() -- Get TCP maximum linear velocity
```

Python function prototype

```
getTcpMaxLinearVelocity() -> float
```

Python example

```
v = getTcpMaxLinearVelocity() # Get TCP maximum linear velocity
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.MotionControl.getTcpMaxLinearVelocity","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":0.5}
```

8.6.5.2.28 getTrajectoryQueueSize()

```
int arcs::common_interface::MotionControl::getTrajectoryQueueSize ()
```

Get the number of enqueued motion planning interpolation points

Returns

The number of enqueued motion planning interpolation points

Exceptions

<code>arcs::common_interface::AuboException</code>	
--	--

Python function prototype

```
getTrajectoryQueueSize(self: pyaubo_sdk.MotionControl) -> int
```

Lua function prototype

```
getTrajectoryQueueSize() -> number
```

Lua example

```
num = getTrajectoryQueueSize()
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.MotionControl.getTrajectoryQueueSize","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":0}
```

8.6.5.2.29 getWorkObjectHold()

```
std::tuple< std::string, std::vector< double > > arcs::common_interface::MotionControl::getWorkObjectHold ()
```

getWorkObjectHold

Note

Not implemented yet

Returns

Returns a tuple containing the control module name and mounting pose

Exceptions

arcs::common_interface::AuboException	
---------------------------------------	--

Python function prototype

```
getWorkObjectHold(self: pyaubo_sdk.MotionControl) -> Tuple[str, List[float]]
```

Lua function prototype

```
getWorkObjectHold() -> table
```

Lua example

```
Object_table = getWorkObjectHold()
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.MotionControl.getWorkObjectHold","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":["",[]]}
```

8.6.5.2.30 hasItemOnConveyorToTrack()

```
bool arcs::common_interface::MotionControl::hasItemOnConveyorToTrack (
    int encoder_id)
```

Whether there is an item on the conveyor that can be tracked

Parameters

encoder_id	Reserved
------------	----------

Returns

If the first item in the queue is in tracking state, returns true. If the first item is not in tracking state, checks the rest of the queue for items within the start window. Items outside the start window are dequeued until an item is found within the window, which is then set to tracking state and returns true. If no item in the queue is within the start window, returns false.

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
hasItemOnConveyorToTrack(self: pyaubo_sdk.MotionControl) -> bool
```

Lua function prototype

```
hasItemOnConveyorToTrack\(\) -> boolean
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.MotionControl.hasItemOnConveyorToTrack","params":[0]↵
,"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":{"true}}
```

8.6.5.2.31 isBlending()

```
bool arcs::common_interface::MotionControl::isBlending ()
```

Generated by Doxygen

Whether it is in the blending area

Exceptions

arcs::common_interface::AuboException	
---	--

Lua function prototype

```
isBlending() -> bool
```

Lua example

```
status = isBlending()
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.MotionControl.isBlending","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":false}
```

8.6.5.2.32 isConveyorTrackExceed()

```
bool arcs::common_interface::MotionControl::isConveyorTrackExceed (
    int encoder_id)
```

Whether the workpiece on the conveyor has moved beyond the maximum limit

Parameters

encoder_id	Reserved
------------	----------

Returns

true : The movement distance exceeds the maximum limit false : The movement distance does not exceed the maximum limit

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
isConveyorTrackExceed(self: pyaubo_sdk.MotionControl, arg0: int) -> bool
```

Lua function prototype

```
isConveyorTrackExceed(encoder_id: number) -> bool
```

JSON-RPC request example

```
{"jsonrpc": "2.0", "method": "rob1.MotionControl.isConveyorTrackExceed", "params": [0], "id": 1}
```

JSON-RPC response example

```
{"id": 1, "jsonrpc": "2.0", "result": false}
```

8.6.5.2.33 isConveyorTrackSync()

```
bool arcs::common_interface::MotionControl::isConveyorTrackSync (
    int encoder_id)
```

Determine whether the conveyor and the robot arm have reached relative rest

Parameters

encoder_id	Reserved
------------	----------

Returns

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
isConveyorTrackSync(self: pyaubo_sdk.MotionControl, arg0: int) -> bool
```

Lua function prototype

```
isConveyorTrackSync(encoder_id: number) -> bool
```

JSON-RPC request example

```
{"jsonrpc": "2.0", "method": "rob1.MotionControl.isConveyorTrackSync", "params": [0], "id": 1}
```

JSON-RPC response example

```
{"id": 1, "jsonrpc": "2.0", "result": false}
```

8.6.5.2.34 isJointSoftServoEnabled()

bool arcs::common_interface::MotionControl::isJointSoftServoEnabled ()

Determine whether the joint current loop stiffness coefficient is enabled.

Returns

Returns true if enabled, otherwise returns false.

Exceptions

arcs::common_interface::AuboException	
---------------------------------------	--

Python function prototype

isJointSoftServoEnabled(self: pyaubo_sdk.MotionControl) -> bool

Lua function prototype

isJointSoftServoEnabled() -> boolean

JSON-RPC request example

```
{"jsonrpc": "2.0", "method": "rob1.MotionControl.isJointSoftServoEnabled", "params": [], "id": 1}
```

JSON-RPC response example

```
{"id": 1, "jsonrpc": "2.0", "result": 1}
```

8.6.5.2.35 isServoModeEnabled()

ARCS_DEPRECATED bool arcs::common_interface::MotionControl::isServoModeEnabled ()

Determine whether the servo mode is enabled.

Use getServoModeSelect instead.

Returns

Returns true if enabled, otherwise returns false.

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
isServoModeEnabled(self: pyaubo_sdk.MotionControl) -> bool
```

Lua function prototype

```
isServoModeEnabled() -> boolean
```

Lua example

```
Servo_status = isServoModeEnabled()
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.MotionControl.isServoModeEnabled","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":false}
```

8.6.5.2.36 isSpeedFractionCritical()

```
bool arcs::common_interface::MotionControl::isSpeedFractionCritical ()
```

Whether it is in the speed fraction critical section

Returns

Returns true if in the speed fraction critical section; otherwise returns false

Exceptions

arcs::common_interface::AuboException	
---	--

Lua function prototype

```
isSpeedFractionCritical() -> bool
```

Lua example

```
status = isSpeedFractionCritical()
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.MotionControl.isSpeedFractionCritical","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":true}
```

8.6.5.2.37 isSupportedTimeOptimal()

bool arcs::common_interface::MotionControl::isSupportedTimeOptimal ()

Check whether the time optimal algorithm is supported true - supported false - not supported

Returns

Return whether the time-optimal algorithm is supported

Exceptions

arcs::common_interface::AuboException	
---------------------------------------	--

Python function prototype

isSupportedTimeOptimal(self: pyaubo_sdk.MotionControl) -> bool

Lua function prototype

isSupportedTimeOptimal() -> boolean

Lua example

flag = isSupportedTimeOptimal()

JSON-RPC request example

```
{"jsonrpc": "2.0", "method": "rob1.MotionControl.isSupportedTimeOptimal", "params": [], "id": 1}
```

JSON-RPC response example

```
{"id": 1, "jsonrpc": "2.0", "result": true}
```

8.6.5.2.38 isTimeOptimalEnabled()

bool arcs::common_interface::MotionControl::isTimeOptimalEnabled ()

Get the time optimal algorithm state: true - enable false - disable

Returns

Returns the time optimal algorithm state

Exceptions

<code>arcs::common_interface::AuboException</code>	
--	--

Python function prototype

```
isTimeOptimalEnabled(self: pyaubo_sdk.MotionControl) -> bool
```

Lua function prototype

```
isTimeOptimalEnabled() -> boolean
```

Lua example

```
flag = isTimeOptimalEnabled()
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.MotionControl.isTimeOptimalEnabled","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":true}
```

8.6.5.2.39 jointOffsetDisable()

```
int arcs::common_interface::MotionControl::jointOffsetDisable ()
```

Disable joint offset

Returns

Returns 0 on success; otherwise returns an error code: AUBO_BAD_STATE AUBO_BUSY - AUBO_BAD_STATE

Exceptions

<code>arcs::common_interface::AuboException</code>	
--	--

Python function prototype

```
jointOffsetDisable(self: pyaubo_sdk.MotionControl) -> int
```

Lua function prototype

```
jointOffsetDisable() -> nil
```

Lua example

```
jointOffsetDisable()
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.MotionControl.jointOffsetDisable","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":0}
```

8.6.5.2.40 jointOffsetEnable()

```
int arcs::common_interface::MotionControl::jointOffsetEnable ()
```

Enable joint offset

Returns

Returns 0 on success; otherwise returns an error code: AUBO_BAD_STATE AUBO_BUSY - AUBO_BAD_STATE

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
jointOffsetEnable(self: pyaubo_sdk.MotionControl) -> int
```

Lua function prototype

```
jointOffsetEnable() -> nil
```

Lua example

```
jointOffsetEnable()
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.MotionControl.jointOffsetEnable","params":[],"id":1}
```

Parameters

offset	Pose offset for each joint
type	

Returns

Returns 0 on success; otherwise returns error code AUBO_BAD_STATE AUBO_BUSY -AUBO_↵
_INVL_ARGUMENT -AUBO_BAD_STATE

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
jointOffsetSet(self: pyaubo_sdk.MotionControl, arg0: List[float], arg1: int) -> int
```

Lua function prototype

```
jointOffsetSet(offset: table, type: number) -> nil
```

Lua example

```
jointOffsetSet({0.1,0,0,0,0,0},1)
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.MotionControl.jointOffsetSet","params":[[0.1,0,0,0,0],1],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":0}
```

8.6.5.2.42 moveCircle()

```
int arcs::common_interface::MotionControl::moveCircle (
    const std::vector< double > & via_pose,
    const std::vector< double > & end_pose,
    double a,
    double v,
    double blend_radius,
    double duration)
Generated by Doxygen
```

Add circular arc motion

Parameters

via_pose	The pose of the via point during the arc motion
end_pose	The pose of the end point of the arc motion
a	Acceleration (if the position change between via_pose and the previous waypoint is less than 1mm and the posture change is greater than 1e-4 rad, this acceleration is treated as angular acceleration in rad/s ² ; otherwise, as linear acceleration in m/s ²)
v	Velocity (if the position change between via_pose and the previous waypoint is less than 1mm and the posture change is greater than 1e-4 rad, this velocity is treated as angular velocity in rad/s; otherwise, as linear velocity in m/s)
blend_radius	Blend radius, unit: m
duration	Execution time, unit: s Normally, when speed and acceleration are given, the trajectory duration can be determined. If you want to extend the trajectory duration, set the duration parameter. Duration can extend the trajectory time, but cannot shorten it. When duration = 0, it means the execution time is not specified, and the time will be calculated based on speed and acceleration. If duration is not 0, a and v values will be ignored.

Returns

Returns 0 on success; otherwise returns error code AUBO_BAD_STATE AUBO_BUSY -AUBO_↵
_INVL_ARGUMENT -AUBO_BAD_STATE

Exceptions

arcs::common_interface::AuboException

Python function prototype

```
moveCircle(self: pyaubo_sdk.MotionControl, arg0: List[float], arg1: List[float], arg2: float, arg3:
float, arg4: float, arg5: float) -> int
```

Lua function prototype

```
moveCircle(via_pose: table, end_pose: table, a: number, v: number, blend_radius: number,
duration: number) -> nil
```

Lua example

```
moveCircle({0.440164,-0.00249391,0.398658,2.45651,0,1.5708},{0.440164,0.166256,0.297599,2.↵
45651,0,1.5708},1.2,0.25,0)
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.MotionControl.moveCircle","params":[[0.440164,-0.00249391,0.↵
398658,2.45651,0,1.5708],[0.440164,0.166256,0.297599,2.45651,0,1.5708],1.2,0.25,0,0],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":0}
```

8.6.5.2.43 moveCircle2()

```
int arcs::common_interface::MotionControl::moveCircle2 (
    const CircleParameters & param)
```

Advanced arc or circular motion

Parameters

param	Circular motion parameters
-------	----------------------------

Returns

Returns 0 on success; otherwise returns an error code: AUBO_BAD_STATE AUBO_BUSY - AUBO_INVL_ARGUMENT -AUBO_BAD_STATE

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
moveCircle2(self: pyaubo_sdk.MotionControl, arg0: arcs::common_interface::CircleParameters) -> int
```

Lua function prototype

```
moveCircle2(param: table) -> nil
```

Lua example

```
moveCircle2({0.440164,-0.00249391,0.398658,2.45651,0,1.570},{0.440164,0.166256,0.297599,2.45651,0,1.5708},1.2,0.25,0,0,0,0,0)
```

JSON-RPC request example

```
{ "jsonrpc": "2.0", "method": "rob1.MotionControl.moveCircle2", "params": [ { "pose_via": [0.440164,-0.00249391,0.398658,2.45651,0,1.570], "pose_to": [0.440164,0.166256,0.297599,2.45651,0,1.5708], "a": 1.2, "v": 0.25, "blend_radius": 0, "duration": 0, "helix": 0, "spiral": 0, "direction": 0, "loop_times": 0 }, "id": 1 }
```

JSON-RPC response example

```
{ "id": 1, "jsonrpc": "2.0", "result": 0 }
```

8.6.5.2.44 moveCircleWithAxisGroup()

```
int arcs::common_interface::MotionControl::moveCircleWithAxisGroup (
    const std::vector< double > & via_pose,
    const std::vector< double > & end_pose,
    double a,
    double v,
    double blend_radius,
    double duration,
    const std::string & group_name,
    const std::vector< double > & extq)
```

moveCircle with external axes synchronized motion

Parameters

group_name	Name of the external axis group
via_pose	The pose of the via point during the arc motion
end_pose	The pose of the end point of the arc motion
a	Acceleration
v	Velocity
blend_radius	Blend radius
duration	Execution time

Returns

8.6.5.2.45 moveIntersection()

```
int arcs::common_interface::MotionControl::moveIntersection (
    const std::vector< std::vector< double > > & poses,
    double a,
    double v,
    double main_pipe_radius,
    double sub_pipe_radius,
    double normal_distance,
    double normal_alpha)
```

Intersection interface

Parameters

poses	Consists of three taught poses (first, move to the starting point. The starting point requirement: the intersection of the plane passing through the main cylinder axis and parallel to the sub-cylinder axis with the sub-cylinder at the bottom) p1: Intersection of the plane passing through the sub-cylinder axis and parallel to the main cylinder axis with the sub-cylinder at the left top p2: Intersection of the plane passing through the sub-cylinder axis and parallel to the main cylinder axis with the main cylinder at the left bottom p3: Intersection of the plane passing through the sub-cylinder axis and parallel to the main cylinder axis with the main cylinder at the right bottom
v	Velocity
a	Acceleration
main_pipe_radius	Main cylinder radius
sub_pipe_radius	Sub cylinder radius
normal_distance	Distance between the two cylinder axes
normal_alpha	Angle between the two cylinder axes

Returns

Returns 0 on success; otherwise returns an error code AUBO_BUSY AUBO_BAD_STATE - AUBO_INVL_ARGUMENT -AUBO_BAD_STATE

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
moveIntersection(self: pyaubo_sdk.MotionControl, arg0: List[float], arg1: float, arg2: float, arg3: float, arg4: float, arg5: float, arg6: float) -> int
```

Lua function prototype

```
moveIntersection(poses: table, a: number, v: number, main_pipe_radius: number, sub_pipe_radius: number, normal_distance: number, normal_alpha: number) -> nil
```

8.6.5.2.46 moveJoint()

```
int arcs::common_interface::MotionControl::moveJoint (
    const std::vector< double > & q,
    double a,
    double v,
    double blend_radius,
    double duration)
Generated by Doxygen
```

Add joint motion

Parameters

q	Joint angles, unit: rad
a	Acceleration, unit: rad/s ² , The maximum value can be obtained via RobotConfig::getJointMaxAccelerations()
v	Velocity, unit: rad/s, The maximum value can be obtained via RobotConfig::getJointMaxSpeeds()
blend_radius	Blend radius, unit: m
duration	Execution time, unit: s Normally, when speed and acceleration are given, the trajectory duration can be determined. If you want to extend the trajectory duration, set the duration parameter. Duration can extend the trajectory time, but cannot shorten it. When duration = 0, it means the execution time is not specified, and the time will be calculated based on speed and acceleration. If duration is not 0, a and v values will be ignored.

Return values

0	Success
AUBO_BAD_STATE(1)	The current safety mode is not Normal, ReducedMode, or Recovery
AUBO_QUEUE_FULL(2)	Planning queue is full
AUBO_BUSY(3)	The previous instruction is being executed
-AUBO_BAD_STATE(-1)	Possible reasons include but are not limited to: thread has been detached, thread terminated, task_id not found, or the current robot mode is not Running
-AUBO_TIMEOUT(-4)	Interface call timeout
-AUBO_INVL_ARGUMENT(-5)	The length of q is less than the degrees of freedom of the current robot arm
AUBO_REQUEST_IGNORE(13)	The length of q is too short and there is no need to stop at that point

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
moveJoint(self: pyaubo_sdk.MotionControl, arg0: List[float], arg1: float, arg2: float, arg3: float, arg4: float) -> int
```

Lua function prototype

```
moveJoint(q: table, a: number, v: number, blend_radius: number, duration: number) -> nil
```

Lua example

```
moveJoint({-2.05177,-0.400292, 1.19625, 0.0285152, 1.57033, -2.28774},0.3,0.3,0,0)
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.MotionControl.moveJoint","params":[[-2.05177, -0.400292, 1.19625, 0.0285152, 1.57033, -2.28774],0.3,0.3,0,0],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":0}
```

8.6.5.2.47 moveJointWithAxisGroup()

```
int arcs::common_interface::MotionControl::moveJointWithAxisGroup (
    const std::vector< double > & q,
    double a,
    double v,
    double blend_radius,
    double duration,
    const std::string & group_name,
    const std::vector< double > & extq)
```

Synchronous motion of robot and external axes

Parameters

group_name	
q	
a	
v	
blend_radius	
duration	

Returns

8.6.5.2.48 moveLine()

Generated by Doxygen

```
int arcs::common_interface::MotionControl::moveLine (
    const std::vector< double > & pose,
    double a,
```

Parameters

pose	Target pose
a	Acceleration (if the position change is less than 1mm and the posture change is greater than 1e-4 rad, this acceleration is treated as angular acceleration in rad/s ² ; otherwise, as linear acceleration in m/s ²). The maximum value can be obtained via RobotConfig::getTcpMaxAccelerations() .
v	Velocity (if the position change is less than 1mm and the posture change is greater than 1e-4 rad, this velocity is treated as angular velocity in rad/s; otherwise, as linear velocity in m/s). The maximum value can be obtained via RobotConfig::getTcpMaxSpeeds() .
blend_radius	Blend radius, unit: m, value between 0.001 and 1
duration	Execution time, unit: s Normally, when speed and acceleration are given, the trajectory duration can be determined. If you want to extend the trajectory duration, set the duration parameter. Duration can extend the trajectory time, but cannot shorten it. When duration = 0, it means the execution time is not specified, and the time will be calculated based on speed and acceleration. If duration is not 0, a and v values will be ignored.

Return values

0	Success
AUBO_BAD_STATE(1)	The current safety mode is not Normal, ReducedMode, or Recovery
AUBO_QUEUE_FULL(2)	Trajectory queue is full
AUBO_BUSY(3)	The previous instruction is being executed
-AUBO_BAD_STATE(-1)	Possible reasons include but are not limited to: thread has been detached, thread terminated, task_id not found, or the current robot mode is not Running
-AUBO_TIMEOUT(-4)	Interface call timeout
-AUBO_INVL_ARGUMENT(-5)	The length of the pose array is less than the degrees of freedom of the current robot arm
AUBO_REQUEST_IGNORE(13)	The length of the pose array is too short and there is no need to stop at that point

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
moveLine(self: pyaubo_sdk.MotionControl, arg0: List[float], arg1: float, arg2: float, arg3: float,
arg4: float) -> int
```

Lua function prototype

```
moveLine(pose: table, a: number, v: number, blend_radius: number, duration: number) -> nil
```

Lua example

```
moveLine({0.58815,0.0532,0.62391,2.46,0.479,1.619},1.3,1.0,0)
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.MotionControl.moveLine","params":[[0.58815,0.0532,0.62391,2.46,0.479,1.619],1.2,0.25,0,0],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":0}
```

8.6.5.2.49 moveLineWithAxisGroup()

```
int arcs::common_interface::MotionControl::moveLineWithAxisGroup (
    const std::vector< double > & pose,
    double a,
    double v,
    double blend_radius,
    double duration,
    const std::string & group_name,
    const std::vector< double > & extq)
```

Linear motion synchronized with external axes

Parameters

group_name	Name of the external axis group
pose	Target pose
a	Acceleration
v	Velocity
blend_radius	Blend radius
duration	Execution time

Parameters

name	Name of the cached path
------	-------------------------

Returns

Returns 0 on success; otherwise returns an error code -AUBO_INVL_ARGUMENT -AUBO_↵
BAD_STATE

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
movePathBuffer(self: pyaubo_sdk.MotionControl, arg0: str) -> int
```

Lua function prototype

```
movePathBuffer(name: string) -> nil
```

Lua example

```
movePathBuffer("traje_name")
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.MotionControl.movePathBuffer","params":["rec"],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":0}
```

8.6.5.2.51 moveProcess()

```
int arcs::common_interface::MotionControl::moveProcess (
    const std::vector< double > & pose,
    double a,
    double v,
    double blend_radius)
```

Add process motion

Parameters

pose	
a	
v	
blend_radius	

Returns

Returns 0 on success; otherwise returns an error code: AUBO_BAD_STATE AUBO_BUSY - AUBO_INVL_ARGUMENT -AUBO_BAD_STATE

Exceptions

arcs::common_interface::AuboException	
---	--

Lua function prototype

```
moveProcess(pose: table, a: number, v: number, blend_radius: number, duration: number) -> nil
```

Lua example

```
moveProcess({0.58815,0.0532,0.62391,2.46,0.479,1.619},1.2,0.25,0)
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.MotionControl.moveProcess","params":[[0.58815,0.0532,0.62391,2.46,0.479,1.619],1.2,0.25,0],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":0}
```

8.6.5.2.52 moveSpiral()

```
int arcs::common_interface::MotionControl::moveSpiral (
    const SpiralParameters & param,
    double blend_radius,
    double v,
    double a,
    double t)
```

Parameters

param	Encapsulated parameters
blend_radius	
v	
a	
t	

Returns

Returns 0 on success; otherwise returns an error code AUBO_BUSY AUBO_BAD_STATE - AUBO_INVL_ARGUMENT -AUBO_BAD_STATE

Exceptions

arcs::common_interface::AuboException	
---	--

JSON-RPC request example

```
{ "jsonrpc": "2.0", "method": "rob1.MotionControl.moveSpiral", "params": [{"spiral": 0.005, "helix": 0.005, "angle": 18.84, "plane": 0, "frame": [0, 0, 0, 0, 0]}, 0, 0.25, 1.2, 0], "id": 1 }
```

JSON-RPC response example

```
{ "id": 1, "jsonrpc": "2.0", "result": 0 }
```

8.6.5.2.53 moveSpline()

```
int arcs::common_interface::MotionControl::moveSpline (
    const std::vector< double > & q,
    double a,
    double v,
    double duration)
```

Perform spline interpolation in joint space

Parameters

q	Joint angles. If the input vector is of size 0, it indicates the end of the spline motion.
---	--

a	Acceleration, unit: rad/s ² . The maximum value can be obtained via RobotConfig::getJointMaxAccelerations() .
v	Velocity, unit: rad/s. The maximum value can be obtained via RobotConfig::getJointMaxSpeeds() .
duration	

Returns

Returns 0 on success; otherwise returns an error code: AUBO_BUSY AUBO_BAD_STATE - AUBO_BAD_STATE

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
moveSpline(self: pyaubo_sdk.MotionControl, arg0: List[float], arg1: float, arg2: float, arg3: float)
-> int
```

Lua function prototype

```
moveSpline(q: table, a: number, v: number, duration: number) -> nil
```

Lua example

```
moveSpline({-2.05177,-0.400292, 1.19625, 0.0285152, 1.57033, -2.28774},1.3,1.0,0)
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.MotionControl.moveSpline","params":[[0,0,0,0,0,0],1,1,0],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":0}
```

8.6.5.2.54 pathBufferAlloc()

```
int arcs::common_interface::MotionControl::pathBufferAlloc (
    const std::string & name,
    int type,
```

Generated by Doxygen

Create a new path point buffer

Parameters

name	Name of the path
type	Type of the path 1: toppra time-optimal path planning 2: cubic_spline (recorded trajectory) 3: Joint B-spline interpolation, at least three points
size	Buffer size

Returns

Returns AUBO_OK(0) on success

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
pathBufferAlloc(self: pyaubo_sdk.MotionControl, arg0: str, arg1: int, arg2: int) -> int
```

Lua function prototype

```
pathBufferAlloc(name: string, type: number, size: number) -> nil
```

Lua exmaple

```
pathBufferAlloc("traje_name",5,100)
```

JSON-RPC request example

```
{ "jsonrpc": "2.0", "method": "rob1.MotionControl.pathBufferAlloc", "params": ["rec", 2, 3], "id": 1 }
```

JSON-RPC response example

```
{ "id": 1, "jsonrpc": "2.0", "result": 0 }
```

8.6.5.2.55 pathBufferAppend()

```
int arcs::common_interface::MotionControl::pathBufferAppend (
    const std::string & name,
    const std::vector< std::vector< double > > & waypoints)
```

Add waypoints to the path buffer

Parameters

name	Name of the path buffer
waypoints	Waypoints

Returns

Returns 0 on success; otherwise returns an error code -AUBO_INVL_ARGUMENT -AUBO_BAD_STATE

Exceptions

<code>arcs::common_interface::AuboException</code>	
--	--

Python function prototype

```
pathBufferAppend(self: pyaubo_sdk.MotionControl, arg0: str, arg1: List[List[float]]) -> int
```

Lua function prototype

```
pathBufferAppend(name: string, waypoints: table) -> nil
```

Lua example

```
pathBufferAppend("traje_name",{ {-0.000000,0.000009,-0.000001,0.000002,0.000002,0.000000},{-0.000001,0.000010,-0.000002,0.000000,0.000003,0.000002}})
```

JSON-RPC request example

```
{ "jsonrpc": "2.0", "method": "rob1.MotionControl.pathBufferAppend", "params": [ "rec", [[ -0.000000, 0.000009, -0.000001, 0.000002, 0.000002, 0.000000 ], [ -0.000001, 0.000010, -0.000002, 0.000000, 0.000003, 0.000002 ] ] ], "id": 1 }
```

JSON-RPC response example

```
{ "id": 1, "jsonrpc": "2.0", "result": 0 }
```

8.6.5.2.56 pathBufferEval()

```
int arcs::common_interface::MotionControl::pathBufferEval (
    const std::string & name,
    const std::vector< double > & a,
    const std::vector< double > & v,
    double t)
```

Perform computation and optimization (time-consuming operations).

Parameters

name	Name of the path point buffer created by pathBufferAlloc
a	Joint acceleration limits, must match the robot DOF, unit: rad/s ²
v	Joint velocity limits, must match the robot DOF, unit: rad/s
t	Time When allocating memory with pathBufferAlloc, the type must be specified. According to the type in pathBufferAlloc: Type 1: t means motion duration Type 2: t means sampling interval Type 3: t means motion duration If t=0, the internal default time is used (recommended) If t!=0, t should be set to number_of_points * interval_between_points (interval >= 0.01)

Returns

Returns 0 on success; otherwise returns an error code -AUBO_INVL_ARGUMENT -AUBO_BAD_STATE

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
pathBufferEval(self: pyaubo_sdk.MotionControl, arg0: str, arg1: List[float], arg2: List[float], arg3: float) -> int
```

Lua function prototype

```
pathBufferEval(name: string, a: table, v: table, t: number) -> nil
```

Lua example

```
pathBufferEval("traje_name",{1,1,1,1,1,1},{1,1,1,1,1,1},0.02)
```

JSON-RPC request example

```
{ "jsonrpc": "2.0", "method": "rob1.MotionControl.pathBufferEval", "params": [ "rec", [1,1,1,1,1,1], [1,1,1,1,1,1], 0.02 ], "id": 1 }
```

JSON-RPC response example

```
{ "id": 1, "jsonrpc": "2.0", "result": 0 }
```

8.6.5.2.57 pathBufferFilter()

```
int arcs::common_interface::MotionControl::pathBufferFilter (
    const std::string & name,
    int order,
    double fd,
    double fs)
```

Joint space path filter

pathBufferFilter

Parameters

name	Name of the path buffer
order	Filter order (usually 2)
fd	Cutoff frequency, the smaller the smoother but with more delay (usually 3-20)
fs	Sampling frequency of discrete data (usually 20-500)

Returns

Returns 0 on success

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
pathBufferFilter(self: pyaubo_sdk.MotionControl, arg0: str, arg1: int, arg2: float, arg3: float) ->
int
```

Lua function prototype

```
pathBufferFilter(name: string, order: number, fd: number, fs:number) -> nil
```

Lua example

```
pathBufferFilter("traje_name",2,5,125)
```

8.6.5.2.58 pathBufferFree()

```
int arcs::common_interface::MotionControl::pathBufferFree (
    const std::string & name)
```

Release path buffer

Parameters

name	Name of the path buffer
------	-------------------------

Returns

Returns 0 on success; otherwise returns an error code -AUBO_INVL_ARGUMENT -AUBO_↵
BAD_STATE

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
pathBufferFree(self: pyaubo_sdk.MotionControl, arg0: str) -> int
```

Lua function prototype

```
pathBufferFree(name: string) -> nil
```

Lua example

```
pathBufferFree("traje_name")
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.MotionControl.pathBufferFree","params":["rec"],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":-5}
```

8.6.5.2.59 pathBufferList()

```
std::vector< std::string > arcs::common_interface::MotionControl::pathBufferList ()
```

List all cached path names

Returns

Returns all cached path names

Exceptions

<code>arcs::common_interface::AuboException</code>	
--	--

Python function prototype

```
pathBufferList(self: pyaubo_sdk.MotionControl) -> List[str]
```

Lua function prototype

```
pathBufferList() -> table
```

Lua example

```
Buffer_table = pathBufferList()
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.MotionControl.pathBufferList","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":[]}
```

8.6.5.2.60 pathBufferValid()

```
bool arcs::common_interface::MotionControl::pathBufferValid (
    const std::string & name)
```

Whether the buffer with the specified name is valid

The buffer must meet three conditions to be valid:

1. The buffer exists and memory has been allocated
2. The number of points passed into the buffer must be greater than or equal to the buffer size
3. pathBufferEval must be executed once

Parameters

name	Name of the buffer
------	--------------------

Returns

Returns true if valid; false otherwise

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
pathBufferValid(self: pyaubo_sdk.MotionControl, arg0: str) -> bool
```

Lua function prototype

```
pathBufferValid(name: string) -> boolean
```

Lua example

```
buffer_status = pathBufferValid("traje_name")
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.MotionControl.pathBufferValid","params":["rec"],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":false}
```

8.6.5.2.61 pathOffsetCoordinate()

```
int arcs::common_interface::MotionControl::pathOffsetCoordinate (
    int ref_coord)
```

Set the reference coordinate system for offset.

Only valid when type=1 in pathOffsetSet.

Parameters

ref_coord	Reference coordinate system: 0-base coordinate, 1-TCP
-----------	---

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
pathOffsetCoordinate(self: pyaubo_sdk.MotionControl, arg0: int) -> float
```

Lua function prototype

```
pathOffsetCoordinate(ref_coord: number) -> number
```

Lua example

```
num = pathOffsetCoordinate(0)
```

JSON-RPC request example

```
{"jsonrpc": "2.0", "method": "rob1.MotionControl.pathOffsetCoordinate", "params": [0], "id": 1}
```

JSON-RPC response example

```
{"id": 1, "jsonrpc": "2.0", "result": 0}
```

8.6.5.2.62 pathOffsetDisable()

```
int arcs::common_interface::MotionControl::pathOffsetDisable ()
```

Disable path offset

Returns

Returns 0 on success; otherwise returns an error code: AUBO_BAD_STATE AUBO_BUSY - AUBO_BAD_STATE

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
pathOffsetDisable(self: pyaubo_sdk.MotionControl) -> int
```

Lua function prototype

```
pathOffsetDisable() -> nil
```

Lua example

```
pathOffsetDisable()
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.MotionControl.pathOffsetDisable","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":0}
```

8.6.5.2.63 pathOffsetEnable()

```
int arcs::common_interface::MotionControl::pathOffsetEnable ()
```

Enable path offset

Returns

Returns 0 on success; otherwise returns an error code: AUBO_BUSY AUBO_BAD_STATE - AUBO_BAD_STATE

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
pathOffsetEnable(self: pyaubo_sdk.MotionControl) -> int
```

Lua function prototype

```
pathOffsetEnable() -> number
```

Lua example

```
num = pathOffsetEnable()
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.MotionControl.pathOffsetEnable","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":0}
```

8.6.5.2.64 pathOffsetLimits()

```
int arcs::common_interface::MotionControl::pathOffsetLimits (
    double v,
    double a)
```

Set the maximum speed and maximum acceleration for offset.

Only valid when type=1 in pathOffsetSet.

Parameters

v	Maximum speed
a	Maximum acceleration

Returns

Returns 0 on success; otherwise returns an error code: AUBO_BUSY AUBO_BAD_STATE - AUBO_INVL_ARGUMENT -AUBO_BAD_STATE

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
pathOffsetLimits(self: pyaubo_sdk.MotionControl, arg0: float, arg1: float) -> int
```

Lua function prototype

```
pathOffsetLimits(v: number, a: number) -> nil
```

Lua example

```
pathOffsetLimits(1.5,2.5)
```

8.6.5.2.65 pathOffsetSet()

```
int arcs::common_interface::MotionControl::pathOffsetSet (
    const std::vector< double > & offset,
    int type = 0)
```

Set path offset

Parameters

offset	Pose offset in each direction
type	Motion type 0-position planning 1-velocity planning

Returns

Returns 0 on success; otherwise returns error code AUBO_BAD_STATE AUBO_BUSY -AUBO_↵
_INVL_ARGUMENT -AUBO_BAD_STATE

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
pathOffsetSet(self: pyaubo_sdk.MotionControl, arg0: List[float], arg1: int) -> int
```

Lua function prototype

```
pathOffsetSet(offset: table, type: number) -> nil
```

Lua example

```
pathOffsetSet({ 0, 0, 0.1, 0, 0, 0 }, 0)
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.MotionControl.pathOffsetSet","params":[[0,0,0.01,0,0,0],0],"id":↵  
1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":0}
```

8.6.5.2.66 pathOffsetSupv()

Parameters

min	Maximum offset in the negative direction of the coordinate axis	
max	Maximum offset in the positive direction of the coordinate axis	
strategy	Monitoring strategy after reaching the maximum offset 1 - Saturation limit, i.e., maintain maximum pose	0 - Disable monitoring 2 - Protective stop

Returns

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
pathOffsetSupv(self: pyaubo_sdk.MotionControl, arg0: List[float], arg1: List[float], arg2: int) ->
int
```

Lua function prototype

```
pathOffsetSupv(min: table, max: table, strategy: number) -> number
```

Lua example

```
num = pathOffsetSupv({0,0,-0.2,0,0,0},{0,0,0.5,0,0,0},0)
```

8.6.5.2.67 resetTcpMaxLinearVelocity()

```
int arcs::common_interface::MotionControl::resetTcpMaxLinearVelocity ()
```

Reset TCP maximum linear velocity

Clear the TCP maximum linear velocity override set by setTcpMaxLinearVelocity

Returns

Returns 0 on success; error code on failure.

Exceptions

<code>arcs::common_interface::AuboException</code>	
--	--

Lua function prototype

```
resetTcpMaxLinearVelocity() -> number
```

Lua example

```
resetTcpMaxLinearVelocity() – Reset TCP maximum linear velocity to the value in the safety configuration
```

Python function prototype

```
resetTcpMaxLinearVelocity() -> int
```

Python example

```
resetTcpMaxLinearVelocity() – Reset TCP maximum linear velocity
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.MotionControl.resetTcpMaxLinearVelocity","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":0}
```

8.6.5.2.68 restoPath()

```
int arcs::common_interface::MotionControl::restoPath ()
```

```
restoPath
```

Returns

Returns 0 on success; otherwise returns an error code: AUBO_BAD_STATE AUBO_BUSY - AUBO_BAD_STATE

Exceptions

<code>arcs::common_interface::AuboException</code>	
--	--

Lua function prototype

```
restoPath() -> number
```

Lua example

```
num restoPath()
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.MotionControl.restoPath","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":0}
```

8.6.5.2.69 resumeMoveJoint()

```
int arcs::common_interface::MotionControl::resumeMoveJoint (
    const std::vector< double > & q,
    double a,
    double v,
    double duration)
```

Move to the pause point using joint motion.

Parameters

q	Joint angles, unit: rad
a	Acceleration, unit: rad/s ²
v	Velocity, unit: rad/s
duration	Execution time, unit: s Normally, when speed and acceleration are given, the trajectory duration can be determined. If you want to extend the trajectory duration, set the duration parameter. Duration can extend the trajectory time, but cannot shorten it. When duration = 0, it means the execution time is not specified, and the time will be calculated based on speed and acceleration. If duration is not 0, a and v values will be ignored.

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
resumeMoveJoint(self: pyaubo_sdk.MotionControl, arg0: List[float], arg1: float, arg2: float, arg3: float) -> int
```

Lua function prototype

```
resumeMoveJoint(q: table, a: number, v: number, duration: number) -> nil
```

Lua example

```
resumeMoveJoint({-2.05177,-0.400292, 1.19625, 0.0285152, 1.57033, -2.28774},1.3,1.0,0)
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.MotionControl.resumeMoveJoint","params":[[-2.05177, -0.400292, 1.19625, 0.0285152, 1.57033, -2.28774],0.3,0.3,0],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":-1}
```

8.6.5.2.70 resumeMoveLine()

```
int arcs::common_interface::MotionControl::resumeMoveLine (
    const std::vector< double > & pose,
    double a,
    double v,
    double duration)
```

Move to the pause point using linear motion.

Parameters

pose	Target pose
a	Acceleration, unit: m/s ²
v	Velocity, unit: m/s

duration	<p>Execution time, unit: s</p> <p>Normally, when speed and acceleration are given, the trajectory duration can be determined. If you want to extend the trajectory duration, set the duration parameter. Duration can extend the trajectory time, but cannot shorten it.</p> <p>When duration = 0, it means the execution time is not specified, and the time will be calculated based on speed and acceleration. If duration is not 0, a and v values will be ignored.</p>
----------	---

Returns

Returns 0 on success; otherwise returns error code -AUBO_INVL_ARGUMENT -AUBO_BAD↵_STATE

Exceptions

arcs::common_interface::AuboException

Python function prototype

```
resumeMoveLine(self: pyaubo_sdk.MotionControl, arg0: List[float], arg1: float, arg2: float, arg3: float) -> int
```

Lua function prototype

```
resumeMoveLine(pose: table, a: number, v: number, duration: number) -> nil
```

Lua example

```
resumeMoveLine({0.58815,0.0532,0.62391,2.46,0.479,1.619},1.2,0.25,0)
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.MotionControl.resumeMoveLine","params":[[0.58815,0.0532,0.62391,2.46,0.479,1.619],1.2,0.25,0],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":0}
```

8.6.5.2.71 resumeSpeedJoint()

```
int arcs::common_interface::MotionControl::resumeSpeedJoint (
```

Generated by Doxygen
 const std::vector< double > & qd,
 double a,
 double t)

Joint resumeSpeedJoint() following (const std::vector< double > & qd, double a, double t) for position, speed, acceleration, and time.

Parameters

qd	Target joint velocity, unit: rad/s
a	Main axis acceleration, unit: rad/s ²
t	Time required for the function to return, unit: s If t = 0, the function returns when the target velocity is reached; otherwise, the function returns after t seconds, regardless of whether the target velocity is reached. If the target velocity is not reached, it will decelerate to zero. If the target velocity is reached, it will move at a constant speed.

Returns

Returns 0 on success; otherwise returns error code AUBO_BUSY -AUBO_INVL_ARGUMENT -AUBO_BAD_STATE

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
resumeSpeedJoint(self: pyaubo_sdk.MotionControl, arg0: List[float], arg1: float, arg2: float) -> int
```

Lua function prototype

```
resumeSpeedJoint(q: table, a: number, t: number) -> nil
```

Lua example

```
resumeSpeedJoint({0.2,0,0,0,0,0},1.5,10)
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.MotionControl.resumeSpeedJoint","params":[[0.2,0,0,0,0,0],1.5,10],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":-1}
```

8.6.5.2.72 resumeSpeedLine()

```
int arcs::common_interface::MotionControl::resumeSpeedLine (
    const std::vector< double > & xd,
    double a,
    double t)
```

Parameters

xd	Tool velocity, unit: m/s
a	Tool position acceleration, unit: m/s^2
t	Time required for the function to return, unit: s If t = 0, the function returns when the target velocity is reached; otherwise, the function returns after t seconds, regardless of whether the target velocity is reached. If the target velocity is not reached, it will decelerate to zero. If the target velocity is reached, it will move at a constant speed.

Returns

Returns 0 on success; otherwise returns error code -AUBO_BAD_STATE

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
resumeSpeedLine(self: pyaubo_sdk.MotionControl, arg0: List[float], arg1: float, arg2: float) -> int
```

Lua function prototype

```
resumeSpeedLine(pose: table, a: number, t: number) -> nil
```

Lua example

```
resumeSpeedLine({0.25,0,0,0,0,0},1.2,100)
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.MotionControl.resumeSpeedLine","params":[[0.25,0,0,0,0,0],1.2,100],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":-1}
```

8.6.5.2.73 resumeStopJoint()

```
int arcs::common_interface::MotionControl::resumeStopJoint (
Generated by Doxygen
double acc)
```

Stop motion in joint space (used after moving to a safe position via resumeSpeedJoint following a collision during process execution)

Parameters

acc	Joint acceleration, unit: rad/s^2
-----	-----------------------------------

Returns

Returns 0 on success; otherwise returns an error code AUBO_BUSY AUBO_BAD_STATE - AUBO_INVL_ARGUMENT -AUBO_BAD_STATE

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
resumeStopJoint(self: pyaubo_sdk.MotionControl, arg0: float) -> int
```

Lua function prototype

```
resumeStopJoint(acc: number) -> nil
```

Lua example

```
resumeStopJoint(31)
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.MotionControl.resumeStopJoint","params":[31],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":-1}
```

8.6.5.2.74 resumeStopLine()

```
int arcs::common_interface::MotionControl::resumeStopLine (
    double acc,
    double acc_rot)
```

Stop motion in Cartesian space (used after moving to a safe position via resumeSpeedLine following a collision during process execution)

Parameters

acc	Position acceleration, unit: m/s ²
acc_rot	Orientation acceleration, unit: rad/s ²

Returns

Returns 0 on success; otherwise returns an error code AUBO_BUSY AUBO_BAD_STATE - AUBO_INVL_ARGUMENT -AUBO_BAD_STATE

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
resumeStopLine(self: pyaubo_sdk.MotionControl, arg0: float, arg1: float) -> int
```

Lua function prototype

```
resumeStopLine(acc: number, acc_rot: number) -> nil
```

Lua example

```
resumeStopLine(10,10)
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.MotionControl.resumeStopLine","params":[10,10],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":-1}
```

8.6.5.2.75 servoCartesian()

```
int arcs::common_interface::MotionControl::servoCartesian (
    const std::vector< double > & pose,
    double a,
    double v,
    double t,
    double lookahead_time,
    double gain)
Generated by Doxygen
```

Cartesian space servo

Currently, only pose and t parameters are available:

Parameters

pose	Pose, unit: m
a	Acceleration, unit: m/s^2
v	Velocity, unit: m/s
t	Execution time, unit: s The larger the t value, the slower the robot moves, and vice versa; The optimal value for this parameter is the interval time for continuous calls to the servoCartesian interface.
lookahead_time ↵	Lookahead time, unit: s Specifies the duration to move before the robot starts to decelerate, used to smooth the trajectory [0.03, 0.2]. When lookahead_time is less than one control cycle, the smaller it is, the greater the overshoot. The optimal value for this parameter is one control cycle.
gain	Proportional gain Proportional gain for tracking the target position [100, 200], used to control the smoothness and accuracy of the motion, The larger the proportional gain, the longer it takes to reach the target position, and the smaller the overshoot.

Return values

0	Success
AUBO_BAD_STATE(1)	The current safety mode is not Normal, ReducedMode, or Recovery
AUBO_QUEUE_FULL(2)	Planning queue is full
AUBO_BUSY(3)	The previous instruction is being executed
-AUBO_BAD_STATE(-1)	Possible reasons include but are not limited to: thread has been detached, thread terminated, task_id not found, or the current robot mode is not Running
-AUBO_TIMEOUT(-4)	Interface call timeout
-AUBO_INVL_ARGUMENT(-5)	Trajectory position or velocity out of range
-AUBO_REQUEST_IGNORE(-13)	Not in servo mode
-AUBO_IK_NO_CONVERGE(-23)	Inverse kinematics calculation does not converge, calculation error;
-AUBO_IK_OUT_OF_RANGE(-24)	Inverse kinematics calculation exceeds robot maximum limits;
AUBO_IK_CONFIG_DISMATCH(-25)	Inverse kinematics input configuration error;
-AUBO_IK_JACOBIAN_FAILED(-26)	Inverse kinematics Jacobian calculation failed;
-AUBO_IK_NO_SOLU(-27)	Analytical solution exists for the target point, but none meet the selection criteria;
-AUBO_IK_UNKOWN_ERROR(-28)	Inverse kinematics returned an unknown error type;

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
servoCartesian(self: pyaubo_sdk.MotionControl, arg0: List[float], arg1: float, arg2: float, arg3: float, arg4: float, arg5: float) -> int
```

Lua function prototype

```
servoCartesian(pose: table, a: number, v: number, t: number, lookahead_time: number, gain: number) -> nil
```

Lua example

```
servoCartesian({0.58712,-0.15775,0.48703,2.76,0.344,1.432},0,0,10,0,0)
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.MotionControl.servoCartesian","params":[[0.58712,-0.15775,0.48703,2.76,0.344,1.432],0,0,10,0,0],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":-13}
```

8.6.5.2.76 servoCartesianWithAxes()

```
int arcs::common_interface::MotionControl::servoCartesianWithAxes (
    const std::vector< double > & pose,
    const std::vector< double > & extq,
    double a,
    double v,
    double t,
    double lookahead_time,
    double gain)
```

Servo motion (with external axes), used for executing offline trajectories, pass-through user planned trajectories, etc.

The difference from servoJointWithAxes is that it receives Cartesian poses instead of joint angles (inverse kinematics is performed internally by the software).

Parameters

pose	
extq	
t	
smooth_scale	
delay_sacle	

Returns

8.6.5.2.77 servoJoint()

```
int arcs::common_interface::MotionControl::servoJoint (
    const std::vector< double > & q,
    double a,
    double v,
    double t,
    double lookahead_time,
    double gain)
```

Joint space servo

Currently only q and t parameters are available;

Parameters

q	Joint angles, unit: rad
a	Acceleration, unit: rad/s ²
v	Velocity, unit: rad/s
t	Execution time, unit: s The larger the t value, the slower the robot moves, and vice versa; The optimal value for this parameter is the interval time for continuous calls to the servoJoint interface.
lookahead_time	Lookahead time, unit: s Specifies the duration to move before the robot starts to decelerate, used to smooth the trajectory [0.03, 0.2]. When lookahead_time is less than one control cycle, the smaller it is, the greater the overshoot. The optimal value for this parameter is one control cycle.
gain	Proportional gain Proportional gain for tracking the target position [100, 200], used to control the smoothness and accuracy of the motion. The larger the proportional gain, the longer it takes to reach the target position, and the smaller the overshoot.

Return values

0	Success
AUBO_BAD_STATE(1)	The current safety mode is not Normal, ReducedMode, or Recovery
AUBO_QUEUE_FULL(2)	Planning queue is full
AUBO_BUSY(3)	The previous instruction is being executed
-AUBO_BAD_STATE(-1)	Possible reasons include but are not limited to: thread has been detached, thread terminated, task_id not found, or the current robot mode is not Running
-AUBO_TIMEOUT(-4)	Interface call timeout
-AUBO_INVL_ARGUMENT(-5)	Trajectory position or velocity out of range
-AUBO_REQUEST_IGNORE(-13)	Not in servo mode

Exceptions

arcs::common_interface::AuboException

Python function prototype

```
servoJoint(self: pyaubo_sdk.MotionControl, arg0: List[float], arg1: float, arg2: float, arg3: float,
arg4: float, arg5: float) -> int
```

Lua function prototype

```
servoJoint(q: table, a: number, v: number, t: number, lookahead_time: number, gain: number)
-> nil
```

Lua example

```
servoJoint({0,0,0,0,0,0},0,0,10,0,0)
```

JSON-RPC request example

```
{"jsonrpc": "2.0", "method": "rob1.MotionControl.servoJoint", "params": [[0,0,0,0,0,0],0,0,10,0,0],
"id": 1}
```

JSON-RPC response example

```
{"id": 1, "jsonrpc": "2.0", "result": -13}
```

8.6.5.2.78 servoJointWithAxes()

```
int arcs::common_interface::MotionControl::servoJointWithAxes (
    const std::vector< double > & q,
    const std::vector< double > & extq,
    double a,
    double v,
    double t,
    double lookahead_time,
    double gain)
```

Servo motion (with external axes), used for executing offline trajectories, pass-through user planned trajectories, etc.

Parameters

q	
extq	
t	
smooth_scale	
delay_sacle	

Returns

8.6.5.2.79 setCirclePathMode()

```
int arcs::common_interface::MotionControl::setCirclePathMode (
    int mode)
```

Set circle path mode

Parameters

mode	<p>Mode</p> <p>0: Tool orientation remains unchanged relative to the arc path point coordinate system</p> <p>1: Orientation changes linearly, rotating around a fixed axis in space, from the start orientation to the target orientation</p> <p>2: Orientation changes from the start orientation, through the via point orientation, to the target orientation</p>
------	--

Returns

Returns 0 on success; otherwise returns an error code AUBO_BAD_STATE AUBO_BUSY - AUBO_INVL_ARGUMENT -AUBO_BAD_STATE

Exceptions

<code>arcs::common_interface::AuboException</code>	
--	--

Python function prototype

```
setCirclePathMode(self: pyaubo_sdk.MotionControl, arg0: int) -> int
```

Lua function prototype

```
setCirclePathMode(mode: number) -> nil
```

Lua example

```
setCirclePathMode(1)
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.MotionControl.setCirclePathMode","params":[0],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":0}
```

8.6.5.2.80 setConveyorTrackCompensate()

```
int arcs::common_interface::MotionControl::setConveyorTrackCompensate (
    int encoder_id,
    double comp)
```

Set the compensation value for conveyor tracking

Parameters

encoder_id	Reserved
comp	Conveyor compensation value

Returns

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
setConveyorTrackCompensate(self: pyaubo_sdk.MotionControl, arg0: int, arg1: float) -> int
```

Lua function prototype

```
setConveyorTrackCompensate(comp: number) -> number
```

JSON-RPC request example

```
{ "jsonrpc": "2.0", "method": "rob1.MotionControl.setConveyorTrackCompensate", "params": [0, 0.1], "id": 1 }
```

JSON-RPC response example

```
{ "id": 1, "jsonrpc": "2.0", "result": 0 }
```

8.6.5.2.81 setConveyorTrackEncoder()

```
int arcs::common_interface::MotionControl::setConveyorTrackEncoder (
    int encoder_id,
    int tick_per_meter)
```

Set conveyor encoder parameters

Parameters

encoder_id	Reserved
tick_per_meter	Linear conveyor: pulses per meter Circular conveyor: pulses per revolution

Returns

Exceptions

<code>arcs::common_interface::AuboException</code>	
--	--

Python function prototype

```
setConveyorTrackEncoder(self: pyaubo_sdk.MotionControl) -> int
```

Lua function prototype

```
setConveyorTrackEncoder(encoder_id: bumber,tick_per_meter: number) -> int
```

Lua example

```
num = setConveyorTrackEncoder(1,40000)
```

JSON-RPC request example

```
{ "jsonrpc": "2.0", "method": "rob1.MotionControl.setConveyorTrackEncoder", "params": [1,40000], "id": 1 }
```

JSON-RPC response example

```
{ "id": 1, "jsonrpc": "2.0", "result": 0 }
```

8.6.5.2.82 setConveyorTrackLimit()

```
int arcs::common_interface::MotionControl::setConveyorTrackLimit (
    int encoder_id,
    double limit)
```

Set the maximum distance limit for conveyor tracking

Parameters

encoder_id	Reserved
limit	Maximum distance limit for conveyor tracking, unit: meter

Returns

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
setConveyorTrackLimit(self: pyaubo_sdk.MotionControl, arg0: int, arg1: double) -> int
```

Lua function prototype

```
setConveyorTrackLimit(encoder_id: number, limit: number) -> int
```

JSON-RPC request example

```
{"jsonrpc": "2.0", "method": "rob1.MotionControl.setConveyorTrackLimit", "params": [0, 1.5], "id": 1}
```

JSON-RPC response example

```
{"id": 1, "jsonrpc": "2.0", "result": 0}
```

8.6.5.2.83 setConveyorTrackSensorOffset()

```
int arcs::common_interface::MotionControl::setConveyorTrackSensorOffset (
    int encoder_id,
    double offset)
```

Set the distance from the conveyor teaching position to the sync switch

Parameters

encoder_id	Reserved
offset	Distance from the conveyor teaching position to the sync switch, unit: meter

Returns

Exceptions

<code>arcs::common_interface::AuboException</code>	
--	--

Python function prototype

```
setConveyorTrackSensorOffset(self: pyaubo_sdk.MotionControl, arg0: int, arg1: double) -> int
```

Lua function prototype

```
setConveyorTrackSensorOffset(encoder_id: number, offset: number) -> int
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.MotionControl.setConveyorTrackSensorOffset","params":[0, 0.↵
2],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":0}
```

8.6.5.2.84 setConveyorTrackStartWindow()

```
int arcs::common_interface::MotionControl::setConveyorTrackStartWindow (
    int encoder_id,
    double window_min,
    double window_max)
```

Set the start window for conveyor tracking

Parameters

encoder_id	Reserved
min_window	Start position of the window, unit: meter
max_window	End position of the window, unit: meter

Returns

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
setConveyorTrackStartWindow(self: pyaubo_sdk.MotionControl, arg0: int, arg1: double, arg2: double) -> int
```

Lua function prototype

```
setConveyorTrackStartWindow(encoder_id: number, min_window: number, max_window: number) -> int
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.MotionControl.setConveyorTrackStartWindow","params":[0, 0.2, 1.0],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":0}
```

8.6.5.2.85 setConveyorTrackSyncSeparation()

```
int arcs::common_interface::MotionControl::setConveyorTrackSyncSeparation (
    int encoder_id,
    double distance,
    double time)
```

Set conveyor sync separation, used to filter out unwanted signals from the sync switch.

Parameters

encoder_id	Reserved
distance	The minimum distance to travel after a sync signal appears before accepting a new sync signal as a valid object, unit: meter
time	The minimum time to elapse after a sync signal appears before accepting a new sync signal as a valid object, unit: second

If distance and time are both set greater than 0, the setting takes effect. If both distance and time are set, distance takes precedence.

Returns

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
setConveyorTrackSyncSeparation(self: pyaubo_sdk.MotionControl, arg0: int, arg1: double, arg2: double) -> int
```

Lua function prototype

```
setConveyorTrackSyncSeparation(encoder__id: number, distance: number, time: number) -> int
```

JSON-RPC request example

```
{"jsonrpc": "2.0", "method": "rob1.MotionControl.setConveyorTrackSyncSeparation", "params": [0, 0.05, 0.2], "id": 1}
```

JSON-RPC response example

```
{"id": 1, "jsonrpc": "2.0", "result": 0}
```

8.6.5.2.86 setEndPath()

```
int arcs::common_interface::MotionControl::setEndPath ()
```

Set end path (terminate blending at current trajectory segment)

Explicitly specifies the currently executing trajectory segment as the boundary at which trajectory blending is terminated. The current trajectory segment will be executed completely, but no velocity or position blending will be applied with subsequent trajectory segments.

This interface does not trigger an emergency stop and does not interrupt the execution of the current trajectory segment. It only affects the blending behavior between trajectory segments.

Returns

Returns 0 on success; error code on failure.

Exceptions

arcs::common_interface::AuboException	
---	--

Lua function prototype

```
setEndPath() -> number
```

Lua example

```
setEndPath() - Terminate trajectory blending at the current segment
```

Python function prototype

```
setEndPath() -> int
```

Python example

```
setEndPath() # Terminate trajectory blending at the current segment
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.MotionControl.setEndPath","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":0}
```

8.6.5.2.87 setEqradius()

```
int arcs::common_interface::MotionControl::setEqradius (
    double eqradius)
```

Set the equivalent radius, in meters.

When using moveLine/moveCircle, the rotation angle of the end effector's pose is converted to an equivalent end position movement. The larger the value, the faster the posture rotation speed.

Parameters

eqradius	0 means only plan the movement, and the posture rotation follows the movement.
----------	--

Returns

Returns 0 on success; otherwise returns an error code: AUBO_BAD_STATE AUBO_BUSY - AUBO_INVL_ARGUMENT -AUBO_BAD_STATE

Exceptions

<code>arcs::common_interface::AuboException</code>	
--	--

JSON-RPC request example

```
{"jsonrpc": "2.0", "method": "rob1.MotionControl.setEqradius", "params": [0.8], "id": 1}
```

JSON-RPC response example

```
{"id": 1, "jsonrpc": "2.0", "result": 0}
```

Python function prototype

```
setEqradius(self: pyaubo_sdk.MotionControl, arg0: float) -> int
```

Lua function prototype

```
setEqradius(eqradius: number) -> number
```

Lua example

```
num = setEqradius(1)
```

8.6.5.2.88 setFuturePointSamplePeriod()

```
int arcs::common_interface::MotionControl::setFuturePointSamplePeriod (
    double sample_time)
```

Set the sampling interval for points on the future path

Parameters

<code>sample_time</code>	Sampling interval, unit: m/s ²
--------------------------	---

Returns

Returns 0 on success; otherwise returns an error code AUBO_BUSY AUBO_BAD_STATE - AUBO_INVL_ARGUMENT -AUBO_BAD_STATE

Exceptions

arcs::common_interface::AuboException	
---	--

8.6.5.2.89 setLookAheadSize()

```
int arcs::common_interface::MotionControl::setLookAheadSize (
    int size)
```

Set look-ahead segment size

1. For tasks requiring high speed smoothness, such as CNC machining, gluing, welding, etc., a longer look-ahead segment size can provide better speed planning and smoother motion.
2. For fast-response tasks such as picking, a shorter look-ahead segment size is preferred to improve response speed, but may cause large speed fluctuations due to insufficiently timely path feeding.

Returns

Returns 0 on success

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
setLookAheadSize(self: pyaubo_sdk.MotionControl, arg0: int) -> int
```

Lua function prototype

```
setLookAheadSize(eqradius: number) -> number
```

JSON-RPC request example

```
{"jsonrpc": "2.0", "method": "rob1.MotionControl.setLookAheadSize", "params": [1], "id": 1}
```

JSON-RPC response example

```
{"id": 1, "jsonrpc": "2.0", "result": 0}
```

8.6.5.2.90 setResumeStartPoint()

```
int arcs::common_interface::MotionControl::setResumeStartPoint (
    const std::vector< double > & q,
```

Parameters

q	Resume start position
move_type	0: Joint space, 1: Cartesian space
blend_radius	Blend radius
qdmax	Maximum joint speed (6 elements)
qddmax	Maximum joint acceleration (6 elements)
vmax	Maximum linear and angular speed for linear motion (2 elements)
amax	Maximum linear and angular acceleration for linear motion (2 elements)

Returns

Returns 0 on success; otherwise returns error code AUBO_BAD_STATE AUBO_BUSY -AUBO_BAD_STATE

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
setResumeStartPoint(self: pyaubo_sdk.MotionControl, arg0: List[float], arg1: int, arg2: float,
arg3: List[float], arg4: List[float], arg5: float, arg6: float) -> int
```

Lua function prototype

```
setResumeStartPoint(q: table, move_type: number, blend_radius: number, qdmax: table,
qddmax: table, vmax: number, amax: number) -> nil
```

Lua example

```
setResumeStartPoint({0,0,0,0,0,0},1,1,{0,0,0,0,0,0},{0,0,0,0,0,0},{1,1},{1,1})
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.MotionControl.setResumeStartPoint","params":[[0,0,0,0,0,0],1,1,[0,0,0,0,0,0],[0,0,0,0,0,0],1,1],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":0}
```

Parameters

enable	Whether to enable
--------	-------------------

Returns

Returns 0 on success; otherwise returns an error code: AUBO_BAD_STATE AUBO_BUSY - AUBO_BAD_STATE

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
setServoMode(self: pyaubo_sdk.MotionControl, arg0: bool) -> int
```

Lua function prototype

```
setServoMode(enable: boolean) -> nil
```

Lua example

```
setServoMode(true)
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.MotionControl.setServoMode","params":[true],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":0}
```

8.6.5.2.92 setServoModeSelect()

```
int arcs::common_interface::MotionControl::setServoModeSelect (
    int mode)
```

Set servo motion mode

Parameters

mode	0 - Exit servo mode 1 - Planning servo mode 2 - Transparent mode (direct send) 3 - Transparent mode (buffered)
------	--

Returns

Lua function prototype

```
setServoModeSelect(0) -> nil
```

Lua example

```
setServoModeSelect(0)
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.MotionControl.setServoModeSelect","params":[0],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":0}
```

8.6.5.2.93 setSpeedFraction()

```
int arcs::common_interface::MotionControl::setSpeedFraction (
    double fraction)
```

Dynamically adjust the robot's speed and acceleration ratio (0., 1.

]

Parameters

fraction	The ratio of robot speed and acceleration
----------	---

Returns

Returns 0 on success; otherwise returns an error code: AUBO_INVL_ARGUMENT AUBO_BUSY AUBO_BAD_STATE -AUBO_INVL_ARGUMENT -AUBO_BAD_STATE

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
setSpeedFraction(self: pyaubo_sdk.MotionControl, arg0: float) -> int
```

Lua function prototype

```
setSpeedFraction(fraction: number) -> nil
```

Lua example

```
setSpeedFraction(0.5)
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.MotionControl.setSpeedFraction","params":[0.8],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":0}
```

8.6.5.2.94 setTcpMaxLinearVelocity()

```
int arcs::common_interface::MotionControl::setTcpMaxLinearVelocity (
    double v)
```

Set TCP maximum linear velocity

Parameters

v	TCP maximum linear velocity value, unit is m/s
---	--

Exceptions

arcs::common_interface::AuboException	
---	--

Lua function prototype

```
setTcpMaxLinearVelocity(number) -> number
```

Lua example

```
setTcpMaxLinearVelocity(0.5) – Set TCP maximum linear velocity to 0.5 meters per second
```

Python function prototype

```
setTcpMaxLinearVelocity(v: float) -> int
```

Python example

```
setTcpMaxLinearVelocity(0.5) – Set TCP maximum linear velocity to 0.5
```

JSON-RPC request example

```
{“jsonrpc”:”2.0”,”method”:”rob1.MotionControl.setTcpMaxLinearVelocity”,”params”:[0.5],”id”:1}
```

JSON-RPC response example

```
{”id”:1,”jsonrpc”:”2.0”,”result”:0}
```

8.6.5.2.95 setTimeOptimalEnable()

```
int arcs::common_interface::MotionControl::setTimeOptimalEnable (
    bool enable)
```

Parameters

enable	
--------	--

Exceptions

arcs::common_interface::AuboException	
---	--

Lua function prototype

```
setTimeOptimalEnable\(\) -> number
```

Lua example

```
setTimeOptimalEnable(true)
```

JSON-RPC request example

```
{“jsonrpc”:”2.0”,”method”:”rob1.MotionControl.setTimeOptimalEnable”,”params”:[true],”id”:1}
```

JSON-RPC response example

```
{”id”:1,”jsonrpc”:”2.0”,”result”:0}
```

8.6.5.2.96 setWorkObjectHold()

```
int arcs::common_interface::MotionControl::setWorkObjectHold (
    const std::string & module_name,
    const std::vector< double > & mounting_pose)
```

Specify the name and mounting position when the workpiece is installed on the end of another robot or external axis.

Note

Not implemented yet

Parameters

module_name	Name of the control module
mounting_pose	Relative position of the grasp, if it is a robot, it is relative to the robot's end center point (not the TCP point)

Returns

Returns 0 on success; otherwise returns an error code: AUBO_BAD_STATE AUBO_BUSY - AUBO_INVL_ARGUMENT -AUBO_BAD_STATE

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
setWorkObjectHold(self: pyaubo_sdk.MotionControl, arg0: str, arg1: List[float]) -> int
```

Lua function prototype

```
setWorkObjectHold(module_name: string, mounting_pose: table) -> nil
```

Lua example

```
setWorkObjectHold("object",{0.2,0.1,-0.4,3.14,0,-1.57})
```

8.6.5.2.97 speedFractionCritical()

```
int arcs::common_interface::MotionControl::speedFractionCritical (
    bool enable)
```

Speed fraction critical section.

Parameters

enable	
--------	--

Returns

Returns 0 on success; otherwise returns an error code: AUBO_BAD_STATE AUBO_BUSY - AUBO_BAD_STATE

Exceptions

arcs::common_interface::AuboException	
---	--

Lua function prototype

[speedFractionCritical\(\)](#) -> nil

Lua example

```
speedFractionCritical(true)
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.MotionControl.speedFractionCritical","params":[true],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":0}
```

8.6.5.2.98 speedJoint()

```
int arcs::common_interface::MotionControl::speedJoint (
    const std::vector< double > & qd,
    double a,
    double t)
```

Joint space velocity following

When the robot arm has not yet reached the target velocity, giving a new target velocity will cause the robot arm to immediately reach the new target velocity.

Parameters

qd	Target joint velocity, unit: rad/s
a	Main axis acceleration, unit: rad/s ²
t	Time required for the function to return, unit: s If t = 0, the function returns when the target velocity is reached; otherwise, the function returns after t seconds, regardless of whether the target velocity is reached. If the target velocity is not reached, it will decelerate to zero. If the target velocity is reached, it will move at a constant speed.

Return values

0	Success
AUBO_BAD_STATE(1)	The current safety mode is not Normal, ReducedMode, or Recovery
AUBO_BUSY(3)	The previous instruction is being executed
-AUBO_BAD_STATE(-1)	Possible reasons include but are not limited to: thread has been detached, thread terminated, task_id not found, or the current robot mode is not Running
-AUBO_TIMEOUT(-4)	Interface call timeout
-AUBO_INVL_ARGUMENT(-5)	The length of the qd array is less than the degrees of freedom of the current robot arm

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
speedJoint(self: pyaubo_sdk.MotionControl, arg0: List[float], arg1: float, arg2: float) -> int
```

Lua function prototype

```
speedJoint(qd: table, a: number, t: number) -> nil
```

Lua example

```
speedJoint({0.2,0,0,0,0,0}, 1.5,10)
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.MotionControl.speedJoint","params":[[0.2,0,0,0,0,0],1.5,100]↵,"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":0}
```

8.6.5.2.99 speedLine()

```
int arcs::common_interface::MotionControl::speedLine (
    const std::vector< double > & xd,
    double a,
    double t)
```

Cartesian space velocity following

When the robot arm has not yet reached the target velocity, giving a new target velocity will cause the robot arm to immediately reach the new target velocity.

Parameters

xd	Tool velocity, unit: m/s
a	Tool position acceleration, unit: m/s^2
t	Time required for the function to return, unit: s If t = 0, the function returns when the target velocity is reached; otherwise, the function returns after t seconds, regardless of whether the target velocity is reached. If the target velocity is not reached, it will decelerate to zero. If the target velocity is reached, it will move at a constant speed.

Return values

0	Success
AUBO_BAD_STATE(1)	The current safety mode is not Normal, ReducedMode, or Recovery
AUBO_BUSY(3)	The previous instruction is being executed
-AUBO_BAD_STATE(-1)	Possible reasons include but are not limited to: thread has been detached, thread terminated, task_id not found, or the current robot mode is not Running
-AUBO_TIMEOUT(-4)	Interface call timeout

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
speedLine(self: pyaubo_sdk.MotionControl, arg0: List[float], arg1: float, arg2: float) -> int
```

Lua function prototype

```
speedLine(pose: table, a: number, t: number) -> nil
```

Lua example

```
speedLine({0.25,0,0,0,0},1.2,100)
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.MotionControl.speedLine","params":[[0.25,0,0,0,0],1.2,100],  
"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":0}
```

8.6.5.2.100 startMove()

```
int arcs::common_interface::MotionControl::startMove ()
```

StartMove is used to resume robot, external axes movement and belonging process after the movement has been stopped

- by the instruction StopMove.
- after execution of StorePath ... RestoPath sequence.
- after asynchronously raised movements errors, such as ERR_PATH_STOP or specific process error after handling in the ERROR handler.

Returns

Returns 0 on success; otherwise returns an error code: AUBO_BAD_STATE AUBO_BUSY - AUBO_BAD_STATE

Exceptions

arcs::common_interface::AuboException	
---	--

Lua function prototype

```
startMove() -> number
```

Lua example

```
num = startMove()
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.MotionControl.startMove","params":[],  
"id":1}
```

Parameters

acc	Joint acceleration, unit: rad/s^2
-----	-----------------------------------

Returns

Returns 0 on success; otherwise returns an error code AUBO_BUSY AUBO_BAD_STATE - AUBO_INVL_ARGUMENT -AUBO_BAD_STATE

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
stopJoint(self: pyaubo_sdk.MotionControl, arg0: float) -> int
```

Lua function prototype

```
stopJoint(acc: number) -> nil
```

Lua example

```
stopJoint(2)
```

JSON-RPC request example

```
{"jsonrpc": "2.0", "method": "rob1.MotionControl.stopJoint", "params": [31], "id": 1}
```

JSON-RPC response example

```
{"id": 1, "jsonrpc": "2.0", "result": 0}
```

8.6.5.2.102 stopLine()

```
int arcs::common_interface::MotionControl::stopLine (
    double acc,
    double acc_rot)
```

Stop motions in Cartesian space such as moveLine/moveCircle.

Parameters

acc	Tool acceleration, unit: m/s^2
acc_rot	

Returns

Returns 0 on success; otherwise returns an error code AUBO_BUSY AUBO_BAD_STATE - AUBO_INVL_ARGUMENT -AUBO_BAD_STATE

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
stopLine(self: pyaubo_sdk.MotionControl, arg0: float, arg1: float) -> int
```

Lua function prototype

```
stopLine(acc: number, acc_rot: number) -> nil
```

Lua example

```
stopLine(10,10)
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.MotionControl.stopLine","params":[10,10],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":0}
```

8.6.5.2.103 stopMove()

```
int arcs::common_interface::MotionControl::stopMove (
    bool quick,
    bool all_tasks)
```

StopMove is used to stop robot and external axes movements and any belonging process temporarily.

If the instruction StartMove is given then the movement and process resumes.

Generated by Doxygen

This instruction can, for example, be used in a trap routine to stop the robot temporarily when an interrupt occurs.

Parameters

quick	true: Stops the robot on the path as fast as possible. Without the optional parameter \Quick, the robot stops on the path, but the braking distance is longer (same as for normal Program Stop).
-------	--

Returns

Returns 0 on success; otherwise returns an error code: AUBO_BAD_STATE AUBO_BUSY - AUBO_BAD_STATE

Exceptions

arcs::common_interface::AuboException	
---	--

Lua function prototype

StopMove(quick: bool,all_tasks: bool) -> number

Lua example

```
num = StopMove(true,true)
```

8.6.5.2.104 storePath()

```
int arcs::common_interface::MotionControl::storePath (
    bool keep_sync)
```

storePath

Parameters

keep_sync	
-----------	--

Returns

Returns 0 on success; otherwise returns an error code: AUBO_BAD_STATE AUBO_BUSY - AUBO_BAD_STATE

Exceptions

arcs::common_interface::AuboException	
---	--

Lua function prototype

```
storePath(keep_sync: bool) -> number
```

Lua example

```
num = storePath()
```

8.6.5.2.105 trackCartesian()

```
int arcs::common_interface::MotionControl::trackCartesian (
    const std::vector< double > & pose,
    double t,
    double smooth_scale,
    double delay_sacle)
```

Tracking motion, used for executing offline trajectories or passing through user-planned trajectories, etc.

The difference from trackJoint is that it receives Cartesian poses instead of joint angles (inverse kinematics is performed internally by the software).

Parameters

pose	
t	
smooth_scale	
delay_sacle	

Returns

Lua function prototype

```
trackCartesian(pose: table,t: number,smooth_scale: number,delay_sacle: number) -> nil
```

Lua example

```
trackCartesian({0.58712,-0.15775,0.48703,2.76,0.344,1.432},0.01,0.5,1)
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.MotionControl.trackCartesian","params":[[0.58712,-0.15775,0.48703,2.76,0.344,1.432],0.01,0.5,1],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":0}
```

8.6.5.2.106 trackJoint()

```
int arcs::common_interface::MotionControl::trackJoint (
    const std::vector< double > & q,
    double t,
    double smooth_scale,
    double delay_sacle)
```

Tracking motion, used for executing offline trajectories or passing through user-planned trajectories, etc.

Parameters

q	
smooth_scale	
delay_sacle	

Returns

Lua function prototype

```
trackJoint(q: table,t: number,smooth_scale: number,delay_sacle: number) -> nil
```

Lua example

```
trackJoint({0,0,0,0,0,0},0.01,0.5,1)
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.MotionControl.trackJoint","params":[[0,0,0,0,0,0],0.01,0.5,1],
```

```
"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":0}
```

8.6.5.2.107 weaveEnd()

```
int arcs::common_interface::MotionControl::weaveEnd ()
```

End weaving

Returns

Returns 0 on success; otherwise returns an error code AUBO_BUSY AUBO_BAD_STATE -
Generated by Doxygen AUBO_BAD_STATE

Exceptions

arcs::common_interface::AuboException	
---	--

JSON-RPC request example

```
{"jsonrpc": "2.0", "method": "rob1.MotionControl.weaveEnd", "params": [], "id": 1}
```

JSON-RPC response example

```
{"id": 1, "jsonrpc": "2.0", "result": 0}
```

8.6.5.2.108 weaveStart()

```
int arcs::common_interface::MotionControl::weaveStart (
    const std::string & params)
```

Start weaving: between weaveStart and weaveEnd, moveLine/moveCircle/moveProcess follows params.

Warning

Start [RuntimeMachine](#) before calling weaveStart.

Parameters

params	Json string.
--------	--------------

Since v0.29

- "type": <string> — Waveform: "SINE" | "SPIRAL" | "TRIANGLE" | "TRAPEZOIDAL" (case-sensitive).
- "step": <number> — Arc-length sampling step, m.
- "amplitude": [<number>, <number>] — Left/right weave amplitude, m.
- "hold_distance": [<number>, <number>] — Dwell distance at peak amplitude along path, m.
- "hold_time": [<number>, <number>] — Dwell time at peak amplitude, s.
- "angle": <number> — Angle to normal plane, rad.
- "direction": <integer> — Initial direction: 0=above path, 1=below path.
- "movep_uniform_vel": <bool> — Uniform-velocity planning (boolean value).

Since v0.31

- Same as v0.29.

New/changed in v0.32

- "frequency": <number> — Weave frequency, Hz (range [0.1, 5]).
- "ori_weave_range": [<number>,<number>,<number>] — Attitude weave range [x,y,z], rad; often used with Archimedean spiral for force-guided hole finding.
- "ori_weave_frequency": [<number>,<number>,<number>] — Attitude weave frequency \leftrightarrow [x,y,z], m^{-1} (per unit arc; range [0, 1]).
- "adjust_cycle_num": <integer> — Number of transition cycles.
- "azimuth": <number> — Waveform azimuth, rad.
- "ridge_height": <number> — Center raise height, m.

Warning

hold_time is supported for SINE only and may be asymmetric.

You may relate frequency = vel / step when an external path speed vel ($\text{m} \cdot \text{s}^{-1}$) is available.

Note

- Two-element arrays are [left,right]; three-element arrays are attitude [x,y,z].
- All angles are in radians (rad).

Returns

Returns 0 on success; otherwise returns an error code:

- AUBO_BUSY
- AUBO_BAD_STATE
- -AUBO_INVL_ARGUMENT
- -AUBO_BAD_STATE

Exceptions

<code>arcs::common_interface::AuboException</code>	
--	--

Example v0.29/v0.31

```
params = {
    "type": "SINE",
    "step": 0.005,
    "amplitude": [0.01, 0.01],
    "hold_distance": [0.001, 0.001],
    "hold_time": [0, 0],
    "angle": 0,
    "direction": 0,
    "movep_uniform_vel": false
}
```

Example v0.32

```

params = {
    "type" : "SINE", // "SINE" "SPIRAL" "TRIANGLE" "TRAPEZOIDAL"
    "step" : 0.0, // frequency = vel / step (vel: m · s-1)
    "frequency" : 0.0, // [0.1, 5] Hz
    "amplitude" : [0.01, 0.01], // m
    "hold_distance" : [0.001, 0.001], // m
    "hold_time" : [0, 0], // s (SINE only)
    "angle" : 0, // rad
    "direction" : 0,
    "ori_weave_range" : [0.001, 0.001, 0.001], // rad
    "ori_weave_frequency" : [0.001, 0.001, 0.001], // m-1
    "adjust_cycle_num" : 0,
    "azimuth" : 0, // rad
    "ridge_height" : 0 // m
}

```

Python function prototype

```
weaveStart(self: pyaubo_sdk.MotionControl, arg0: str) -> int
```

Python example

```

robot_name = rpc_cli.getRobotNames()[0]
robot_interface = rpc_cli.getRobotInterface(robot_name)
robot_interface.getMotionControl().weaveStart(params)

```

Lua function prototype

```
weaveStart(params: string) -> nil
```

Lua example (v0.29/0.31)

```
weaveStart("{\"type\":\"SINE\", \"step\":0.005, \"amplitude\":[0.01,0.01], \"hold_distance\":[0.001,0.001], \"hold_time\":[0,0], \"angle\":0, \"direction\":0, \"azimuth\":0, \"ridge_height\":0, \"ori_weave_range\":[0.001,0.001,0.001], \"ori_weave_frequency\":[0.001,0.001,0.001], \"adjust_cycle_num\":0}")
```

Lua example (v0.32)

```
weaveStart("{\"type\":\"SINE\", \"step\":0.0, \"frequency\":2.0, \"amplitude\":[0.01,0.01], \"hold_distance\":[0.001,0.001], \"hold_time\":[0,0], \"angle\":0, \"direction\":0, \"azimuth\":0, \"ridge_height\":0, \"ori_weave_range\":[0.001,0.001,0.001], \"ori_weave_frequency\":[0.001,0.001,0.001], \"adjust_cycle_num\":0}")
```

JSON-RPC request example (v0.29/0.31)

```
{ "jsonrpc": "2.0", "method": "rob1.MotionControl.weaveStart", "params": [{"type": "SINE", "step": 0.005, "amplitude": [0.01, 0.01], "hold_distance": [0.001, 0.001], "hold_time": [0, 0], "angle": 0, "direction": 0, "azimuth": 0, "ridge_height": 0, "ori_weave_range": [0.001, 0.001, 0.001], "ori_weave_frequency": [0.001, 0.001, 0.001], "adjust_cycle_num": 0}] }
```

JSON-RPC request example (v0.32)

```
{ "jsonrpc": "2.0", "method": "rob1.MotionControl.weaveStart", "params": [{"type": "SINE", "step": 0.0, "frequency": 2.0, "amplitude": [0.01, 0.01], "hold_distance": [0.001, 0.001], "hold_time": [0, 0], "angle": 0, "direction": 0, "azimuth": 0, "ridge_height": 0, "ori_weave_range": [0.001, 0.001, 0.001], "ori_weave_frequency": [0.001, 0.001, 0.001], "adjust_cycle_num": 0}] }
```

JSON-RPC response example

```
{ "id": 1, "jsonrpc": "2.0", "result": 0 }
```

Parameters

params	Json string used to define weaving parameters { "frequency": <num>, "amplitude": {<num>,<num>} }
--------	--

Returns

Returns 0 on success

Exceptions

arcs::common_interface::AuboException	
---	--

8.6.6 RobotAlgorithm ()

Interfaces related to robot algorithms

Collaboration diagram for RobotAlgorithm ():



Functions

- [ForceSensorCalibResult](#) [arcs::common_interface::RobotAlgorithm::calibrateTcpForceSensor](#) (const std::vector< std::vector< double > > &forces, const std::vector< std::vector< double > > &poses)
Force sensor calibration algorithm (three-point calibration method)
- [ForceSensorCalibResultWithError](#) [arcs::common_interface::RobotAlgorithm::calibrateTcpForceSensor2](#) (const std::vector< std::vector< double > > &forces, const std::vector< std::vector< double > > &poses)
Force sensor calibration algorithm (three-point calibration method)
- [ResultWithErrno](#) [arcs::common_interface::RobotAlgorithm::calibrateTcpForceSensor3](#) (const std::vector< std::vector< double > > &forces, const std::vector< std::vector< double > > &poses, const double &mass, const std::vector< double > &cog)
Force sensor offset calibration algorithm
- int [arcs::common_interface::RobotAlgorithm::payloadIdentify](#) (const std::string &data_file_no_↔ payload, const std::string &data_file_with_payload)
Payload identification algorithm interface based on current
- int [arcs::common_interface::RobotAlgorithm::payloadIdentify1](#) (const std::string &file_name)
New version of payload identification algorithm interface based on current
- int [arcs::common_interface::RobotAlgorithm::payloadCalculateFinished](#) ()

Whether payload identification calculation is finished

- [Payload arcs::common_interface::RobotAlgorithm::getPayloadIdentifyResult](#) ()

Get the result of payload identification

- [bool arcs::common_interface::RobotAlgorithm::frictionModelIdentify](#) (const std::vector< std::vector< double > > &q, const std::vector< std::vector< double > > &qd, const std::vector< std::vector< double > > &qdd, const std::vector< std::vector< double > > &temp)

Joint friction model identification algorithm interface

- [ResultWithErrno arcs::common_interface::RobotAlgorithm::calibWorkpieceCoordinatePara](#) (const std::vector< std::vector< double > > &q, int type)

Workpiece coordinate calibration algorithm interface (requires correct TCP offset set before calling)
Input multiple sets of joint angles and calibration type, output workpiece coordinate pose (relative to robot base)

- [ResultWithErrno arcs::common_interface::RobotAlgorithm::forwardDynamics](#) (const std::vector< double > &q, const std::vector< double > &torqs)

Forward dynamics

- [ResultWithErrno arcs::common_interface::RobotAlgorithm::forwardDynamics1](#) (const std::vector< double > &q, const std::vector< double > &torqs, const std::vector< double > &tcp_offset)

Forward dynamics based on the given TCP offset

- [ResultWithErrno arcs::common_interface::RobotAlgorithm::forwardKinematics](#) (const std::vector< double > &q)

Forward kinematics, based on the activated TCP offset (the most recently set via setTcpOffset) Input joint angles, output TCP pose

- [ResultWithErrno arcs::common_interface::RobotAlgorithm::forwardKinematics1](#) (const std::vector< double > &q, const std::vector< double > &tcp_offset)

Forward kinematics Input joint angles, output TCP pose

- [ResultWithErrno arcs::common_interface::RobotAlgorithm::forwardToolKinematics](#) (const std::vector< double > &q)

Forward kinematics (ignoring TCP offset)

- [ResultWithErrno1 arcs::common_interface::RobotAlgorithm::forwardKinematicsAll](#) (const std::vector< double > &q)

Forward kinematics, based on the activated TCP offset (the most recently set via setTcpOffset) Input joint angles, output links poses

- [ResultWithErrno arcs::common_interface::RobotAlgorithm::inverseKinematics](#) (const std::vector< double > &qnear, const std::vector< double > &pose)

Inverse kinematics Input TCP pose and reference joint angles, output joint angles

- [ResultWithErrno arcs::common_interface::RobotAlgorithm::inverseKinematics1](#) (const std::vector< double > &qnear, const std::vector< double > &pose, const std::vector< double > &tcp_offset)

Inverse kinematics Input TCP pose and reference joint angles, output joint angles

- [ResultWithErrno1 arcs::common_interface::RobotAlgorithm::inverseKinematicsAll](#) (const std::vector< double > &pose)

Solve all inverse kinematics solutions based on the activated TCP offset

- [ResultWithErrno1 arcs::common_interface::RobotAlgorithm::inverseKinematicsAll1](#) (const std::vector< double > &pose, const std::vector< double > &tcp_offset)

Solve all inverse kinematics solutions based on the provided TCP offset

- [ResultWithErrno arcs::common_interface::RobotAlgorithm::inverseToolKinematics](#) (const std::vector< double > &qnear, const std::vector< double > &pose)

Inverse kinematics (ignoring TCP offset)

- [ResultWithErrno1 arcs::common_interface::RobotAlgorithm::inverseToolKinematicsAll](#) (const std::vector< double > &pose)

Inverse kinematics (ignoring TCP offset)

- [ResultWithErrno3 arcs::common_interface::RobotAlgorithm::getRobotConfiguration](#) (const std::vector< double > &q)

- Calculate and return the corresponding robot configuration based on input joint angles
- [ResultWithErrno](#) [arcs::common_interface::RobotAlgorithm::calcJacobian](#) (const std::vector< double > &q, bool base_or_end)
 - Calculate the Jacobian matrix at the robot end-effector
- std::vector< std::vector< double > > [arcs::common_interface::RobotAlgorithm::pathBlend3Points](#) (int type, const std::vector< double > &q_start, const std::vector< double > &q_via, const std::vector< double > &q_to, double r, double d)
 - Solve the blended trajectory points
- int [arcs::common_interface::RobotAlgorithm::generatePayloadIdentifyTraj](#) (const std::string &name, const [TrajConfig](#) &traj_conf)
 - Generate excitation trajectory for payload identification This interface internally calls pathBufferAppend The offline trajectory is stored in the buffer, and can be executed later via movePathBuffer
- int [arcs::common_interface::RobotAlgorithm::payloadIdentifyTrajGenFinished](#) ()
 - Whether payload identification trajectory generation is finished
- std::vector< std::vector< double > > [arcs::common_interface::RobotAlgorithm::pathMoveS](#) (const std::vector< std::vector< double > > &q_s, double d)
 - Solve the trajectory points for moveS
- [ResultWithErrno1](#) [arcs::common_interface::RobotAlgorithm::calibVibrationParams](#) (const std::vector< std::vector< double > > &q, const std::vector< std::vector< double > > &qd, const std::vector< std::vector< double > > &target_q, const std::vector< std::vector< double > > &target_qd, const std::vector< std::vector< double > > &target_qdd, const std::vector< double > &tool_offset)
- [ResultWithErrno1](#) [arcs::common_interface::RobotAlgorithm::calibVibrationParams1](#) (const std::string &record_cache_name, const std::vector< double > &tool_offset)
- int [arcs::common_interface::RobotAlgorithm::needVibrationRecalib](#) (const [VibrationRecalibrationParameter](#) ¶m1, const [VibrationRecalibrationParameter](#) ¶m2, double threshold)
- int [arcs::common_interface::RobotAlgorithm::validatePath](#) (int type, const std::vector< double > &start, double r1, const std::vector< double > &end, double r2, double d)
 - Validate the reachability of the robot's motion path from start point to end point

8.6.6.1 Detailed Description

Interfaces related to robot algorithms

8.6.6.2 Function Documentation

8.6.6.2.1 calcJacobian()

```
ResultWithErrno arcs::common_interface::RobotAlgorithm::calcJacobian (
    const std::vector< double > & q,
    bool base_or_end)
```

Calculate the Jacobian matrix at the robot end-effector

Parameters

q	Joint angles
base_or_end	Reference frame: base (or end-effector) true: described in base frame false: described in end-effector frame

Returns

Whether the Jacobian matrix is valid The first parameter of the return value is the Jacobian matrix for this configuration, the second is the error code. The error code list was updated after versions 0.28.1-rc.21 and 0.29.0-alpha.25. Previously, inverse kinematics errors returned 30082. The updated error codes are: 0 - Success -23 - Inverse kinematics calculation does not converge, calculation error -24 - Inverse kinematics calculation exceeds robot limits -25 - Inverse kinematics input configuration error -26 - Inverse kinematics Jacobian calculation failed -27 - Analytical solution exists but none satisfy the selection criteria -28 - Unknown error in inverse kinematics If the error code is not 0, the first parameter of the return value is the input reference joint angles qnear

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
calJacobian(self: pyaubo_sdk.RobotAlgorithm, arg0: List[float], arg1: bool) -> Tuple[List[float], int]
```

Lua function prototype

```
calJacobian(q: table, base_or_end: boolean) -> table
```

Lua example

```
calJ_result = calJacobian({0.58815,0.0532,0.62391,2.46,0.479,1.619},true)
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.RobotAlgorithm.calcJacobian","params":[[0.58815,0.0532,0.62391,2.46,0.479,1.619],true],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":[[0.20822779551242535,-0.5409416184208162,0.2019786999613013,0.061264982268770196,-0.026269884327316487,0.10131708699859962,0.26388933410019777,-0.36074292664199115,0.1346954733416397,0.04085636647597124,-0.07244204452918337,0.0708466286633346,0.0,0.10401808481666497,-0.12571344758923886,-0.07741290545882097,0.18818543519232858,0.04628646442706299,0.0,0.5548228314607867,-0.5548228314607868,0.5548228314607868,-0.7901273140338193,0.37230961532208007,0.0,-0.8319685244586092,0.8319685244586091,-0.8319685244586091,-0.5269197820578843,-0.8184088260676008,1.0,3.749399456654644e-33,-6.512048180336603e-18,1.0956823467534067e-16,-0.31313634553301894,0.43771285536682175],0]}
```

8.6.6.2.2 calibrateTcpForceSensor()

[ForceSensorCalibResult](#) [arcs::common_interface::RobotAlgorithm::calibrateTcpForceSensor](#) (

```
const std::vector< std::vector< double > > & forces,
const std::vector< std::vector< double > > & poses)
```

Generated by Doxygen

Parameters

force	Force data
q	Joint angles

Returns

Calibration result

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
calibrateTcpForceSensor(self: pyaubo_sdk.RobotAlgorithm, arg0: List[List[float]], arg1: List[List[
float]]) -> Tuple[List[float], List[float], float, List[float]]
```

Lua function prototype

```
calibrateTcpForceSensor(force: table, q: table) -> table
```

Lua example

```
cal_table = calibrateTcpForceSensor({10.0,10.0,10.0,-1.2,-1.2,-1.2}, {3.083,1.227,1.098,0.670,-1.
870,-0.397})
```

8.6.6.2.3 calibrateTcpForceSensor2()

```
ForceSensorCalibResultWithError arcs::common_interface::RobotAlgorithm::calibrateTcpForceSensor2 (
    const std::vector< std::vector< double > > & forces,
    const std::vector< std::vector< double > > & poses)
```

Force sensor calibration algorithm (three-point calibration method)

Parameters

forces	
poses	

Returns

force_offset, com, mass, angle error

Exceptions

arcs::common_interface::AuboException	
---	--

8.6.6.2.4 calibrateTcpForceSensor3()

ResultWithErrno arcs::common_interface::RobotAlgorithm::calibrateTcpForceSensor3 (

```

    const std::vector< std::vector< double > > & forces,
    const std::vector< std::vector< double > > & poses,
    const double & mass,
    const std::vector< double > & cog)

```

Force sensor offset calibration algorithm

Parameters

forces	Force data
poses	
m	Mass, unit: kg
cog	Center of gravity, unit: m, format (CoGx, CoGy, CoGz)

Returns

Calibration result

Exceptions

arcs::common_interface::AuboException	
---	--

8.6.6.2.5 calibVibrationParams()

ResultWithErrno1 arcs::common_interface::RobotAlgorithm::calibVibrationParams (

```

    const std::vector< std::vector< double > > & q,
    const std::vector< std::vector< double > > & qd,
    const std::vector< std::vector< double > > & target_q,
    const std::vector< std::vector< double > > & target_qd,
    const std::vector< std::vector< double > > & target_qdd,
    const std::vector< double > & tool_offset)

```

8.6.6.2.6 calibVibrationParams1()

[ResultWithErrno](#) arcs::common_interface::RobotAlgorithm::calibVibrationParams1 (
const std::string & record_cache_name,
const std::vector< double > & tool_offset)

8.6.6.2.7 calibWorkpieceCoordinatePara()

[ResultWithErrno](#) arcs::common_interface::RobotAlgorithm::calibWorkpieceCoordinatePara (
const std::vector< std::vector< double > > & q,
int type)

Workpiece coordinate calibration algorithm interface (requires correct TCP offset set before calling) Input multiple sets of joint angles and calibration type, output workpiece coordinate pose (relative to robot base)

Parameters

q	Joint angles
type	Calibration type

Returns

Calculation result (workpiece coordinate pose) and error code

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
calibWorkpieceCoordinatePara(self: pyaubo_sdk.RobotAlgorithm, arg0: List[List[float]], arg1: int)
-> Tuple[List[float], int]
```

Lua function prototype

```
calibWorkpieceCoordinatePara(q: table, type: number) -> table, number
```

Lua example

```
coord_pose, coord_num = calibWorkpieceCoordinatePara({3.083,1.227,1.098,0.670,-1.870,-0.4397},1)
```

8.6.6.2.8 forwardDynamics()

Parameters

q	Joint angles
torqs	

Returns

Calculation result and error code

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
forwardDynamics(self: pyaubo_sdk.RobotAlgorithm, arg0: List[float], arg1: List[float]) -> Tuple[
    List[float], int]
```

Lua function prototype

```
forwardDynamics(q: table, torqs: table) -> table, number
```

Lua example

```
Dynamics, fk_result = forwardDynamics({3.083,1.227,1.098,0.670,-1.870,-0.397},{0.0,0.0,0.0,0.0,0.0,0.0})
```

8.6.6.2.9 forwardDynamics1()

```
ResultWithErrno arcs::common_interface::RobotAlgorithm::forwardDynamics1 (
    const std::vector< double > & q,
    const std::vector< double > & torqs,
    const std::vector< double > & tcp_offset)
```

Forward dynamics based on the given TCP offset

Parameters

q	Joint angles
torqs	

tcp_offset	TCP offset
------------	------------

Returns

Calculation result and error code, same as forwardDynamics

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
forwardDynamics(self: pyaubo_sdk.RobotAlgorithm, arg0: List[float], arg1: List[float], arg2: List[
float]) -> Tuple[List[float], int]
```

Lua function prototype

```
forwardDynamics1(q: table, torqs: table, tcp_offset: table) -> table, number
```

Lua example

```
Dynamics, fk_result = forwardDynamics1({3.083,1.227,1.098,0.670,-1.870,-0.397},{0.0,0.0,0.0,0.0,0.0,0.0},
{0.0,0.13201,0.03879,0,0,0})
```

8.6.6.2.10 forwardKinematics()

```
ResultWithErrno arcs::common_interface::RobotAlgorithm::forwardKinematics (
    const std::vector< double > & q)
```

Forward kinematics, based on the activated TCP offset (the most recently set via setTcpOffset) Input joint angles, output TCP pose

Parameters

q	Joint angles
---	--------------

Returns

TCP pose and whether the result is valid The first parameter of the return value is the forward kinematics result, the second is the error code. Error codes are as follows: 0 - Success -1 - The status of the robot is incorrect (not initialized yet, you can try calling it again) -5 - The input joint angles is invalid (dimension error)

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
forwardKinematics(self: pyaubo_sdk.RobotAlgorithm, arg0: List[float]) -> Tuple[List[float], int]
```

Lua function prototype

```
forwardKinematics(q: table) -> table, number
```

Lua example

```
pose, fk_result = forwardKinematics({3.083,1.227,1.098,0.670,-1.870,-0.397})
```

JSON-RPC request example

```
{ "jsonrpc": "2.0", "method": "rob1.RobotAlgorithm.forwardKinematics", "params": [[3.083688522170976, 1.2273215976885394, 1.098072739631141, 0.6705738810610149, -1.870715392248607, -0.39708546603119627], "id": 1 }
```

JSON-RPC response example

```
{ "id": 1, "jsonrpc": "2.0", "result": [[0.7137448715395925, 0.08416057568819092, 0.6707994191515292, 2.4599818776908724, 0.4789772388601265, 1.6189630435878408], 0] }
```

8.6.6.2.11 forwardKinematics1()

ResultWithErrno arcs::common_interface::RobotAlgorithm::forwardKinematics1 (

```
const std::vector< double > & q,
```

```
const std::vector< double > & tcp_offset)
```

Forward kinematics Input joint angles, output TCP pose

Parameters

q	Joint angles
tcp_offset	TCP offset

Returns

TCP pose and whether the result is valid The first parameter of the return value is the forward kinematics result, the second is the error code. Error codes are as follows: 0 - Success 1 - The status of the robot is incorrect (not initialized yet, you can try calling it again) -5 - The input joint angles or tcp offset is invalid (dimension error)

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
forwardKinematics(self: pyaubo_sdk.RobotAlgorithm, arg0: List[float], arg1: List[float]) ->
Tuple[List[float], int]
```

Lua function prototype

```
forwardKinematics1(q: table, tcp_offset: table) -> table, number
```

Lua example

```
pose, fk_result = forwardKinematics1({3.083,1.227,1.098,0.670,-1.870,-0.397},{0.0,0.13201,0.03879,0,0,0})
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.RobotAlgorithm.forwardKinematics1","params":[[3.083688522170976,1.2273215976885394,1.098072739631141,0.6705738810610149,-1.870715392248607,-0.39708546603119627],[0.0, 0.13201,0.03879,0,0,0]],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":[[0.7137636726659518,0.0837705432006433,0.6710022027216355,2.459981877690872,0.4789772388601267,1.6189630435878408],0]}
```

Since

0.24.1

8.6.6.2.12 forwardKinematicsAll()

[ResultWithErrno1](#) arcs::common_interface::RobotAlgorithm::forwardKinematicsAll (const std::vector< double > & q)

Forward kinematics, based on the activated TCP offset (the most recently set via setTcpOffset) Input joint angles, output links poses

Parameters

q	Joint angles
---	--------------

Returns

links poses and whether the result is valid The first parameter of the return value is the forward kinematics result, the second is the error code. Error codes are as follows: 0 - Success -1 - The status of the robot is incorrect (not initialized yet, you can try calling it again) -5 - The input joint angles is invalid (dimension error)

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
forwardKinematicsAll(self: pyaubo_sdk.RobotAlgorithm, arg0: List[float]) -> Tuple[List[List[float]], int]
```

Lua function prototype

```
forwardKinematicsAll(q: table) -> table, number
```

Lua example

```
pose, fk_result = forwardKinematicsAll({3.083,1.227,1.098,0.670,-1.870,-0.397})
```

JSON-RPC request example

```
{ "jsonrpc": "2.0", "method": "rob1.RobotAlgorithm.forwardKinematicsAll", "params": { "q": [-7.32945e-11, -0.261799, 1.74533, 0.436332, 1.5708, -2.14136e-10] }, "id": 1 }
```

JSON-RPC response example

```
{ "id": 1, "jsonrpc": "2.0", "result": [[ [0.0010004, 0.0, 0.122, 0.0, 0.0, 3.141592653589793], [0.0010003999910823934, -0.12166795404953117, 0.12205392567412496, 1.5690838552993878, -1.3089969602251492, 0.0016541204887245322], [0.10659809449330016, -0.12166124765791932, 0.5161518260534821, -1.5718540206872664, 0.43633111081725523, -0.002370419930605744], [0.4482255342315581, -0.1226390142692, 0.3568490758201224, -0.000788980482890453, 1.5665204619271216, -1.5719618592940798], [0.5519603828181741, -0.12282266347465487, 0.35639753697117343, 1.5707938201843654, 0.0042721909279596635, 1.569044569428874], [0.5513243939877184, -0.12401224959696328, 0.2615193425222568, -3.1349118101994096, 0.004272190926529066, 1.569044569643007], [0.7494883076798231, -0.025647479212994567, -0.04023458911333461, -3.1349118101994096, 0.004272190926529066, 1.569044569643007] ], [0] ] }
```

Parameters

q	Joint angles
---	--------------

Returns

Flange center pose and whether the result is valid The first parameter of the return value is the forward kinematics result, the second is the error code. Error codes are as follows: 0 - Success -1 - The status of the robot is incorrect (not initialized yet, you can try calling it again) -5 - The input joint angles is invalid (dimension error)

Lua function prototype

```
forwardToolKinematics(q: table) -> table, number
```

Lua example

```
pose, fk_result = forwardToolKinematics({3.083,1.227,1.098,0.670,-1.870,-0.397})
```

Exceptions

arcs::common_interface::AuboException	
---	--

JSON-RPC request example

```
{ "jsonrpc": "2.0", "method": "rob1.RobotAlgorithm.forwardToolKinematics", "params": [[3.083688522170976, 1.2273215976885394, 1.098072739631141, 0.6705738810610149, -1.870715392248607, -0.39708546603119627], "id": 1 }
```

JSON-RPC response example

```
{ "id": 1, "jsonrpc": "2.0", "result": [[0.5881351149440136, 0.05323734739426938, 0.623922550656701, 2.4599818776908724, 0.4789772388601265, 1.6189630435878408], 0] }
```

8.6.6.2.14 frictionModelIdentify()

```
bool arcs::common_interface::RobotAlgorithm::frictionModelIdentify (
    const std::vector< std::vector< double > > & q,
    const std::vector< std::vector< double > > & qd,
    const std::vector< std::vector< double > > & qdd,
    const std::vector< std::vector< double > > & temp)
```

Joint friction model identification algorithm interface

Parameters

q	Joint positions
qd	Joint velocities
qdd	Joint accelerations
temp	Joint temperatures

Returns

Whether identification succeeded

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
frictionModelIdentify(self: pyaubo_sdk.RobotAlgorithm, arg0: List[List[float]], arg1: List[List[←
[float]], arg2: List[List[float]], arg3: List[List[float]]) -> bool
```

Lua function prototype

```
frictionModelIdentify(q: table, qd: table, qdd: table, temp: table) -> boolean
```

Lua example

```
Identify_result = frictionModelIdentify({3.083,1.227,1.098,0.670,-1.870,-0.397}, {10.0,10.0,10.←
0,10.0,10.0,10.0},{20.0,20.0,20.0,20.0,20.0,20.0},{30.0,30.0,30.0,30.0,30.0,30.0})
```

8.6.6.2.15 generatePayloadIdentifyTraj()

```
int arcs::common_interface::RobotAlgorithm::generatePayloadIdentifyTraj (
    const std::string & name,
    const TrajConfig & traj_conf)
```

Generate excitation trajectory for payload identification This interface internally calls pathBufferAppend The offline trajectory is stored in the buffer, and can be executed later via movePathBuffer

Parameters

name	Trajectory name
traj_conf	Joint trajectory constraints traj_conf.move_axis: Moving axes Since users may not want large multi-joint movements during payload identification, it is recommended to use traj_conf.move_axis=LoadIdentifyMoveAxis::Joint_4_6; traj_conf.init_joint: Initial joint angles. To avoid singularity issues near joint 5 zero position, set $\text{abs}(\text{traj_conf.init_joint}[4]) \geq 0.3(\text{rad})$, preferably close to 1.57(rad). Other joints can be set arbitrarily. traj_conf.lower_joint_bound, traj_conf.upper_joint_bound: Joint angle limits, dimensions should match config.move_axis. Recommended: upper_joint_bound=2, lower_joint_bound=-2 config.max_velocity, config.max_acceleration: Joint velocity and acceleration limits, dimensions should match config.move_axis. For safety and driver performance, recommended: max_velocity=3, max_acceleration=5

Returns

Returns 0 on success; error code on failure AUBO_BUSY AUBO_BAD_STATE -AUBO_INVL↵_ARGUMENT -AUBO_BAD_STATE

Exceptions

arcs::common_interface::AuboException	
---	--

8.6.6.2.16 getPayloadIdentifyResult()

Payload [arcs::common_interface::RobotAlgorithm::getPayloadIdentifyResult \(\)](#)

Get the result of payload identification

Returns

Exceptions

arcs::common_interface::AuboException	
---	--

JSON-RPC request example

```
{"jsonrpc": "2.0", "method": "rob1.RobotAlgorithm.getPayloadIdentifyResult", "params": [], "id": 1}
```

JSON-RPC response example

```
{"id": 1, "jsonrpc": "2.0", "result": [0.0, [], [], []]}
```

8.6.6.2.17 getRobotConfiguration()

```
ResultWithErrno3 arcs::common_interface::RobotAlgorithm::getRobotConfiguration (
    const std::vector< double > & q)
```

Calculate and return the corresponding robot configuration based on input joint angles

The robot configuration consists of three dimensions of states, with the following definitions for each dimension:

- Shoulder direction: LEFT (Shoulder to the left) / RIGHT (Shoulder to the right)
- Elbow direction: UP (Elbow up) / DOWN (Elbow down)
- Wrist state: FLIP (Wrist flipped) / NOFLIP (Wrist not flipped)

The combination of the three dimensions forms 8 basic configurations (L/R + U/D + F/N). This interface parses the current configuration type from the input joint angles and returns its integer value corresponding to the RobotConfiguration enumeration and an error code (Note: The interface returns combined configuration values, e.g., LUF corresponds to 0, not the individual enumeration values of LEFT/UP/FLIP).

Parameters

q	Input joint angle array, 6 elements for 6-axis robot, unit: radians (rad)
---	---

Returns

Robot configuration result and error code (type: [ResultWithErrno3](#), i.e., std::tuple<int, int>):

- First int: Integer value corresponding to the RobotConfiguration enumeration, value range and meanings: -1 (NONE) - Invalid configuration 0 (LUF) - LEFT+UP+FLIP (Left shoulder, elbow up, wrist flipped) 1 (LUN) - LEFT+UP+NOFLIP (Left shoulder, elbow up, wrist not flipped) 2 (LDF) - LEFT+DOWN+FLIP (Left shoulder, elbow down, wrist flipped) 3 (LDN) - LEFT+DOWN+NOFLIP (Left shoulder, elbow down, wrist not flipped) 4 (RUF) - RIGHT+UP+FLIP (Right shoulder, elbow up, wrist flipped) 5 (RUN) - RIGHT+UP+NOFLIP (Right shoulder, elbow up, wrist not flipped) 6 (RDF) - RIGHT+DOWN+FLIP (Right shoulder, elbow down, wrist flipped) 7 (RDN) - RIGHT+DOWN+NOFLIP (Right shoulder, elbow down, wrist not flipped)
- Second int: Error code with the following meanings: 0 - Success: Configuration calculation completed, valid configuration value returned -1 - Abnormal robot state: Not initialized completely, try reinitializing before calling -5 - Invalid input parameters: Incorrect dimension of joint angle array (not 6 elements) or values out of reasonable range

Exceptions

arcs::common_interface::AuboException	Thrown when input parameters are illegal (e.g., empty array, wrong number of elements)
---	--

Python function prototype

```
getRobotConfiguration(self: pyaubo_sdk.RobotAlgorithm, arg0: List[float]) -> Tuple[int, int]
```

Lua function prototype

```
getRobotConfiguration(q: table) -> number, number
```

Lua example

```
- Input 6-axis joint angles (unit: rad) to get configuration and error code local joint_angles = {3.083,1.227,1.098,0.670,-1.870,-0.397} local config_val, err_code = getRobotConfiguration(joint_angles) if err_code == 0 then print("Robot configuration value: ", config_val) - Example output: 4 (corresponding to RUF, Right shoulder, elbow up, wrist flipped) else print("Failed to get configuration, error code: ", err_code) end
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.RobotAlgorithm.getRobotConfiguration","params":[[3.083688522170976,1.2273215976885394,1.098072739631141,0.6705738810610149,-1.870715392248607,-0.39708546603119627]],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":[4,0]}
```

8.6.6.2.18 inverseKinematics()

```
ResultWithErrno arcs::common_interface::RobotAlgorithm::inverseKinematics (
    const std::vector< double > & qnear,
    const std::vector< double > & pose)
```

Inverse kinematics Input TCP pose and reference joint angles, output joint angles

Parameters

qnear	Reference joint angles
pose	TCP pose

Returns

Joint angles and whether the inverse kinematics result is valid The first parameter of the return value is the inverse kinematics result, the second is the error code. Error codes are as follows: 0 - Success -1 - The status of the robot is incorrect (not initialized yet, you can try calling it again) -5 - The input reference joint angles or tcp pose is invalid (dimension error) -23 - Inverse kinematics calculation does not converge, calculation error -24 - Inverse kinematics calculation exceeds robot limits -25 - Inverse kinematics input configuration error -26 - Inverse kinematics Jacobian calculation failed -27 - Analytical solution exists but none satisfy the selection criteria -28 - Unknown error in inverse kinematics If the error code is not 0, the first parameter of the return value is the input reference joint angles qnear

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
inverseKinematics(self: pyaubo_sdk.RobotAlgorithm, arg0: List[float], arg1: List[float]) -> Tuple[
List[float], int]
```

Lua function prototype

```
inverseKinematics(qnear: table, pose: table) -> table, int
```

Lua example

```
joint,ik_result = inverseKinematics({0,0,0,0,0,0},{0.81665,-0.20419,0.43873,-3.135,0.004,1.569})
```

JSON-RPC request example

```
{ "jsonrpc": "2.0", "method": "rob1.RobotAlgorithm.inverseKinematics", "params": [[0,0,0,0,0,0],[0.71374,0.08417,0.6708,2.46,0.479,1.619]], "id": 1 }
```

JSON-RPC response example

```
{ "id": 1, "jsonrpc": "2.0", "result": [[3.083688522170976,1.2273215976885394,1.098072739631141,0.6705738810610149,-1.870715392248607,-0.39708546603119627],0] }
```

8.6.6.2.19 inverseKinematics1()

[ResultWithErrno](#) arcs::common_interface::RobotAlgorithm::inverseKinematics1 (

```
const std::vector< double > & qnear,
const std::vector< double > & pose,
const std::vector< double > & tcp_offset)
```

Inverse kinematics Input TCP pose and reference joint angles, output joint angles

Parameters

qnear	Reference joint angles
pose	TCP pose
tcp_offset	TCP offset

Returns

Joint angles and whether the result is valid, same as inverseKinematics

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
inverseKinematics1(self: pyaubo_sdk.RobotAlgorithm, arg0: List[float], arg1: List[float], arg2: List[float]) -> Tuple[List[float], int]
```

Lua function prototype

```
inverseKinematics1(qnear: table, pose: table, tcp_offset: table) -> table, int
```

Lua example

```
joint,ik_result = inverseKinematics1({0,0,0,0,0,0},{0.81665,-0.20419,0.43873,-3.135,0.004,1.569},{0.04,-0.035,0.1,0,0,0})
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.RobotAlgorithm.inverseKinematics1","params":[[0,0,0,0,0,0],[0.71374,0.08417,0.6708,2.46,0.479,1.619],[0.0, 0.13201,0.03879,0,0,0]],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":[[3.084454549595208,1.2278265883747776,1.0986586440159576,0.6708221281915528,-1.8712459848518375,-0.3965111476861782],0]}
```

8.6.6.2.20 inverseKinematicsAll()

[ResultWithErrno](#) arcs::common_interface::RobotAlgorithm::inverseKinematicsAll (const std::vector< double > & pose)

Solve all inverse kinematics solutions based on the activated TCP offset

Parameters

pose	TCP pose
------	----------

Returns

Joint angles and whether the result is valid The returned error code is the same as inverseKinematics

Exceptions

arcs::common_interface::AuboException	
---	--

JSON-RPC request example

```
{ "jsonrpc": "2.0", "method": "rob1.RobotAlgorithm.inverseKinematicsAll", "params": [[0.71374, 0.08417, 0.6708, 2.46, 0.479, 1.619]], "id": 1 }
```

JSON-RPC response example

```
{ "id": 1, "jsonrpc": "2.0", "result": [[[3.083688522170976, 1.2273215976885394, 1.098072739631141, 0.6705738810610149, -1.870715392248607, -0.39708546603119627], [3.081056801097411, 0.17985038037652645, -1.0991717292664145, -0.4806460200109001, -1.869182975312333, -0.402066016835411], [0.4090095277807992, -0.1623365054641728, 1.081775890307679, 0.26993250263224805, 0.9738255833642309, 0.000572556627720845], [0.4116449425067969, -1.1931664523907126, -1.0822709833775688, -0.8665964106161371, 0.9732141569888207, 0.006484919654891586]], 0] ] }
```

8.6.6.2.21 inverseKinematicsAll1()

```
ResultWithErrno1 arcs::common_interface::RobotAlgorithm::inverseKinematicsAll1 (
    const std::vector< double > & pose,
    const std::vector< double > & tcp_offset)
```

Solve all inverse kinematics solutions based on the provided TCP offset

Parameters

pose	TCP pose
tcp_offset	TCP offset

Returns

Joint angles and whether the result is valid, same as inverseKinematicsAll The returned error code is the same as inverseKinematics

Exceptions

arcs::common_interface::AuboException	
---	--

JSON-RPC request example

```
{ "jsonrpc": "2.0", "method": "rob1.RobotAlgorithm.inverseKinematicsAll1", "params": [[0.71374, 0.08417, 0.6708, 2.46, 0.479, 1.619], [0.0, 0.13201, 0.03879, 0, 0, 0]], "id": 1 }
```

JSON-RPC response example

```
{ "id": 1, "jsonrpc": "2.0", "result": [[[3.084454549595208, 1.2278265883747776, 1.0986586440159576, 0.6708221281915528, -1.8712459848518375, -0.3965111476861782], [3.0818224058231602, 0.17980369843203092, -1.0997576631122077, -0.48102131527371267, -1.8697135490338517, -0.40149459722060593], [0.40972960018231047, -0.16226026285489026, 1.0823403816496, 0.2700204411869427, 0.9734251963887868, 0.0012903686498106507], [0.41236549588802296, -1.193621392918341, -1.0828346680836718, -0.8671097369314354, 0.972815367289568, 0.007206851371073478]], 0] }
```

8.6.6.2.22 inverseToolKinematics()

```
ResultWithErrno arcs::common_interface::RobotAlgorithm::inverseToolKinematics (
    const std::vector< double > & qnear,
    const std::vector< double > & pose)
```

Inverse kinematics (ignoring TCP offset)

Parameters

qnear	Reference joint angles
pose	Flange center pose

Returns

Joint angles and whether the result is valid The first parameter of the return value is the inverse kinematics result, the second is the error code. Error codes are as follows: 0 - Success -1 - The status of the robot is incorrect (not initialized yet, you can try calling it again) -5 - The input reference joint angles or pose is invalid (dimension error)

Exceptions

<code>arcs::common_interface::AuboException</code>	
--	--

Lua function prototype

```
inverseToolKinematics(qnear: table, pose: table) -> table, int
```

Lua example

```
joint, ik_result = inverseToolKinematics({0,0,0,0,0,0},{0.58815,0.0532,0.62391,2.46,0.479,1.619})
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.RobotAlgorithm.inverseToolKinematics","params":[[0,0,0,0,0,0],
[0.58815,0.0532,0.62391,2.46,0.479,1.619]],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":[[3.083609363838651,1.22736129158332,1.098095443698268,0.6705395395487186,-1.8706605026855632,-0.39714507002376465],0]}
```

8.6.6.2.23 inverseToolKinematicsAll()

[ResultWithError](#) arcs::common_interface::RobotAlgorithm::inverseToolKinematicsAll (const std::vector< double > & pose)

Inverse kinematics (ignoring TCP offset)

Parameters

qnear	Reference joint angles
pose	Flange center pose

Returns

Joint angles and whether the result is valid

Exceptions

arcs::common_interface::AuboException	
---	--

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.RobotAlgorithm.inverseToolKinematicsAll","params":[[0.58815,0.0532,0.62391,2.46,0.479,1.619]],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":[[[3.083609363838651,1.22736129158332,1.098095443698268,0.6705395395487186,-1.8706605026855632,-0.39714507002376465], [3.0809781797426523,0.17987122696706134,-1.0991932793263717,-0.4807053707530958,-1.8691282890274434,-0.40212516672751814], [0.40892195618737215,-0.16235398607358653,1.081812753177426,0.27003586475871766,0.9738744130114284,0.00048462518316674287], [0.41155633414333076,-1.1932173012004512,-1.082306542045813,-0.8665312056504818,0.9732632365861417,0.0063958311601771175]],0]}
```

8.6.6.2.24 needVibrationRecalib()

```
int arcs::common_interface::RobotAlgorithm::needVibrationRecalib (
    const VibrationRecalibrationParameter & param1,
    const VibrationRecalibrationParameter & param2,
    double threshold)
```

8.6.6.2.25 pathBlend3Points()

```
std::vector< std::vector< double > > arcs::common_interface::RobotAlgorithm::pathBlend3Points (
    int type,
    const std::vector< double > & q_start,
    const std::vector< double > & q_via,
    const std::vector< double > & q_to,
    double r,
    double d)
```

Solve the blended trajectory points

Parameters

type	0-movej and movej 1-movej and movej 2-movej and movej 3-movej and movej
q_start	Start point before blending
q_via	Blending point
q_to	End point after blending
r	Blend radius at q_via
d	Sampling distance

Returns

Discrete trajectory points (x, y, z) of the blend segment at q_via in Cartesian space

Exceptions

arcs::common_interface::AuboException

Python function prototype

```
pathBlend3Points(self: pyaubo_sdk.RobotAlgorithm, arg0: int, arg1: List[float], arg2: List[float],
arg3: List[float], arg4: float, arg5: float) -> List[List[float]]
```

Lua function prototype

```
pathBlend3Points(type: number, q_start: table, q_via: table, q_to: table, r: number, d: number)
-> table, number
```

Lua example

```
q_via , num = pathBlend3Points(1,{0.58815,0.0532,0.62391,2.46,0.479,1.619},{0.0,-0.2618,1.↵
7453,0.4364,1.5711,0.0}, {0.3234,-0.5405,1.5403,0.5881,1.2962,0.7435},0.25,0.02)
```

8.6.6.2.26 pathMoveS()

```
std::vector< std::vector< double > > arcs::common_interface::RobotAlgorithm::pathMoveS (
    const std::vector< std::vector< double > > & qs,
    double d)
```

Solve the trajectory points for moveS

pathMoveS

Parameters

qs	Spline trajectory generation point set
d	Sampling distance

Returns

Exceptions

arcs::common_interface::AuboException	
---	--

8.6.6.2.27 payloadCalculateFinished()

```
int arcs::common_interface::RobotAlgorithm::payloadCalculateFinished ()
```

Whether payload identification calculation is finished

Returns

0 if finished; 1 if in progress; <0 if failed;

Exceptions

arcs::common_interface::AuboException	
---	--

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.RobotAlgorithm.payloadCalculateFinished","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":0}
```

8.6.6.2.28 payloadIdentify()

```
int arcs::common_interface::RobotAlgorithm::payloadIdentify (
    const std::string & data_file_no_payload,
    const std::string & data_file_with_payload)
```

[Payload](#) identification algorithm interface based on current

Requires collecting position, velocity, and current data when running the excitation trajectory without load, as well as with load.

Parameters

data_file_no_payload	File path of joint data when running the excitation trajectory without load (.csv format), 18 columns in total: 6 joint positions, 6 joint velocities, 6 joint currents
data_file_with_payload	File path of joint data when running the excitation trajectory with load (.csv format), 18 columns in total: 6 joint positions, 6 joint velocities, 6 joint currents

Returns

Identification result

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
payloadIdentify(self: pyaubo_sdk.RobotAlgorithm, arg0: List[List[float]], arg1: List[List[float]]) ->
Tuple[List[float], List[float], float, List[float]]
```

Lua function prototype

```
payloadIdentify(data_with_payload: table, data_with_payload: table) -> table
```

8.6.6.2.29 payloadIdentify1()

```
int arcs::common_interface::RobotAlgorithm::payloadIdentify1 (
    const std::string & file_name)
```

New version of payload identification algorithm interface based on current

Generated by Doxygen
Requires collecting at least three points of position, velocity, acceleration, current, temperature, end sensor data, and base data when running with load

Parameters

data	File path of joint data with load (.csv format), 42 columns in total, end sensor data and base data default to 0
------	--

Returns

Identification result

8.6.6.2.30 payloadIdentifyTrajGenFinished()

```
int arcs::common_interface::RobotAlgorithm::payloadIdentifyTrajGenFinished ()
```

Whether payload identification trajectory generation is finished

Returns

0 if finished; 1 if in progress; <0 if failed;

Exceptions

arcs::common_interface::AuboException	
---	--

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.RobotAlgorithm.payloadIdentifyTrajGenFinished","params":[],
"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":0}
```

8.6.6.2.31 validatePath()

```
int arcs::common_interface::RobotAlgorithm::validatePath (
    int type,
    const std::vector< double > & start,
    double r1,
    const std::vector< double > & end,
    double r2,
    double d)
```

Validate the reachability of the robot's motion path from start point to end point

Generated by Doxygen

This interface verifies whether the specified path has unreachable conditions such as out-of-limit, self-collision, singularity, etc., by sampling. It supports four path type validations: Joint-Joint, Joint-Pose, Pose-Joint, Pose-Pose.

Parameters

type	Path type identifier, specifying the data type of start and end points: 0 - Start: Joint angles, End: Joint angles 1 - Start: Pose, End: Pose
start	Start point data of the path <ul style="list-style-type: none"> • Joint angles • Pose
r1	Start point blending radius, unit: m (meters), used to smooth path sampling near the start point
end	End point data of the path, format is consistent with start (matching the type specified by type)
r2	End point blending radius, unit: m (meters), used to smooth path sampling near the end point
d	Path sampling interval, unit: m (meters). Smaller intervals improve verification accuracy but increase time consumption

Returns

Path reachability result code: 0 - Reachable: Path is normal, movement is possible -18 - Joint/ Cartesian space limits exceeded in the path -21 - Trajectory generation failed -22 - Self-collision of the robot body in the path -24 - Path passing through robot singular configuration -27 - The target point has a solution but exceeds joint limits configuration

Exceptions

arcs::common_interface::AuboException	Thrown when parameters are invalid (e.g., mismatched data length, values out of reasonable range, etc.)
---	---

Lua function prototype

```
validatePath(type, start, r1, end, r2, d) -> integer
```

Lua example

```
- Validate reachability of Joint-Joint path for 6-axis robot local start_joint = {0.0, 0.0, 90.0, 0.0, 90.0, 0.0} local end_joint = {30.0, 0.0, 90.0, 0.0, 90.0, 0.0} result = validatePath(0, start_joint, 0.01, end_joint, 0.01, 0.05)
```

Python function prototype

```
validatePath(type: int, start: list[float], r1: float, end: list[float], r2: float, d: float) -> int
```

Python example

Validate reachability of Joint-Pose path for 6-axis robot

```
start_joint = [0.0, 0.0, math.pi/2, 0.0, math.pi/2, 0.0] end_pose = [0.5, 0.2, 0.8, 0.0, math.pi/2, 0.0]
result = validatePath(1, start_joint, 0.01, end_pose, 0.01, 0.05)
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.RobotAlgorithm.validatePath","params":[0, [0.0,0.0,1.57,0.0,1.57,0.0], 0.01, [0.52,0.0,1.57,0.0,1.57,0.0], 0.01, 0.005],"id":1}
```

JSON-RPC response example `{"id":1,"jsonrpc":"2.0","result":0}`

8.6.7 RobotConfig ()

[RobotConfig](#).

Collaboration diagram for RobotConfig ():



Functions

- `std::string arcs::common_interface::RobotConfig::getName ()`
Get the robot's name.
- `int arcs::common_interface::RobotConfig::getDof ()`
Get the robot's degrees of freedom (from the hardware abstraction layer).
- `double arcs::common_interface::RobotConfig::getCycletime ()`
Get the robot's servo control cycle time (from the hardware abstraction layer).
- `int arcs::common_interface::RobotConfig::setSlowDownFraction (int level, double fraction)`
Set the speed reduction ratio for the preset slow-down mode.
- `double arcs::common_interface::RobotConfig::getSlowDownFraction (int level)`
Get the speed reduction ratio for the preset slow-down mode.
- `double arcs::common_interface::RobotConfig::getDefaultToolAcc ()`
Get the default tool acceleration, in m/s^2 .
- `double arcs::common_interface::RobotConfig::getDefaultToolSpeed ()`
Get the default tool speed, in m/s .
- `double arcs::common_interface::RobotConfig::getDefaultJointAcc ()`
Get the default joint acceleration, in rad/s^2 .
- `double arcs::common_interface::RobotConfig::getDefaultJointSpeed ()`
Get the default joint speed, in rad/s .
- `std::string arcs::common_interface::RobotConfig::getRobotType ()`

- Get the robot type code.
- `std::string arcs::common_interface::RobotConfig::getRobotSubType ()`
Get the robot sub-type code.
- `std::string arcs::common_interface::RobotConfig::getControlBoxType ()`
Get the control box type code.
- `int arcs::common_interface::RobotConfig::setMountingPose (const std::vector< double > &pose)`
Set the mounting pose (robot base coordinate system relative to world coordinate system) world->base
- `std::vector< double > arcs::common_interface::RobotConfig::getMountingPose ()`
Get the mounting pose (robot base coordinate system relative to world coordinate system)
- `int arcs::common_interface::RobotConfig::attachRobotBaseTo (const std::string &frame)`
Attach the robot base to a coordinate frame.
- `int arcs::common_interface::RobotConfig::setWorkObjectData (const WObjectData &wobj)`
Set work object data.
- `int arcs::common_interface::RobotConfig::setCollisionLevel (int level)`
Set the collision sensitivity level.
- `int arcs::common_interface::RobotConfig::getCollisionLevel ()`
Get the collision sensitivity level.
- `int arcs::common_interface::RobotConfig::setCollisionStopType (int type)`
Set the collision stop type.
- `int arcs::common_interface::RobotConfig::getCollisionStopType ()`
Get the collision stop type.
- `int arcs::common_interface::RobotConfig::setHomePosition (const std::vector< double > &positions)`
Set the robot's home position.
- `std::vector< double > arcs::common_interface::RobotConfig::getHomePosition ()`
Get the robot's home position.
- `int arcs::common_interface::RobotConfig::setFreedriveDamp (const std::vector< double > &damp)`
Set freedrive damping.
- `std::vector< double > arcs::common_interface::RobotConfig::getFreedriveDamp ()`
Get freedrive damping.
- `int arcs::common_interface::RobotConfig::setHandguidDamp (const std::vector< double > &damp)`
Set handguiding damping.
- `std::vector< double > arcs::common_interface::RobotConfig::getHandguidDamp ()`
Get handguiding damping.
- `std::unordered_map< std::string, std::vector< double > > arcs::common_interface::RobotConfig::getKinematicsParameters (bool real)`
Get the robot's DH parameters: alpha, a, d, theta, beta.
- `std::unordered_map< std::string, std::vector< double > > arcs::common_interface::RobotConfig::getKinematicsCompensationValues (double ref_temperature)`
Get DH parameter compensation values (alpha, a, d, theta, beta) at the specified temperature.
- `int arcs::common_interface::RobotConfig::setKinematicsCompensate (const std::unordered_map< std::string, std::vector< double > > ¶m)`
Set standard DH compensation to the robot.
- `int arcs::common_interface::RobotConfig::setPersistentParameters (const std::string ¶m)`
Set parameters to be saved to the interface board base.
- `int arcs::common_interface::RobotConfig::setHardwareCustomParameters (const std::string ¶m)`
Set custom parameters for the hardware abstraction layer.
- `std::string arcs::common_interface::RobotConfig::getHardwareCustomParameters (const std::string ¶m)`
Get custom parameters from the hardware abstraction layer.

- `int arcs::common_interface::RobotConfig::setRobotZero ()`
Set the robot joint zero position.
- `std::vector< std::string > arcs::common_interface::RobotConfig::getTcpForceSensorNames ()`
Get the names of available TCP force sensors.
- `int arcs::common_interface::RobotConfig::selectTcpForceSensor (const std::string &name)`
Set the TCP force sensor.
- `int arcs::common_interface::RobotConfig::setTcpForceSensorPose (const std::vector< double > &sensor_pose)`
Set the sensor mounting pose.
- `std::vector< double > arcs::common_interface::RobotConfig::getTcpForceSensorPose ()`
Get the sensor mounting pose.
- `bool arcs::common_interface::RobotConfig::hasTcpForceSensor ()`
Whether a TCP force sensor is installed.
- `int arcs::common_interface::RobotConfig::setTcpForceOffset (const std::vector< double > &force_offset)`
Set the TCP force/torque offset.
- `std::vector< double > arcs::common_interface::RobotConfig::getTcpForceOffset ()`
Get the TCP force/torque offset.
- `std::vector< std::string > arcs::common_interface::RobotConfig::getBaseForceSensorNames ()`
Get the names of available base force sensors.
- `int arcs::common_interface::RobotConfig::selectBaseForceSensor (const std::string &name)`
Set the base force sensor.
- `bool arcs::common_interface::RobotConfig::hasBaseForceSensor ()`
Whether a base force sensor is installed.
- `int arcs::common_interface::RobotConfig::setBaseForceOffset (const std::vector< double > &force_offset)`
Set the base force/torque offset.
- `std::vector< double > arcs::common_interface::RobotConfig::getBaseForceOffset ()`
Get the base force/torque offset.
- `uint32_t arcs::common_interface::RobotConfig::getSafetyParametersChecksum ()`
Get the safety parameter checksum CRC32.
- `int arcs::common_interface::RobotConfig::confirmSafetyParameters (const RobotSafetyParameterRange ¶meters)`
Initiate a request to confirm safety configuration parameters: Write safety configuration parameters to the safety interface board flash or file.
- `uint32_t arcs::common_interface::RobotConfig::calcSafetyParametersChecksum (const RobotSafetyParameterRange ¶meters)`
Calculate the CRC32 checksum of safety parameters.
- `std::vector< double > arcs::common_interface::RobotConfig::getJointMaxPositions ()`
Get the joint maximum positions (physical limits).
- `std::vector< double > arcs::common_interface::RobotConfig::getJointMinPositions ()`
Get the joint minimum positions (physical limits).
- `std::vector< double > arcs::common_interface::RobotConfig::getJointMaxSpeeds ()`
Get the joint maximum speeds (physical limits).
- `std::vector< double > arcs::common_interface::RobotConfig::getJointMaxAccelerations ()`
Get the joint maximum accelerations (physical limits).
- `std::vector< double > arcs::common_interface::RobotConfig::getTcpMaxSpeeds ()`
Get the TCP maximum speed (physical limits).
- `std::vector< double > arcs::common_interface::RobotConfig::getTcpMaxAccelerations ()`
Get the TCP maximum acceleration (physical limits).
- `int arcs::common_interface::RobotConfig::setGravity (const std::vector< double > &gravity)`

- Set the robot installation attitude (gravity vector).
- `std::vector< double > arcs::common_interface::RobotConfig::getGravity ()`
Get the robot's installation attitude (gravity vector).
- `int arcs::common_interface::RobotConfig::setTcpOffset (const std::vector< double > &offset)`
Set the current TCP offset.
- `std::vector< double > arcs::common_interface::RobotConfig::getTcpOffset ()`
Get the current TCP offset.
- `int arcs::common_interface::RobotConfig::setToolInertial (double m, const std::vector< double > &com, const std::vector< double > &inertial)`
Set tool mass, center of mass, and inertia.
- `int arcs::common_interface::RobotConfig::setPayload (double m, const std::vector< double > &cog, const std::vector< double > &aom, const std::vector< double > &inertial)`
Set the payload.
- `Payload arcs::common_interface::RobotConfig::getPayload ()`
Get the payload.
- `bool arcs::common_interface::RobotConfig::toolSpaceInRange (const std::vector< double > &pose)`
Whether the end-effector pose is within the safety range.
- `int arcs::common_interface::RobotConfig::firmwareUpdate (const std::string &fw)`
Initiate a firmware upgrade request.
- `std::tuple< std::string, double > arcs::common_interface::RobotConfig::getFirmwareUpdateProcess ()`
Get the current firmware upgrade process.
- `std::vector< double > arcs::common_interface::RobotConfig::getLimitJointMaxPositions ()`
Get the joint maximum positions (currently used limit values).
- `std::vector< double > arcs::common_interface::RobotConfig::getLimitJointMinPositions ()`
Get the joint minimum positions (currently used limit values).
- `std::vector< double > arcs::common_interface::RobotConfig::getLimitJointMaxSpeeds ()`
Get the joint maximum speeds (currently used limit values).
- `std::vector< double > arcs::common_interface::RobotConfig::getLimitJointMaxAccelerations ()`
Get the joint maximum accelerations (currently used limit values).
- `double arcs::common_interface::RobotConfig::getLimitTcpMaxSpeed ()`
Get the TCP maximum speed (currently used limit value).
- `SafeguardStopType arcs::common_interface::RobotConfig::getSafeguardStopType ()`
Get the current safeguard stop type.
- `int arcs::common_interface::RobotConfig::getSafeguardStopSource ()`
Get the complete safeguard stop source as a bitmask.
- `int arcs::common_interface::RobotConfig::getRobotEmergencyStopSource ()`
Get the complete robot emergency stop source as a bitmask.
- `std::string arcs::common_interface::RobotConfig::getSelectedTcpForceSensorName ()`
Get the name of the selected TCP force sensor.
- `int arcs::common_interface::RobotConfig::attachWeldingGun (double m, const std::vector< double > &cog, const std::vector< double > &inertial)`
- `int arcs::common_interface::RobotConfig::setCollisionThreshold (const std::vector< double > &threshold)`
Enable optimized collision force.
- `std::vector< double > arcs::common_interface::RobotConfig::getCollisionThreshold ()`
Get collision threshold.
- `int arcs::common_interface::RobotConfig::enableEndCollisionCheck (bool enable)`
Set end collision threshold.
- `bool arcs::common_interface::RobotConfig::isEndCollisionCheckEnabled ()`
Get collision check is enabled.

8.6.7.1 Detailed Description

[RobotConfig](#).

8.6.7.2 Function Documentation

8.6.7.2.1 attachRobotBaseTo()

```
int arcs::common_interface::RobotConfig::attachRobotBaseTo (
    const std::string & frame)
```

Attach the robot base to a coordinate frame.

If this frame moves, the robot will move accordingly. Used for applications like ground rails or gantries. When this function is called, the relationship between the frame and ROBOTBASE is fixed.

8.6.7.2.2 attachWeldingGun()

```
int arcs::common_interface::RobotConfig::attachWeldingGun (
    double m,
    const std::vector< double > & cog,
    const std::vector< double > & inertia)
```

8.6.7.2.3 calcSafetyParametersChecksum()

```
uint32_t arcs::common_interface::RobotConfig::calcSafetyParametersChecksum (
    const RobotSafetyParameterRange & parameters)
```

Calculate the CRC32 checksum of safety parameters.

Returns

`crc32`

Exceptions

<code>arcs::common_interface::AuboException</code>	
--	--

8.6.7.2.4 confirmSafetyParameters()

```
int arcs::common_interface::RobotConfig::confirmSafetyParameters (
    const RobotSafetyParameterRange & parameters)
```

Initiate a request to confirm safety configuration parameters: Write safety configuration parameters to the safety interface board flash or file.

Parameters

parameters	Safety configuration parameters.
------------	----------------------------------

Returns

Returns 0 on success; error code on failure. AUBO_BUSY AUBO_BAD_STATE -AUBO_INVL↔
_ARGUMENT -AUBO_BAD_STATE

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
confirmSafetyParameters(self: pyaubo_sdk.RobotConfig, arg0: arcs::common\_interface::RobotSafetyParameterRan
-> int
```

Lua function prototype

8.6.7.2.5 enableEndCollisionCheck()

```
int arcs::common_interface::RobotConfig::enableEndCollisionCheck (
    bool enable)
```

Set end collision threshold.

Parameters

threshold	
-----------	--

Returns

Returns 0 on success; error code on failure.

JSON-RPC request example

```
{ "jsonrpc": "2.0", "method": "rob1.RobotConfig.enableEndCollisionCheck", "params": [true], "id": 1 }
```

Generated by Doxygen

JSON-RPC response example

```
{ "id": 1, "jsonrpc": "2.0", "result": "None" }
```

Parameters

fw	<p>Firmware upgrade path. The format is: firmware package path#upgrade node list. The firmware package path and node list are separated by a hash (#), and nodes are separated by commas (.). If a node name ends with !, it means to force upgrade (without version check) that node; otherwise, the node will be upgraded only if the current version differs from the target version. You can flexibly specify which nodes to upgrade as needed. For example,</p> <p>/tmp/firmware_update-1.0.42-rc.5+2347b0d.firm#master_mcu!,slace_mcu!,base!,tool!,joint1!,joint2!,joint3!,joint4!,joint5!,joint6! means to force upgrade the interface board main, interface board slave, base, tool, and joints 1-6. "all" means all nodes, e.g.</p> <p>/tmp/firm_XXX.firm#all means upgrade all nodes with version check,</p> <p>/tmp/firm_XXX.firm#all! means force upgrade all nodes.</p>
----	--

Returns

Returns 0 if the command is sent successfully; otherwise, returns an error code. -AUBO_BAD_STATE: The current state of the [RuntimeMachine](#) is not Stopped, so the firmware upgrade request is rejected. AUBO_BAD_STATE = 1. -AUBO_TIMEOUT: Timeout. AUBO_TIMEOUT = 4.

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
firmwareUpdate(self: pyaubo_sdk.RobotConfig, arg0: str) -> int
```

Lua function prototype

```
firmwareUpdate(fw: string) -> nil
```

Lua example

```
firmwareUpdate("/tmp/firmware_update-1.0.42-rc.12+3e33eac.firm#master_mcu")
```

JSON-RPC request example

```
{ "jsonrpc": "2.0", "method": "rob1.RobotConfig.firmwareUpdate", "params": ["/tmp/firmware_update-1.0.42-rc.12+3e33eac.firm#master_mcu"], "id": 1 }
```

JSON-RPC response example

```
{ "id": 1, "jsonrpc": "2.0", "result": 0 }
```


Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
getBaseForceOffset(self: pyaubo_sdk.RobotConfig) -> List[float]
```

Lua function prototype

```
getBaseForceOffset() -> table
```

Lua example

```
BaseForceOffset = getBaseForceOffset()
```

JSON-RPC request example

```
{"jsonrpc": "2.0", "method": "rob1.RobotConfig.getBaseForceOffset", "params": [], "id": 1}
```

JSON-RPC response example

```
{"id": 1, "jsonrpc": "2.0", "result": []}
```

8.6.7.2.8 getBaseForceSensorNames()

```
std::vector< std::string > arcs::common_interface::RobotConfig::getBaseForceSensorNames ()
```

Get the names of available base force sensors.

Returns

Returns the names of available base force sensors.

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
getBaseForceSensorNames(self: pyaubo_sdk.RobotConfig) -> List[str]
```

Lua function prototype

```
getBaseForceSensorNames() -> table
```

Lua example

```
BaseForceSensorNames = getBaseForceSensorNames()
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.RobotConfig.getBaseForceSensorNames","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":[]}
```

8.6.7.2.9 getCollisionLevel()

```
int arcs::common_interface::RobotConfig::getCollisionLevel ()
```

Get the collision sensitivity level.

Returns

Collision sensitivity level.

Exceptions

arcs::common_interface::AuboException	
---------------------------------------	--

Python function prototype

```
getCollisionLevel(self: pyaubo_sdk.RobotConfig) -> int
```

Lua function prototype

```
getCollisionLevel() -> number
```

Lua example

```
CollisionLevel = getCollisionLevel()
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.RobotConfig.getCollisionLevel","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":6}
```

8.6.7.2.10 getCollisionStopType()

```
int arcs::common_interface::RobotConfig::getCollisionStopType ()
```

Get the collision stop type.

Returns

The collision stop type.

0: Floating after collision, i.e., enters freedrive mode after collision

1: Stop after collision

2: Brake after collision

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
getCollisionStopType(self: pyaubo_sdk.RobotConfig) -> int
```

Lua function prototype

```
getCollisionStopType\(\) -> number
```

Lua example

```
CollisionStopType = getCollisionStopType\(\)
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.RobotConfig.getCollisionStopType","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":1}
```

8.6.7.2.11 getCollisionThreshold()

```
std::vector< double > arcs::common_interface::RobotConfig::getCollisionThreshold ()
```

Get collision threshold.

Returns

General collision threshold.

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
getCollisionThreshold(self: pyaubo_sdk.RobotConfig) -> List[float]
```

Lua function prototype

```
getCollisionThreshold() -> table
```

Lua example

```
CollisionThreshold = getCollisionThreshold()
```

JSON-RPC request example

```
{ "jsonrpc": "2.0", "method": "rob1.RobotConfig.getCollisionThreshold", "params": [[99.0, 114.0, 103.0, 56.0, 51.0, 60.0, 54.6111111, 66.22222222, 59.66666667, 28.94444444, 27.05555556, 30.11111111, 19.1, 28.0, 25.0, 7.3, 7.9, 6.2]], "id": 1 }
```

JSON-RPC response example

```
{ "id": 1, "jsonrpc": "2.0", "result": "None" }
```

8.6.7.2.12 getControlBoxType()

```
std::string arcs::common_interface::RobotConfig::getControlBoxType ()
```

Get the control box type code.

Returns

The control box type code.

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
getControlBoxType(self: pyaubo_sdk.RobotConfig) -> str
```

Lua function prototype

```
getControlBoxType() -> string
```

Lua example

```
ControlBoxType = getControlBoxType()
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.RobotConfig.getControlBoxType","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":"cb_ISSStation"}
```

8.6.7.2.13 getCycletime()

double arcs::common_interface::RobotConfig::getCycletime ()

Get the robot's servo control cycle time (from the hardware abstraction layer).

Returns

The robot's servo control cycle time.

Exceptions

arcs::common_interface::AuboException	
---------------------------------------	--

Python function prototype

```
getCycletime(self: pyaubo_sdk.RobotConfig) -> float
```

Lua function prototype

```
getCycletime() -> number
```

Lua example

```
robot_Cycletime = getCycletime()
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.RobotConfig.getCycletime","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":0.005}
```

8.6.7.2.14 getDefaultJointAcc()

double arcs::common_interface::RobotConfig::getDefaultJointAcc ()

Get the default joint acceleration, in rad/s².

Returns

The default joint acceleration.

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

getDefaultJointAcc(self: pyaubo_sdk.RobotConfig) -> float

Lua function prototype

[getDefaultJointAcc\(\)](#) -> number

Lua example

DefaultJointAcc = [getDefaultJointAcc\(\)](#)

JSON-RPC request example

```
{"jsonrpc": "2.0", "method": "rob1.RobotConfig.getDefaultJointAcc", "params": [], "id": 1}
```

JSON-RPC response example

```
{"id": 1, "jsonrpc": "2.0", "result": 0.0}
```

8.6.7.2.15 getDefaultJointSpeed()

double arcs::common_interface::RobotConfig::getDefaultJointSpeed ()

Get the default joint speed, in rad/s.

Returns

The default joint speed.

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
getDefaultJointSpeed(self: pyaubo_sdk.RobotConfig) -> float
```

Lua function prototype

```
getDefaultJointSpeed() -> number
```

Lua example

```
DefaultJointSpeed = getDefaultJointSpeed()
```

JSON-RPC request example

```
{"jsonrpc": "2.0", "method": "rob1.RobotConfig.getDefaultJointSpeed", "params": [], "id": 1}
```

JSON-RPC response example

```
{"id": 1, "jsonrpc": "2.0", "result": 0.0}
```

8.6.7.2.16 getDefaultToolAcc()

```
double arcs::common_interface::RobotConfig::getDefaultToolAcc ()
```

Get the default tool acceleration, in m/s².

Returns

The default tool acceleration.

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
getDefaultToolAcc(self: pyaubo_sdk.RobotConfig) -> float
```

Lua function prototype

```
getDefaultToolAcc() -> number
```

Lua example

```
DefaultToolAcc = getDefaultToolAcc()
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.RobotConfig.getDefaultToolAcc","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":0.0}
```

8.6.7.2.17 getDefaultToolSpeed()

```
double arcs::common_interface::RobotConfig::getDefaultToolSpeed ()
```

Get the default tool speed, in m/s.

Returns

The default tool speed.

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
getDefaultToolSpeed(self: pyaubo_sdk.RobotConfig) -> float
```

Lua function prototype

```
getDefaultToolSpeed() -> number
```

Lua example

```
DefaultToolSpeed = getDefaultToolSpeed()
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.RobotConfig.getDefaultToolSpeed","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":0.0}
```


8.6.7.2.18 getDof()

```
int arcs::common_interface::RobotConfig::getDof ()
```

Get the robot's degrees of freedom (from the hardware abstraction layer).

Returns

The robot's degrees of freedom.

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
getDof(self: pyaubo_sdk.RobotConfig) -> int
```

Lua function prototype

```
getDof\(\) -> number
```

Lua example

```
robot_dof = getDof\(\)
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.RobotConfig.getDof","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":6}
```

8.6.7.2.19 getFirmwareUpdateProcess()

```
std::tuple< std::string, double > arcs::common_interface::RobotConfig::getFirmwareUpdateProcess ()
```

Get the current firmware upgrade process.

Returns

The current firmware upgrade process.

The first element is the step name. If it is "failed", the firmware upgrade failed.

The second element is the upgrade progress (0~1). When finished, returns ("", 1)

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
getFirmwareUpdateProcess(self: pyaubo_sdk.RobotConfig) -> Tuple[str, float]
```

Lua function prototype

```
getFirmwareUpdateProcess() -> table
```

Lua example

```
step,progress = getFirmwareUpdateProcess()
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.RobotConfig.getFirmwareUpdateProcess","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":[" ",0.0]}
```

8.6.7.2.20 getFreedriveDamp()

```
std::vector< double > arcs::common_interface::RobotConfig::getFreedriveDamp ()
```

Get freedrive damping.

Returns

Freedrive damping values.

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
getFreedriveDamp(self: pyaubo_sdk.RobotConfig) -> List[float]
```

Lua function prototype

```
getFreedriveDamp() -> table
```

Lua example

```
FreedriveDamp = getFreedriveDamp()
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.RobotConfig.getFreedriveDamp","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":[0.5,0.5,0.5,0.5,0.5,0.5]}
```

8.6.7.2.21 getGravity()

```
std::vector< double > arcs::common_interface::RobotConfig::getGravity ()
```

Get the robot's installation attitude (gravity vector).

If the robot base is equipped with an attitude sensor, data is read from the sensor; otherwise, the user setting is used.

Returns

Returns the installation attitude.

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
getGravity(self: pyaubo_sdk.RobotConfig) -> List[float]
```

Lua function prototype

```
getGravity() -> table
```

Lua example

```
Gravity = getGravity()
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.RobotConfig.getGravity","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":[0.0,0.0,-9.87654321]}
```

8.6.7.2.22 getHandguidDamp()

```
std::vector< double > arcs::common_interface::RobotConfig::getHandguidDamp ()
```

Get handguiding damping.

Returns

handguiding damping

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
getHandguidDamp(self: pyaubo_sdk.RobotConfig) -> List[float]
```

Lua fuunction prototype

```
getHandguidDamp\(\) -> table
```

Lua example

```
HandguidDamp = getHandguidDamp\(\)
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.RobotConfig.getHandguidDamp","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":[0.5,0.5,0.5,0.5,0.5,0.5]}
```

8.6.7.2.23 getHardwareCustomParameters()

```
std::string arcs::common_interface::RobotConfig::getHardwareCustomParameters (
    const std::string & param)
```

Get custom parameters from the hardware abstraction layer.

Parameters

--	--

return Returns custom parameters from the hardware abstraction layer.

Exceptions

arcs::common_interface::AuboException	
---	--

8.6.7.2.24 getHomePosition()

std::vector< double > arcs::common_interface::RobotConfig::getHomePosition ()

Get the robot's home position.

Returns

Exceptions

arcs::common_interface::AuboException	
---	--

JSON-RPC request example

```
{"jsonrpc": "2.0", "method": "rob1.RobotConfig.getHomePosition", "params": [], "id": 1}
```

JSON-RPC response example

```
{"id": 1, "jsonrpc": "2.0", "result": [0.0, -0.2617993877991494, 1.74532925199433, 0.4363323129985824, 1.570796326794897, 0.0]}
```

8.6.7.2.25 getJointMaxAccelerations()

std::vector< double > arcs::common_interface::RobotConfig::getJointMaxAccelerations ()

Get the joint maximum accelerations (physical limits).

Generated by Doxygen

Returns the joint maximum accelerations.

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
getJointMaxAccelerations(self: pyaubo_sdk.RobotConfig) -> List[float]
```

Lua function prototype

```
getJointMaxAccelerations() -> table
```

Lua example

```
JointMaxAccelerations = getJointMaxAccelerations()
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.RobotConfig.getJointMaxAccelerations","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":[31.104877758314785,31.104877758314785,31.104877758314785,20.73625684294463,20.73625684294463,20.73625684294463]}
```

8.6.7.2.26 getJointMaxPositions()

```
std::vector< double > arcs::common_interface::RobotConfig::getJointMaxPositions ()
```

Get the joint maximum positions (physical limits).

Returns

Returns the joint maximum positions.

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
getJointMaxPositions(self: pyaubo_sdk.RobotConfig) -> List[float]
```

Lua function prototype

```
getJointMaxPositions() -> table
```

Lua example

```
JointMaxPositions = getJointMaxPositions()
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.RobotConfig.getJointMaxPositions","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":[6.283185307179586,6.283185307179586,6.283185307179586,6.283185307179586,6.283185307179586,6.283185307179586]}
```

8.6.7.2.27 getJointMaxSpeeds()

```
std::vector< double > arcs::common_interface::RobotConfig::getJointMaxSpeeds ()
```

Get the joint maximum speeds (physical limits).

Returns

Returns the joint maximum speeds.

Exceptions

arcs::common_interface::AuboException

Python function prototype

```
getJointMaxSpeeds(self: pyaubo_sdk.RobotConfig) -> List[float]
```

Lua function prototype

```
getJointMaxSpeeds() -> table
```

Lua example

```
JointMaxSpeeds = getJointMaxSpeeds()
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.RobotConfig.getJointMaxSpeeds","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":[3.892084231947355,3.892084231947355,3.892084231947355,3.892084231947355,3.892084231947355,3.892084231947355]}
```

8.6.7.2.28 `getJointMinPositions()`

```
std::vector< double > arcs::common_interface::RobotConfig::getJointMinPositions ()
```

Get the joint minimum positions (physical limits).

Returns

Returns the joint minimum positions.

Exceptions

<code>arcs::common_interface::AuboException</code>	
--	--

Python function prototype

```
getJointMinPositions(self: pyaubo_sdk.RobotConfig) -> List[float]
```

Lua function prototype

```
getJointMinPositions() -> table
```

Lua example

```
JointMinPositions = getJointMinPositions()
```

JSON-RPC request example

```
{ "jsonrpc": "2.0", "method": "rob1.RobotConfig.getJointMinPositions", "params": [], "id": 1 }
```

JSON-RPC response example

```
{ "id": 1, "jsonrpc": "2.0", "result": [-6.283185307179586, -6.283185307179586, -6.283185307179586, -6.283185307179586, -6.283185307179586, -6.283185307179586] }
```

8.6.7.2.29 `getKinematicsCompensate()`

```
std::unordered_map< std::string, std::vector< double > > arcs::common_interface::RobotConfig::getKinematicsCompensate (
    double ref_temperature)
```

Get DH parameter compensation values (alpha, a, d, theta, beta) at the specified temperature.

Parameters

ref_temperature	Reference temperature in °C, default is 20°C
-----------------	--

Returns

Returns DH parameter compensation values.

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
getKinematicsCompensate(self: pyaubo_sdk.RobotConfig, arg0: float) -> Dict[str, List[float]]
```

Lua function prototype

```
getKinematicsCompensate(ref_temperature: number) -> table
```

Lua example

```
KinematicsCompensate = getKinematicsCompensate(20)
```

JSON-RPC request example

```
{"jsonrpc": "2.0", "method": "rob1.RobotConfig.getKinematicsCompensate", "params": [20], "id": 1}
```

JSON-RPC response example

```
{"id": 1, "jsonrpc": "2.0", "result": {"a": [0.0, -0.0001255999959539622, 0.0006348000024445355, -0.0002398000069661066, -0.00018950000230688602, 3.7000001611886546e-05], "alpha": [0.0, 0.000678930024150759, 0.002438219962641597, 0.0049148499965667725, 0.00048695001169107854, 0.004321440123021603], "beta": [0.0, 0.0, 0.0, 0.0, 0.0, 0.0], "d": [0.0, 4.769999941345304e-05, -4.769999941345304e-05, 4.769999941345304e-05, 0.0003789000038523227, 0.0], "theta": [0.0, 0.0, 0.0, 0.0, 0.0, 0.0]}}
```

8.6.7.2.30 getKinematicsParam()

```
std::unordered_map< std::string, std::vector< double > > arcs::common_interface::RobotConfig::getKinematicsParam (
    bool real)
```

Get the robot's DH parameters: alpha, a, d, theta, beta.

Parameters

real	Read real parameters (theoretical + compensation) or theoretical parameters.
------	--

Returns

Returns the robot's DH parameters.

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
getKinematicsParam(self: pyaubo_sdk.RobotConfig, arg0: bool) -> Dict[str, List[float]]
```

Lua function prototype

```
getKinematicsParam(real: boolean) -> table
```

Lua example

```
KinematicsParam = getKinematicsParam(true)
```

JSON-RPC request example

```
{"jsonrpc": "2.0", "method": "rob1.RobotConfig.getKinematicsParam", "params": [true], "id": 1}
```

JSON-RPC response example

```
{"id": 1, "jsonrpc": "2.0", "result": {"a": [0.0, -0.0001255999959539622, 0.4086348000024445, 0.3757601999930339, -0.00018950000230688602, 3.7000001611886546e-05], "alpha": [0.0, -1.5701173967707458, 3.1440308735524347, 3.141592653589793, -1.5703093767832055, 1.5751177669179182], "beta": [0.0, 0.0, 0.0, 0.0, 0.0, 0.0], "d": [0.122, 0.12154769999941345, -4.769999941345304e-05, 4.769999941345304e-05, 0.10287890000385232, 0.116], "theta": [3.141592653589793, -1.5707963267948966, 0.0, -1.5707963267948966, 0.0, 0.0]}}
```

8.6.7.2.31 getLimitJointMaxAccelerations()

```
std::vector< double > arcs::common_interface::RobotConfig::getLimitJointMaxAccelerations ()
```

Get the joint maximum accelerations (currently used limit values).

Returns

Returns the joint maximum accelerations (currently used limit values).

Exceptions

<code>arcs::common_interface::AuboException</code>	
--	--

Python function prototype

```
getLimitJointMaxAccelerations(self: pyaubo_sdk.RobotConfig) -> List[float]
```

Lua function prototype

```
getLimitJointMaxAccelerations() -> table
```

Lua example

```
LimitJointMaxAccelerations = getLimitJointMaxAccelerations()
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.RobotConfig.getLimitJointMaxAccelerations","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":[31.104877758314785,31.104877758314785,31.104877758314785,20.73625684294463,20.73625684294463,20.73625684294463]}
```

8.6.7.2.32 getLimitJointMaxPositions()

```
std::vector< double > arcs::common_interface::RobotConfig::getLimitJointMaxPositions ()
```

Get the joint maximum positions (currently used limit values).

Returns

Returns the joint maximum positions (currently used limit values).

Exceptions

<code>arcs::common_interface::AuboException</code>	
--	--

Python function prototype

```
getLimitJointMaxPositions(self: pyaubo_sdk.RobotConfig) -> List[float]
```

Lua function prototype

```
getLimitJointMaxPositions() -> table
```

Lua example

```
LimitJointMaxPositions = getLimitJointMaxPositions()
```

JSON-RPC request example

```
{"jsonrpc": "2.0", "method": "rob1.RobotConfig.getLimitJointMaxPositions", "params": [], "id": 1}
```

JSON-RPC response example

```
{"id": 1, "jsonrpc": "2.0", "result": [6.2831854820251465, 6.2831854820251465, 6.2831854820251465, 6.2831854820251465, 6.2831854820251465, 6.2831854820251465]}
```

8.6.7.2.33 getLimitJointMaxSpeeds()

```
std::vector< double > arcs::common_interface::RobotConfig::getLimitJointMaxSpeeds ()
```

Get the joint maximum speeds (currently used limit values).

Returns

Returns the joint maximum speeds (currently used limit values).

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
getLimitJointMaxSpeeds(self: pyaubo_sdk.RobotConfig) -> List[float]
```

Lua function prototype

```
getLimitJointMaxSpeeds() -> table
```

Lua example

```
LimitJointMaxSpeeds = getLimitJointMaxSpeeds()
```

JSON-RPC request example

```
{"jsonrpc": "2.0", "method": "rob1.RobotConfig.getLimitJointMaxSpeeds", "params": [], "id": 1}
```

JSON-RPC response example

```
{"id": 1, "jsonrpc": "2.0", "result": [3.8920841217041016, 3.8920841217041016, 3.8920841217041016, 3.8920841217041016, 3.8920841217041016, 3.8920841217041016]}
```

8.6.7.2.34 getLimitJointMinPositions()

```
std::vector< double > arcs::common_interface::RobotConfig::getLimitJointMinPositions ()
```

Get the joint minimum positions (currently used limit values).

Returns

Returns the joint minimum positions (currently used limit values).

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
getLimitJointMinPositions(self: pyaubo_sdk.RobotConfig) -> List[float]
```

Lua function prototype

```
getLimitJointMinPositions\(\) -> table
```

Lua example

```
LimitJointMinPositions = getLimitJointMinPositions\(\)
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.RobotConfig.getLimitJointMinPositions","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":[-6.2831854820251465,-6.2831854820251465,-6.2831854820251465,-6.2831854820251465,-6.2831854820251465,-6.2831854820251465]}
```

8.6.7.2.35 getLimitTcpMaxSpeed()

```
double arcs::common_interface::RobotConfig::getLimitTcpMaxSpeed ()
```

Get the TCP maximum speed (currently used limit value).

Returns

Returns the TCP maximum speed (currently used limit value).

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
getLimitTcpMaxSpeed(self: pyaubo_sdk.RobotConfig) -> List[float]
```

Lua function prototype

```
getLimitTcpMaxSpeed() -> number
```

Lua example

```
LimitTcpMaxSpeed = getLimitTcpMaxSpeed()
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.RobotConfig.getLimitTcpMaxSpeed","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":2.0}
```

8.6.7.2.36 getMountingPose()

```
std::vector< double > arcs::common_interface::RobotConfig::getMountingPose ()
```

Get the mounting pose (robot base coordinate system relative to world coordinate system)

Returns

Mounting pose

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
getMountingPose(self: pyaubo_sdk.RobotConfig) -> List[float]
```

Lua function prototype

```
getMountingPose() -> table
```

Lua example

```
MountingPose = getMountingPose()
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.RobotConfig.getMountingPose","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":[0.0,0.0,0.0,0.0,0.0,0.0]}
```

8.6.7.2.37 getName()

std::string arcs::common_interface::RobotConfig::getName ()

Get the robot's name.

Returns

The robot's name.

Exceptions

arcs::common_interface::AuboException	
---------------------------------------	--

Python function prototype

```
getName(self: pyaubo_sdk.RobotConfig) -> str
```

Lua function prototype

```
getName() -> string
```

Lua example

```
robot_name = getName()
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.RobotConfig.getName","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":"rob1"}
```

8.6.7.2.38 `getPayload()`

`Payload` `arcs::common_interface::RobotConfig::getPayload ()`

Get the payload.

Returns

The payload. The first element is the mass in kg; The second element is the center of gravity in meters, format (CoGx, CoGy, CoGz); The third element is the axis of moment in radians, format (rx, ry, rz); The fourth element is the inertia in kg*m², format (Ixx, Iyy, Izz, Ixy, Ixz, Iyz)

Exceptions

<code>arcs::common_interface::AuboException</code>	
--	--

Python function prototype

```
getPayload(self: pyaubo_sdk.RobotConfig) -> Tuple[float, List[float], List[float], List[float]]
```

Lua function prototype

```
getPayload() -> number, table, table, table
```

Lua example

```
m, cog, aom, inertia = getPayload()
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.RobotConfig.getPayload","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":[3.0,[0.0,0.0,0.0],[0.0,0.0,0.0],[0.0,0.0,0.0,0.0,0.0,0.0]]}
```

8.6.7.2.39 `getRobotEmergencyStopSource()`

`int` `arcs::common_interface::RobotConfig::getRobotEmergencyStopSource ()`

Get the complete robot emergency stop source as a bitmask.

Returns

Returns all robot emergency stop sources.

Reasons for emergency stop: Emergency stop triggered by control box button - 1<<0 Emergency stop triggered by teach pendant button - 1<<1 Emergency stop triggered by handle button - 1<<2 Emergency stop triggered by fixed IO - 1<<3

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.RobotConfig.getRobotEmergencyStopSource","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":0}
```


8.6.7.2.40 getRobotSubType()

```
std::string arcs::common_interface::RobotConfig::getRobotSubType ()
```

Get the robot sub-type code.

Returns

The robot sub-type code.

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
getRobotSubType(self: pyaubo_sdk.RobotConfig) -> str
```

Lua function prototype

```
getRobotSubType\(\) -> string
```

Lua example

```
RobotSubType = getRobotSubType\(\)
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.RobotConfig.getRobotSubType","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":"B0"}
```

8.6.7.2.41 getRobotType()

```
std::string arcs::common_interface::RobotConfig::getRobotType ()
```

Get the robot type code.

Returns

The robot type code.

Exceptions

<code>arcs::common_interface::AuboException</code>	
--	--

Python function prototype

```
getRobotType(self: pyaubo_sdk.RobotConfig) -> str
```

Lua function prototype

```
getRobotType() -> string
```

Lua example

```
RobotType = getRobotType()
```

JSON-RPC request example

```
{"jsonrpc": "2.0", "method": "rob1.RobotConfig.getRobotType", "params": [], "id": 1}
```

JSON-RPC response example

```
{"id": 1, "jsonrpc": "2.0", "result": "aubo_i5H"}
```

8.6.7.2.42 getSafeguardStopSource()

```
int arcs::common_interface::RobotConfig::getSafeguardStopSource ()
```

Get the complete safeguard stop source as a bitmask.

Returns

Returns all safeguard stop sources.

Reasons for safeguard stop: Safeguard stop triggered by configurable safety IO in manual mode - 1<<0
 Safeguard stop triggered by configurable safety IO in automatic mode - 1<<1 Safeguard stop triggered by control box SI input - 1<<2 Safeguard stop triggered by teach pendant three-position switch - 1<<3
 Safeguard stop triggered by switching from auto to manual - 1<<4

JSON-RPC request example

```
{"jsonrpc": "2.0", "method": "rob1.RobotConfig.getSafeguardStopSource", "params": [], "id": 1}
```

JSON-RPC response example

```
{"id": 1, "jsonrpc": "2.0", "result": 0}
```

8.6.7.2.43 getSafeguardStopType()

[SafeguedStopType](#) arcs::common_interface::RobotConfig::getSafeguardStopType ()

Get the current safeguard stop type.

Returns

Returns the current safeguard stop type.

JSON-RPC request example

```
{ "jsonrpc": "2.0", "method": "rob1.RobotConfig.getSafeguardStopType", "params": [], "id": 1 }
```

JSON-RPC response example

```
{ "id": 1, "jsonrpc": "2.0", "result": "None" }
```

8.6.7.2.44 getSafetyParametersChecksum()

uint32_t arcs::common_interface::RobotConfig::getSafetyParametersChecksum ()

Get the safety parameter checksum CRC32.

Returns

Returns the safety parameter checksum.

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
getSafetyParametersChecksum(self: pyaubo_sdk.RobotConfig) -> int
```

Lua function prototype

```
getSafetyParametersChecksum\(\) -> number
```

Lua

```
SafetyParametersChecksum = getSafetyParametersChecksum\(\)
```

JSON-RPC request example

```
{ "jsonrpc": "2.0", "method": "rob1.RobotConfig.getSafetyParametersChecksum", "params": [], "id": 1 }
```

JSON-RPC response example

```
{ "id": 1, "jsonrpc": "2.0", "result": 2033397241 }
```

8.6.7.2.45 `getSelectedTcpForceSensorName()`

```
std::string arcs::common_interface::RobotConfig::getSelectedTcpForceSensorName ()
```

Get the name of the selected TCP force sensor.

Returns

The name of the currently selected TCP force sensor.

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
getSelectedTcpForceSensorName(self: pyaubo_sdk.RobotConfig) -> str
```

Lua function prototype

```
getSelectedTcpForceSensorName\(\) -> str
```

Lua example

```
TcpForceSensorName = getSelectedTcpForceSensorName\(\)
```

JSON-RPC request example

```
{ "jsonrpc": "2.0", "method": "rob1.RobotConfig.getSelectedTcpForceSensorName", "params": [],  
  "id": 1 }
```

8.6.7.2.46 `getSlowDownFraction()`

```
double arcs::common_interface::RobotConfig::getSlowDownFraction (  
    int level)
```

Get the speed reduction ratio for the preset slow-down mode.

Parameters

level	Slow-down level 1, 2
-------	----------------------

Returns

The speed reduction ratio for the preset slow-down mode.

Exceptions

arcs::common_interface::AuboException	
---	--

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.RobotConfig.getSlowDownFraction","params":[1],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":0.5}
```

8.6.7.2.47 getTcpForceOffset()

```
std::vector< double > arcs::common_interface::RobotConfig::getTcpForceOffset ()
```

Get the TCP force/torque offset.

Returns

Returns the TCP force/torque offset.

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
getTcpForceOffset(self: pyaubo_sdk.RobotConfig) -> List[float]
```

Lua function prototype

```
getTcpForceOffset\(\) -> table
```

Lua example

```
TcpForceOffset = getTcpForceOffset\(\)
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.RobotConfig.getTcpForceOffset","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":[0.0,0.0,0.0,0.0,0.0,0.0]}
```

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
getTcpForceSensorNames(self: pyaubo_sdk.RobotConfig) -> List[str]
```

Lua function prototype

```
getTcpForceSensorNames() -> table
```

Lua example

```
TcpForceSensorNames = getTcpForceSensorNames()
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.RobotConfig.getTcpForceSensorNames","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":[]}
```

8.6.7.2.49 getTcpForceSensorPose()

```
std::vector< double > arcs::common_interface::RobotConfig::getTcpForceSensorPose ()
```

Get the sensor mounting pose.

Returns

Sensor mounting pose.

Exceptions

arcs::common_interface::AuboException	
---	--

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.RobotConfig.getTcpForceSensorPose","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":[0.0,0.0,0.0,0.0,0.0,0.0]}
```

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
getTcpMaxAccelerations(self: pyaubo_sdk.RobotConfig) -> List[float]
```

Lua function prototype

```
getTcpMaxAccelerations() -> table
```

Lua example

```
TcpMaxAccelerations = getTcpMaxAccelerations()
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.RobotConfig.getTcpMaxAccelerations","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":[10.0,10.0]}
```

8.6.7.2.51 getTcpMaxSpeeds()

```
std::vector< double > arcs::common_interface::RobotConfig::getTcpMaxSpeeds ()
```

Get the TCP maximum speed (physical limits).

Returns

Returns the TCP maximum speed.

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
getTcpMaxSpeeds(self: pyaubo_sdk.RobotConfig) -> List[float]
```

Lua function prototype

```
getTcpMaxSpeeds() -> table
```

Lua example

```
TcpMaxSpeeds = getTcpMaxSpeeds()
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.RobotConfig.getTcpMaxSpeeds","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":[2.0,5.0]}
```

8.6.7.2.52 getTcpOffset()

```
std::vector< double > arcs::common_interface::RobotConfig::getTcpOffset ()
```

Get the current TCP offset.

The TCP offset is represented as (x, y, z, rx, ry, rz). x, y, z are the position offsets of the Tool Center Point (TCP) relative to the flange center in the base coordinate system, in meters. rx, ry, rz are the orientation offsets of the TCP relative to the flange center in the base coordinate system, as ZYX Euler angles in radians.

Returns

The current TCP offset, in the form (x, y, z, rx, ry, rz)

Exceptions

<code>arcs::common_interface::AuboException</code>
--

Python function prototype

```
getTcpOffset(self: pyaubo_sdk.RobotConfig) -> List[float]
```

Lua function prototype

```
getTcpOffset() -> table
```

Lua example

```
TcpOffset = getTcpOffset()
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.RobotConfig.getTcpOffset","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":[0.0,0.0,0.0,0.0,0.0,0.0]}
```


8.6.7.2.53 hasBaseForceSensor()

```
bool arcs::common_interface::RobotConfig::hasBaseForceSensor ()
```

Whether a base force sensor is installed.

Returns

Returns true if installed; false otherwise.

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
hasBaseForceSensor(self: pyaubo_sdk.RobotConfig) -> bool
```

Lua function prototype

```
hasBaseForceSensor\(\) -> boolean
```

Lua example

```
hasBaseForceSensor = hasBaseForceSensor\(\)
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.RobotConfig.hasBaseForceSensor","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":false}
```

8.6.7.2.54 hasTcpForceSensor()

```
bool arcs::common_interface::RobotConfig::hasTcpForceSensor ()
```

Whether a TCP force sensor is installed.

Returns

Returns true if installed; false otherwise.

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
hasTcpForceSensor(self: pyaubo_sdk.RobotConfig) -> bool
```

Lua function prototype

```
hasTcpForceSensor() -> boolean
```

Lua example

```
hasTcpForceSensor = hasTcpForceSensor()
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.RobotConfig.hasTcpForceSensor","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":true}
```

8.6.7.2.55 isEndCollisionCheckEnabled()

```
bool arcs::common_interface::RobotConfig::isEndCollisionCheckEnabled ()
```

Get collision check is enabled.

Returns

end collision threshold.

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
isEndCollisionCheckEnabled(self: pyaubo_sdk.RobotConfig) -> bool
```

Lua function prototype

```
isEndCollisionCheckEnabled() -> bool
```

Lua example

```
IsEndCollisionCheckEnabled = isEndCollisionCheckEnabled()
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.RobotConfig.isEndCollisionCheckEnabled","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":true}
```

8.6.7.2.56 selectBaseForceSensor()

```
int arcs::common_interface::RobotConfig::selectBaseForceSensor (
    const std::string & name)
```

Set the base force sensor.

If there is a built-in base force sensor, the built-in sensor will be used by default.

Parameters

name	Name of the base force sensor.
------	--------------------------------

Returns

Returns 0 on success; error code on failure. AUBO_BUSY AUBO_BAD_STATE -AUBO_INVL↵
_ARGUMENT -AUBO_BAD_STATE

Exceptions

arcs::common_interface::AuboException	
---------------------------------------	--

Python function prototype

```
selectBaseForceSensor(self: pyaubo_sdk.RobotConfig, arg0: str) -> int
```

Parameters

name	Name of the TCP force sensor.
------	-------------------------------

Returns

Returns 0 on success; error code on failure. AUBO_BUSY AUBO_BAD_STATE -AUBO_INVL↔
_ARGUMENT -AUBO_BAD_STATE

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
selectTcpForceSensor(self: pyaubo_sdk.RobotConfig, arg0: str) -> int
```

Lua function prototype

```
selectTcpForceSensor(name: string) -> nil
```

8.6.7.2.58 setBaseForceOffset()

```
int arcs::common_interface::RobotConfig::setBaseForceOffset (
    const std::vector< double > & force_offset)
```

Set the base force/torque offset.

Parameters

force_offset	Base force/torque offset
--------------	--------------------------

Returns

Returns 0 on success; error code on failure AUBO_BUSY AUBO_BAD_STATE -AUBO_INVL↔
_ARGUMENT -AUBO_BAD_STATE

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
setBaseForceOffset(self: pyaubo_sdk.RobotConfig, arg0: List[float]) -> int
```

Lua function prototype

```
setBaseForceOffset(force_offset: table) -> nil
```

Lua

```
setBaseForceOffset({0.0,0.0,0.0,0.0,0.0,0.0,0.0})
```

8.6.7.2.59 setCollisionLevel()

```
int arcs::common_interface::RobotConfig::setCollisionLevel (
    int level)
```

Set the collision sensitivity level.

The higher the value, the more sensitive.

Parameters

level	Collision sensitivity level 0: Disable collision detection 1~9: Collision sensitivity levels
-------	--

Returns

Returns 0 on success; error code on failure AUBO_BUSY AUBO_BAD_STATE -AUBO_INVL↵
_ARGUMENT -AUBO_BAD_STATE

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
setCollisionLevel(self: pyaubo_sdk.RobotConfig, arg0: int) -> int
```

Lua function prototype

```
setCollisionLevel(level: number) -> nil
```

Lua example

```
setCollisionLevel(6)
```

JSON-RPC request example

```
{"jsonrpc": "2.0", "method": "rob1.RobotConfig.setCollisionLevel", "params": [6], "id": 1}
```

JSON-RPC response example

```
{"id": 1, "jsonrpc": "2.0", "result": 0}
```

8.6.7.2.60 setCollisionStopType()

```
int arcs::common_interface::RobotConfig::setCollisionStopType (
    int type)
```

Set the collision stop type.

Parameters

type	Type
	0: Floating after collision, i.e., enters freedrive mode after collision
	1: Stop after collision
	2: Brake after collision

Returns

Returns 0 on success; error code on failure AUBO_BUSY AUBO_BAD_STATE -AUBO_INVL←
_ARGUMENT -AUBO_BAD_STATE

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
setCollisionStopType(self: pyaubo_sdk.RobotConfig, arg0: int) -> int
```

Lua function prototype

```
setCollisionStopType(type: number) -> nil
```

Lua example

```
setCollisionStopType(1)
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.RobotConfig.setCollisionStopType","params":[1],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":0}
```

8.6.7.2.61 setCollisionThreshold()

```
int arcs::common_interface::RobotConfig::setCollisionThreshold (
    const std::vector< double > & threshold)
```

Enable optimized collision force.

Parameters

threshold	
-----------	--

Returns

Returns 0 on success; error code on failure.

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.RobotConfig.setCollisionThreshold","params":[[99.0, 114.0,
103.0, 56.0, 51.0, 60.0,54.61111111, 66.22222222, 59.66666667, 28.94444444, 27.05555556, 30.↵
11111111,19.1, 28.0, 25.0, 7.3, 7.9, 6.2]], "id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result": "None"}
```

8.6.7.2.62 setFreedriveDamp()

Parameters

damp	Damping values.
------	-----------------

Returns

Returns 0 on success; error code on failure AUBO__BUSY AUBO__BAD_STATE -AUBO__INVL↵
_ARGUMENT -AUBO__BAD_STATE

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
setFreedriveDamp(self: pyaubo_sdk.RobotConfig, arg0: List[float]) -> int
```

Lua function prototype

```
setFreedriveDamp(damp: table) -> nil
```

Lua example

```
setFreedriveDamp({0.5,0.5,0.5,0.5,0.5,0.5})
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.RobotConfig.setFreedriveDamp","params":[[0.5,0.5,0.5,0.5,0.5,0.5↵  
5,0.5]],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":0}
```

8.6.7.2.63 setGravity()

```
int arcs::common_interface::RobotConfig::setGravity (
    const std::vector< double > & gravity)
```

Set the robot installation attitude (gravity vector).

Parameters

gravity	Installation attitude (gravity vector)
---------	--

Returns

Returns 0 on success; error code on failure AUBO_BUSY AUBO_BAD_STATE -AUBO_INVL↔
_ARGUMENT -AUBO_BAD_STATE

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
setGravity(self: pyaubo_sdk.RobotConfig, arg0: List[float]) -> int
```

Lua function prototype

```
setGravity(gravity: table) -> nil
```

Lua example

```
setGravity({0.0,0.0,-9.87654321})
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.RobotConfig.setGravity","params":[[0.0,0.0,-9.87654321]],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":0}
```

8.6.7.2.64 setHandguidDamp()

```
int arcs::common_interface::RobotConfig::setHandguidDamp (
    const std::vector< double > & damp)
```

Set handguiding damping.

Parameters

damp	damping coef.
------	---------------

Returns

return 0 for success return rror code for failure AUBO_BUSY AUBO_BAD_STATE -AUBO_↵
INVL_ARGUMENT -AUBO_BAD_STATE

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
setHandguidDamp(self: pyaubo_sdk.RobotConfig, arg0: List[float]) -> int
```

Lua function prototype

```
setHandguidDamp(damp: table) -> number
```

Lua example

```
setHandguidDamp({0.5,0.5,0.5,0.5,0.5,0.5})
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.RobotConfig.setHandguidDamp","params":[[0.5,0.5,0.5,0.5,0.↵  
5,0.5]],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":0}
```

8.6.7.2.65 setHardwareCustomParameters()

```
int arcs::common_interface::RobotConfig::setHardwareCustomParameters (
    const std::string & param)
```

Set custom parameters for the hardware abstraction layer.

The purpose is to ensure compatibility between different hardware.

Parameters

param	Custom parameters.
-------	--------------------

Returns

Returns 0 on success; error code on failure. AUBO_BUSY AUBO_BAD_STATE -AUBO_INVL↵
_ARGUMENT -AUBO_BAD_STATE

Exceptions

arcs::common_interface::AuboException	
---	--

8.6.7.2.66 setHomePosition()

```
int arcs::common_interface::RobotConfig::setHomePosition (
    const std::vector< double > & positions)
```

Set the robot's home position.

Parameters

positions	Joint angles
-----------	--------------

Returns

Returns 0 on success; error code on failure AUBO_BUSY AUBO_BAD_STATE -AUBO_INVL↵
_ARGUMENT -AUBO_BAD_STATE

Exceptions

arcs::common_interface::AuboException	
---	--

JSON-RPC request example

```
{ "jsonrpc": "2.0", "method": "rob1.RobotConfig.setHomePosition", "params": [[0.0,-0.2617993877991494,1.↵  
74532925199433,0.4363323129985824,1.570796326794897,0.0]], "id": 1 }
```

JSON-RPC response example

```
{ "id": 1, "jsonrpc": "2.0", "result": 0 }
```

Parameters

param	
-------	--

Returns

Returns 0 on success; error code on failure AUBO_BUSY AUBO_BAD_STATE -AUBO_INVL↔
_ARGUMENT -AUBO_BAD_STATE

Exceptions

arcs::common_interface::AuboException	
---	--

8.6.7.2.68 setMountingPose()

```
int arcs::common_interface::RobotConfig::setMountingPose (
    const std::vector< double > & pose)
```

Set the mounting pose (robot base coordinate system relative to world coordinate system) world->base

Typically used in multi-robot systems, default is [0,0,0,0,0,0]

Parameters

pose	Mounting pose
------	---------------

Returns

Returns 0 on success; error code on failure AUBO_BUSY AUBO_BAD_STATE -AUBO_INVL↔
_ARGUMENT -AUBO_BAD_STATE

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
setMountingPose(self: pyaubo_sdk.RobotConfig, arg0: List[float]) -> int
```

Lua function prototype

```
setMountingPose(pose: table) -> nil
```

Lua example

```
setMountingPose({0.0,0.0,0.0,0.0,0.0,0.0,0.0})
```

JSON-RPC request example

```
{"jsonrpc": "2.0", "method": "rob1.RobotConfig.setMountingPose", "params": [[0.0,0.0,0.0,0.0,0.0,0.0,0.0], "id": 1}
```

JSON-RPC response example

```
{"id": 1, "jsonrpc": "2.0", "result": 0}
```

8.6.7.2.69 setPayload()

```
int arcs::common_interface::RobotConfig::setPayload (
    double m,
    const std::vector< double > & cog,
    const std::vector< double > & aom,
    const std::vector< double > & inertia)
```

Set the payload.

Parameters

m	Mass, unit: kg
cog	Center of gravity, unit: m, format (CoGx, CoGy, CoGz)
aom	Axis of moment, unit: rad, format (rx, ry, rz)
inertia	Inertia, unit: kg*m^2, format (Ixx, Iyy, Izz, Ixy, Ixz, Iyz) or (Ixx, Ixy, Ixz, Iyx, Iyy, Iyz, Izx, Izy, Izz)

Returns

Returns 0 on success; error code on failure AUBO_BUSY AUBO_BAD_STATE -AUBO_INVALID_ARGUMENT -AUBO_BAD_STATE

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
setPayload(self: pyaubo_sdk.RobotConfig, arg0: float, arg1: List[float], arg2: List[float], arg3: List[float]) -> int
```

Lua function prototype

```
setPayload(m: number, cog: table, aom: table, inertia: table) -> nil
```

Lua example

```
setPayload(3, {0,0,0}, {0,0,0}, {0,0,0,0,0,0,0,0,0})
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.RobotConfig.setPayload","params":[3,[0,0,0],[0,0,0],[0,0,0,0,0,0,0,0,0]],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":0}
```

8.6.7.2.70 setPersistentParameters()

```
int arcs::common_interface::RobotConfig::setPersistentParameters (
    const std::string & param)
```

Set parameters to be saved to the interface board base.

Parameters

param	Compensation data.
-------	--------------------

Returns

Returns 0 on success; error code on failure. AUBO_BUSY AUBO_BAD_STATE -AUBO_INVALID_ARGUMENT -AUBO_BAD_STATE

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
setPersistentParameters(self: pyaubo_sdk.RobotConfig, arg0: str) -> int
```

Lua function prototype

```
setPersistentParameters(param: string) -> nil
```

8.6.7.2.71 setRobotZero()

```
int arcs::common_interface::RobotConfig::setRobotZero ( )
```

Set the robot joint zero position.

Returns

Returns 0 on success; error code on failure AUBO_BUSY AUBO_BAD_STATE -AUBO_BAD↵
_STATE

Exceptions

<code>arcs::common_interface::AuboException</code>	
--	--

Python function prototype

```
setRobotZero(self: pyaubo_sdk.RobotConfig) -> int
```

Lua function prototype

```
setRobotZero() -> nil
```

Lua example

```
setRobotZero()
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.RobotConfig.setRobotZero","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":0}
```

8.6.7.2.72 setSlowDownFraction()

```
int arcs::common_interface::RobotConfig::setSlowDownFraction (
    int level,
    double fraction)
```

Set the speed reduction ratio for the preset slow-down mode.

Parameters

level	Slow-down level 1, 2
fraction	

Returns

Returns 0 on success; error code on failure AUBO_BUSY AUBO_BAD_STATE AUBO_INVL↵
_ARGUMENT -AUBO_BAD_STATE

Exceptions

arcs::common_interface::AuboException	
---	--

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.RobotConfig.setSlowDownFraction","params":[1,0.8],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":0}
```

8.6.7.2.73 setTcpForceOffset()

```
int arcs::common_interface::RobotConfig::setTcpForceOffset (
    const std::vector< double > & force_offset)
```

Set the TCP force/torque offset.

Parameters

force_offset	TCP force/torque offset
--------------	-------------------------

Returns

Returns 0 on success; error code on failure

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
setTcpForceOffset(self: pyaubo_sdk.RobotConfig, arg0: List[float]) -> int
```

Lua function prototype

```
setTcpForceOffset(force_offset: table) -> nil
```

Lua example

```
setTcpForceOffset({0.0,0.0,0.0,0.0,0.0,0.0,0.0})
```

8.6.7.2.74 setTcpForceSensorPose()

```
int arcs::common_interface::RobotConfig::setTcpForceSensorPose (
    const std::vector< double > & sensor_pose)
```

Set the sensor mounting pose.

Parameters

sensor_pose	Sensor mounting pose
-------------	----------------------

Returns

Returns 0 on success; error code on failure AUBO_BUSY AUBO_BAD_STATE -AUBO_INVL↵
_ARGUMENT -AUBO_BAD_STATE

Exceptions

arcs::common_interface::AuboException	
---	--

8.6.7.2.75 setTcpOffset()

```
int arcs::common_interface::RobotConfig::setTcpOffset (
    const std::vector< double > & offset)
```

Set the current TCP offset.

Parameters

offset	The current TCP offset, in the form (x, y, z, rx, ry, rz)
--------	---

Returns

Returns 0 on success; error code on failure AUBO_BUSY AUBO_BAD_STATE -AUBO_INVL↵
_ARGUMENT -AUBO_BAD_STATE

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
setTcpOffset(self: pyaubo_sdk.RobotConfig, arg0: List[float]) -> int
```

Lua function prototype

```
setTcpOffset(offset: table) -> nil
```

Lua example

```
setTcpOffset({0.0,0.0,0.0,0.0,0.0,0.0})
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.RobotConfig.setTcpOffset","params":[[0.0,0.0,0.0,0.0,0.0,0.0]]↵  
,"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":0}
```

8.6.7.2.76 setToolInertial()

```
int arcs::common_interface::RobotConfig::setToolInertial (
    double m,
    const std::vector< double > & com,
    const std::vector< double > & inertial)
```

Set tool mass, center of mass, and inertia.

Parameters

m	Tool mass
com	Center of mass
inertial	Inertia

Returns

Returns 0 on success; error code on failure AUBO_BUSY AUBO_BAD_STATE -AUBO_INVL↵
_ARGUMENT -AUBO_BAD_STATE

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
setToolInertial(self: pyaubo_sdk.RobotConfig, arg0: float, arg1: List[float], arg2: List[float]) ->
int
```

Lua function prototype

```
setToolInertial(m: number, com: table, inertial: table) -> nil
```

Lua example

```
setToolInertial(3, {0,0,0}, {0,0,0}, {0,0,0,0,0,0,0,0})
```

8.6.7.2.77 setWorkObjectData()

```
int arcs::common_interface::RobotConfig::setWorkObjectData (
    const WObjectData & wobj)
```

Set work object data.

All programmed points are based on the work object coordinate system.

8.6.7.2.78 toolSpaceInRange()

```
bool arcs::common_interface::RobotConfig::toolSpaceInRange (
    const std::vector< double > & pose)
```

Whether the end-effector pose is within the safety range.

Parameters

pose	End-effector pose
------	-------------------

Returns

Returns true if within the safety range; otherwise returns false

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
toolSpaceInRange(self: pyaubo_sdk.RobotConfig, arg0: List[float]) -> bool
```

Lua function prototype

```
toolSpaceInRange(pose: table) -> boolean
```

Lua example

```
SpaceInRange = toolSpaceInRange({0.58712,-0.15775, 0.48703, 2.76, 0.344, 1.432})
```

JSON-RPC request example

```
{ "jsonrpc": "2.0", "method": "rob1.RobotConfig.toolSpaceInRange", "params": [[0.58712, -0.15775, 0.48703, 2.76, 0.344, 1.432]], "id": 1 }
```

JSON-RPC response example

```
{ "id": 1, "jsonrpc": "2.0", "result": true }
```

8.6.8 RobotManage ()

[RobotManage](#).

Collaboration diagram for RobotManage ():



Functions

- int [arcs::common_interface::RobotManage::poweron](#) ()
Initiate robot power-on request
- int [arcs::common_interface::RobotManage::startup](#) ()
Initiate robot startup request
- int [arcs::common_interface::RobotManage::releaseRobotBrake](#) ()
Initiate robot brake release request
- int [arcs::common_interface::RobotManage::lockRobotBrake](#) ()
Initiate robot brake lock request
- int [arcs::common_interface::RobotManage::poweroff](#) ()
Initiate robot power-off request
- int [arcs::common_interface::RobotManage::backdrive](#) (bool enable)
Initiate robot backdrive request
- int [arcs::common_interface::RobotManage::freedrive](#) (bool enable)
Initiate robot freedrive request This interface is deprecated after software version 0.31.x, use [handguideMode](#) instead: [handguideMode](#)({1,1,1,1,1}, {0,0,0,0,0})
- int [arcs::common_interface::RobotManage::setHandguideParams](#) (const std::vector< int > &freeAxes, const std::vector< double > &feature)
Advanced hand-guiding mode
- std::vector< int > [arcs::common_interface::RobotManage::getHandguideFreeAxes](#) ()
Get Axes that can be moved
- std::vector< double > [arcs::common_interface::RobotManage::getHandguideFeature](#) ()
Get the drag reference coordinate system
- int [arcs::common_interface::RobotManage::handguideMode](#) (const std::vector< int > &freeAxes, const std::vector< double > &feature)
Advanced hand-guiding mode
- int [arcs::common_interface::RobotManage::exitHandguideMode](#) ()
Exit hand-guiding mode
- int [arcs::common_interface::RobotManage::getHandguideStatus](#) ()
Get the status of the hand-guiding device (whether it is in a singular space)
- int [arcs::common_interface::RobotManage::getHandguideTrigger](#) ()
Get the trigger source of the hand-guiding device
- bool [arcs::common_interface::RobotManage::isHandguideEnabled](#) ()
Get the hand-guiding enable status
- int [arcs::common_interface::RobotManage::setSim](#) (bool enable)
Initiate robot enter/exit simulation mode request
- int [arcs::common_interface::RobotManage::setOperationalMode](#) (OperationalModeType mode)
Set the robot operational mode
- OperationalModeType [arcs::common_interface::RobotManage::getOperationalMode](#) ()
Get the robot operational mode
- RobotControlModeType [arcs::common_interface::RobotManage::getRobotControlMode](#) ()
Get the control mode
- bool [arcs::common_interface::RobotManage::isFreedriveEnabled](#) ()
Whether the freedrive mode is enabled
- bool [arcs::common_interface::RobotManage::isBackdriveEnabled](#) ()
Whether the backdrive mode is enabled
- bool [arcs::common_interface::RobotManage::isSimulationEnabled](#) ()
Whether the simulation mode is enabled
- int [arcs::common_interface::RobotManage::setUnlockProtectiveStop](#) ()
Clear protective stop, including collision stop

- `int arcs::common_interface::RobotManage::restartInterfaceBoard ()`
Reset the safety interface board, usually called after the robot is powered off and needs to be reset, such as after emergency stop or fault.
- `int arcs::common_interface::RobotManage::recordCacheFree (const std::string &name)`
Free and clear recorded data of the specified memory cache
- `int arcs::common_interface::RobotManage::startRecordCache (const std::string &name)`
Start real-time trajectory recording to memory cache (no file output)
- `int arcs::common_interface::RobotManage::stopRecordCache ()`
Stop current real-time trajectory recording to memory cache
- `int arcs::common_interface::RobotManage::pauseRecordCache (bool pause)`
Pause or resume current real-time trajectory recording to memory cache
- `int arcs::common_interface::RobotManage::getRecordCache (const std::string &name, size_t frames=0)`
Get recorded data from the specified memory cache
- `int arcs::common_interface::RobotManage::startRecord (const std::string &file_name)`
Start real-time trajectory recording
- `int arcs::common_interface::RobotManage::stopRecord ()`
Stop real-time recording
- `int arcs::common_interface::RobotManage::pauseRecord (bool pause)`
Pause real-time recording
- `int arcs::common_interface::RobotManage::setLinkModeEnable (bool enable)`
Initiate robot enter/exit link mode request.
- `bool arcs::common_interface::RobotManage::isLinkModeEnabled ()`
Whether the link mode is enabled.
- `int arcs::common_interface::RobotManage::generateDiagnoseFile (const std::string &reason)`
Manually trigger the generation of a diagnostic file

8.6.8.1 Detailed Description

[RobotManage](#).

8.6.8.2 Function Documentation

8.6.8.2.1 backdrive()

```
int arcs::common_interface::RobotManage::backdrive (
    bool enable)
```

Initiate robot backdrive request

Parameters

enable	
--------	--

Returns

— Returns 0 on success; error code on failure AUBO_BUSY AUBO_BAD_STATE AUBO_INVL⇄
_ARGUMENT AUBO_BAD_STATE Generated by Doxygen

Exceptions

<code>arcs::common_interface::AuboException</code>	
--	--

Python function prototype

```
backdrive(self: pyaubo_sdk.RobotManage, arg0: bool) -> int
```

Lua function prototype

```
backdrive(enable: boolean) -> number
```

Lua example

```
num = backdrive(false)
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.RobotManage.backdrive","params":[false],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":0}
```

C++ example

```
auto rpc_cli = std::make_shared<RpcClient>();
auto robot_name = rpc_cli->getRobotNames().front();
rpc_cli->getRobotInterface(robot_name)->getRobotManage()->backdrive(true);
```

8.6.8.2.2 exitHandguideMode()

```
int arcs::common_interface::RobotManage::exitHandguideMode ()
```

Exit hand-guiding mode

Note

Requires software version 0.31.x or later

Returns

Returns 0 on success; error code on failure AUBO_BUSY AUBO_BAD_STATE -AUBO_BAD↵
_STATE

Exceptions

arcs::common_interface::AuboException	
---	--

8.6.8.2.3 freedrive()

```
int arcs::common_interface::RobotManage::freedrive (
    bool enable)
```

Initiate robot freedrive request This interface is deprecated after software version 0.31.x, use handguide↵
Mode instead: handguideMode({1,1,1,1,1}, {0,0,0,0,0})

Parameters

enable	
--------	--

Returns

Returns 0 on success; error code on failure AUBO_BUSY AUBO_BAD_STATE -AUBO_INVL↵
_ARGUMENT -AUBO_BAD_STATE

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
freedrive(self: pyaubo_sdk.RobotManage, arg0: bool) -> int
```

Lua function prototype

```
freedrive(enable: boolean) -> number
```

Lua example

```
num = freedrive(false)
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.RobotManage.freedrive","params":[true],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":0}
```

C++ example

```
auto rpc_cli = std::make_shared<RpcClient>();
auto robot_name = rpc_cli->getRobotNames().front();
rpc_cli->getRobotInterface(robot_name)->getRobotManage()->freedrive(true);
```


8.6.8.2.4 generateDiagnoseFile()

```
int arcs::common_interface::RobotManage::generateDiagnoseFile (
    const std::string & reason)
```

Manually trigger the generation of a diagnostic file

Returns

Returns 0 if the command is successfully issued; error code on failure.

-AUBO_BAD_STATE: The current state of the [RuntimeMachine](#) is not Stopped, the firmware upgrade request is rejected. The value of AUBO_BAD_STATE is 1.

-AUBO_TIMEOUT: Timeout. The value of AUBO_TIMEOUT is 4.

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
generateDiagnoseFile(self: pyaubo_sdk.RobotManage, arg0: str) -> int
```

Lua function prototype

```
generateDiagnoseFile(reason: string) -> nil
```

Lua example

```
generateDiagnoseFile("reason")
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.RobotManage.generateDiagnoseFile","params":["reason"],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":false}
```

C++ example

```
auto rpc_cli = std::make_shared<RpcClient>();
auto robot_name = rpc_cli->getRobotNames().front();
bool isEnabled =
    rpc_cli->getRobotInterface(robot_name)->getRobotManage()->generateDiagnoseFile("reason");
```

8.6.8.2.5 getHandguideFeature()

```
std::vector< double > arcs::common_interface::RobotManage::getHandguideFeature ()
```

Get the drag reference coordinate system

8.6.8.2.6 getHandguideFreeAxes()

```
std::vector< int > arcs::common_interface::RobotManage::getHandguideFreeAxes ()
```

Get Axes that can be moved

8.6.8.2.7 getHandguideStatus()

```
int arcs::common_interface::RobotManage::getHandguideStatus ()
```

Get the status of the hand-guiding device (whether it is in a singular space)

Note

Not implemented yet

Returns

- 0 - Normal operation.
- 1 - Approaching singular space.
- 2 - Extremely close to a singularity, large hand-guiding damping will occur.

Exceptions

arcs::common_interface::AuboException	
---	--

8.6.8.2.8 getHandguideTrigger()

```
int arcs::common_interface::RobotManage::getHandguideTrigger ()
```

Get the trigger source of the hand-guiding device

Note

Not implemented yet

Returns

Exceptions

arcs::common_interface::AuboException	
---	--

8.6.8.2.9 getOperationalMode()

[OperationalModeType](#) arcs::common_interface::RobotManage::getOperationalMode ()

Get the robot operational mode

Returns

Robot operational mode

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

getOperationalMode(self: pyaubo_sdk.RobotManage) -> [arcs::common_interface::OperationalModeType](#)

Lua function prototype

[getOperationalMode\(\)](#) -> number

Lua example

```
num = getOperationalMode\(\)
```

JSON-RPC request example

```
{ "jsonrpc": "2.0", "method": "rob1.RobotManage.getOperationalMode", "params": [], "id": 1 }
```

JSON-RPC response example

```
{ "id": 1, "jsonrpc": "2.0", "result": "Manual" }
```

C++ example

```
auto rpc_cli = std::make_shared<RpcClient>();
auto robot_name = rpc_cli->getRobotNames().front();
OperationalModeType mode =
    robot_manage.getRobotInterface(robot_name)->getRobotManage()->getOperationalMode();
```

8.6.8.2.10 getRecordCache()

Parameters

name	Cache name
frames	Frame limit: <ul style="list-style-type: none"> frames=0: get all recorded frames (1 frame = 1 control cycle) frames>0: get up to the latest 'frames' frames

Returns

Returns the number of frames fetched (≥ 0) on success; negative error code on failure

Exceptions

arcs::common_interface::AuboException	
---	--

Lua function prototype

```
getRecordCache(name: string, frames: number = 0) -> number
```

Lua example

```
n = getRecordCache("rec") -- all frames, returns count
n = getRecordCache("rec", 2000) -- latest 2000 frames, returns count
```

8.6.8.2.11 getRobotControlMode()

[RobotControlModeType](#) `arcs::common_interface::RobotManage::getRobotControlMode ()`

Get the control mode

Returns

Control mode

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
getRobotControlMode(self: pyaubo_sdk.RobotManage) -> arcs::common_interface::RobotControlModeType
```

Lua function prototype

```
getRobotControlMode() -> number
```

Lua example

```
num = getRobotControlMode()
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.RobotManage.getRobotControlMode","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":"Position"}
```

C++ example

```
auto rpc_cli = std::make_shared<RpcClient>();
auto robot_name = rpc_cli->getRobotNames().front();
RobotControlModeType mode =
rpc_cli->getRobotInterface(robot_name)->getRobotManage()->getRobotControlMode();
```

8.6.8.2.12 handguideMode()

```
int arcs::common_interface::RobotManage::handguideMode (
    const std::vector< int > & freeAxes,
    const std::vector< double > & feature)
```

Advanced hand-guiding mode

Parameters

freeAxes	Axes that can be moved: 0-cannot move, 1-can move
feature	If the dimension is 0, it means hand-guiding based on the TCP coordinate system

Returns

Returns 0 on success; error code on failure AUBO_BUSY AUBO_BAD_STATE -AUBO_INVL↵
_ARGUMENT -AUBO_BAD_STATE

Exceptions

arcs::common_interface::AuboException	
---	--

8.6.8.2.13 isBackdriveEnabled()

```
bool arcs::common_interface::RobotManage::isBackdriveEnabled ()
```

Whether the backdrive mode is enabled

Returns

Returns true if enabled; otherwise returns false

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
isBackdriveEnabled(self: pyaubo_sdk.RobotManage) -> bool
```

Lua function prototype

```
isBackdriveEnabled\(\) -> boolean
```

Lua example

```
BackdriveEnabled = isBackdriveEnabled\(\)
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.RobotManage.isBackdriveEnabled","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":false}
```

C++ example

```
auto rpc_cli = std::make_shared<RpcClient>();
auto robot_name = rpc_cli->getRobotNames().front();
bool isEnabled =
rpc_cli->getRobotInterface(robot_name)->getRobotManage()->isBackdriveEnabled();
```

Generated by Doxygen

Exceptions

<code>arcs::common_interface::AuboException</code>	
--	--

Python function prototype

```
isFreedriveEnabled(self: pyaubo_sdk.RobotManage) -> bool
```

Lua function prototype

```
isFreedriveEnabled() -> boolean
```

Lua example

```
FreedriveEnabled = isFreedriveEnabled()
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.RobotManage.isFreedriveEnabled","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":false}
```

C++ example

```
auto rpc_cli = std::make_shared<RpcClient>();
auto robot_name = rpc_cli->getRobotNames().front();
bool isEnabled =
rpc_cli->getRobotInterface(robot_name)->getRobotManage()->isFreedriveEnabled();
```

8.6.8.2.15 isHandguideEnabled()

```
bool arcs::common_interface::RobotManage::isHandguideEnabled ()
```

Get the hand-guiding enable status

Returns

Returns true if enabled; false if disabled

Exceptions

arcs::common_interface::AuboException	
---	--

Lua function prototype

```
isHandguideEnabled() -> boolean
```

Lua example

```
Handguide = isHandguideEnabled()
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.RobotManage.isHandguideEnabled","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":false}
```

8.6.8.2.16 isLinkModeEnabled()

```
bool arcs::common_interface::RobotManage::isLinkModeEnabled ()
```

Whether the link mode is enabled.

In link mode, users can control the robot via external IO (users can configure the IO functions).

Returns

Returns true if enabled; otherwise returns false

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
isLinkModeEnabled(self: pyaubo_sdk.RobotManage) -> bool
```


Lua function prototype

[isLinkModeEnabled\(\)](#) -> boolean

Lua example

LinkModeEnabled = [isLinkModeEnabled\(\)](#)

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.RobotManage.isLinkModeEnabled","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":false}
```

C++ example

```
auto rpc_cli = std::make_shared<RpcClient>();
auto robot_name = rpc_cli->getRobotNames().front();
bool isEnabled =
rpc_cli->getRobotInterface(robot_name)->getRobotManage()->isLinkModeEnabled();
```

8.6.8.2.17 isSimulationEnabled()

bool arcs::common_interface::RobotManage::isSimulationEnabled ()

Whether the simulation mode is enabled

Returns

Returns true if enabled; otherwise returns false

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

isSimulationEnabled(self: pyaubo_sdk.RobotManage) -> bool

Lua function prototype

[isSimulationEnabled\(\)](#) -> boolean

Lua example

```
SimulationEnabled = isSimulationEnabled()
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.RobotManage.isSimulationEnabled","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":true}
```

C++ example

```
auto rpc_cli = std::make_shared<RpcClient>();
auto robot_name = rpc_cli->getRobotNames().front();
bool isEnabled =
rpc_cli->getRobotInterface(robot_name)->getRobotManage()->isSimulationEnabled();
```

8.6.8.2.18 lockRobotBrake()

```
int arcs::common_interface::RobotManage::lockRobotBrake ()
```

Initiate robot brake lock request

Returns

Returns 0 on success; error code on failure AUBO_BAD_STATE

Exceptions

arcs::common_interface::AuboException

Python function prototype

```
lockRobotBrake(self: pyaubo_sdk.RobotManage) -> int
```

Lua function prototype

```
lockRobotBrake() -> number
```

Lua example

```
num = lockRobotBrake()
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.RobotManage.lockRobotBrake","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":0}
```

C++ example

```
auto rpc_cli = std::make_shared<RpcClient>();
auto robot_name = rpc_cli->getRobotNames().front();
rpc_cli->getRobotInterface(robot_name)->getRobotManage()->lockRobotBrake();
```

8.6.8.2.19 pauseRecord()

```
int arcs::common_interface::RobotManage::pauseRecord (
    bool pause)
```

Pause real-time recording

Parameters

pause	
-------	--

Returns

Returns 0 on success; error code on failure AUBO_BUSY AUBO_BAD_STATE -AUBO_BAD↵
_STATE

Exceptions

arcs::common_interface::AuboException	
---	--

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.RobotManage.pauseRecord","params":[true],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":0}
```

8.6.8.2.20 pauseRecordCache()

```
int arcs::common_interface::RobotManage::pauseRecordCache (
    bool pause)
```

Pause or resume current real-time trajectory recording to memory cache

Parameters

pause	true to pause cache recording; false to resume cache recording
-------	--

Exceptions

arcs::common_interface::AuboException	
---	--

Lua function prototype

```
pauseRecordCache(pause: boolean) -> nil
```

Lua example

```
pauseRecordCache(true) – pause pauseRecordCache(false) – resume
```

8.6.8.2.21 poweroff()

```
int arcs::common_interface::RobotManage::poweroff ()
```

Initiate robot power-off request

Returns

Returns 0 on success; error code on failure AUBO_BAD_STATE

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
poweroff(self: pyaubo_sdk.RobotManage) -> int
```

Lua function prototype

```
poweroff\(\) -> number
```

Lua example

```
num = poweroff\(\)
```

JSON-RPC request example

```
{"jsonrpc": "2.0", "method": "rob1.RobotManage.poweroff", "params": [], "id": 1}
```

JSON-RPC response example

```
{"id": 1, "jsonrpc": "2.0", "result": 0}
```

C++ example

```
auto rpc_cli = std::make_shared<RpcClient>();
auto robot_name = rpc_cli->getRobotNames().front();
rpc_cli->getRobotInterface(robot_name)->getRobotManage()->poweroff();
```

8.6.8.2.22 poweron()

```
int arcs::common_interface::RobotManage::poweron ()
```

Initiate robot power-on request

Returns

Returns 0 on success; error code on failure AUBO_BAD_STATE

Exceptions

<code>arcs::common_interface::AuboException</code>	
--	--

Python function prototype

```
poweron(self: pyaubo_sdk.RobotManage) -> int
```

Lua function prototype

```
poweron() -> number
```

Lua example

```
num = poweron()
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.RobotManage.poweron","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":0}
```

C++ example

```
auto rpc_cli = std::make_shared<RpcClient>();
auto robot_name = rpc_cli->getRobotNames().front();
rpc_cli->getRobotInterface(robot_name)->getRobotManage()->poweron();
```

8.6.8.2.23 recordCacheFree()

```
int arcs::common_interface::RobotManage::recordCacheFree (
    const std::string & name)
```

Parameters

name	Cache name
------	------------

Returns

Returns 0 on success; negative error code on failure

Exceptions

arcs::common_interface::AuboException	
---	--

Lua function prototype

```
recordCacheFree(name: string) -> nil
```

Lua example

```
recordCacheFree("rec")
```

8.6.8.2.24 releaseRobotBrake()

```
int arcs::common_interface::RobotManage::releaseRobotBrake ()
```

Initiate robot brake release request

Returns

Returns 0 on success; error code on failure AUBO_BAD_STATE

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
releaseRobotBrake(self: pyaubo_sdk.RobotManage) -> int
```

Lua function prototype

```
releaseRobotBrake() -> number
```

Lua example

```
num = releaseRobotBrake()
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.RobotManage.releaseRobotBrake","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":0}
```

C++ example

```
auto rpc_cli = std::make_shared<RpcClient>();
auto robot_name = rpc_cli->getRobotNames().front();
rpc_cli->getRobotInterface(robot_name)->getRobotManage()->releaseRobotBrake();
```

8.6.8.2.25 restartInterfaceBoard()

```
int arcs::common_interface::RobotManage::restartInterfaceBoard ()
```

Reset the safety interface board, usually called after the robot is powered off and needs to be reset, such as after emergency stop or fault.

Returns

Returns 0 on success; error code on failure AUBO_BUSY AUBO_BAD_STATE -AUBO_BAD↵
_STATE

Exceptions

arcs::common_interface::AuboException	
---------------------------------------	--

Python function prototype

```
restartInterfaceBoard(self: pyaubo_sdk.RobotManage) -> int
```

Lua function prototype

```
restartInterfaceBoard() -> number
```

Lua example

```
num = restartInterfaceBoard()
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.RobotManage.restartInterfaceBoard","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":0}
```

C++ example

```
auto rpc_cli = std::make_shared<RpcClient>();
auto robot_name = rpc_cli->getRobotNames().front();
rpc_cli->getRobotInterface(robot_name)->getRobotManage()->restartInterfaceBoard();
```

8.6.8.2.26 setHandguideParams()

```
int arcs::common_interface::RobotManage::setHandguideParams (
    const std::vector< int > & freeAxes,
    const std::vector< double > & feature)
```

Advanced hand-guiding mode

Parameters

freeAxes	Axes that can be moved: 0-cannot move, 1-can move
feature	If the dimension is 0, it means hand-guiding based on the TCP coordinate system

Returns

Returns 0 on success; error code on failure AUBO_BUSY AUBO_BAD_STATE -AUBO_INVL↔
_ARGUMENT -AUBO_BAD_STATE

Exceptions

arcs::common_interface::AuboException	
---	--

8.6.8.2.27 setLinkModeEnable()

```
int arcs::common_interface::RobotManage::setLinkModeEnable (
    bool enable)
```

Generated by Doxygen

Initiate robot enter/exit link mode request.

Parameters

enable	
--------	--

Returns

Returns 0 on success; error code on failure AUBO_BUSY AUBO_REQUEST_IGNORE -AUBO_↵
_BAD_STATE

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
setLinkModeEnable(self: pyaubo_sdk.RobotManage, arg0: bool) -> int
```

Lua function prototype

```
setLinkModeEnable(enable: boolean) -> number
```

Lua example

```
num = setLinkModeEnable(true)
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.RobotManage.setLinkModeEnable","params":[true,"id":1]}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":0}
```

C++ example

```
auto rpc_cli = std::make_shared<RpcClient>();
auto robot_name = rpc_cli->getRobotNames().front();
rpc_cli->getRobotInterface(robot_name)->getRobotManage()->setLinkModeEnable(true);
```

8.6.8.2.28 setOperationalMode()

```
int arcs::common_interface::RobotManage::setOperationalMode (
```

Generated by Doxygen [OperationalModeType](#) mode)

Set the robot operational mode

Parameters

mode	Operational mode
------	------------------

Returns

Returns 0 on success; error code on failure -AUBO_BAD_STATE

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
setOperationalMode(self: pyaubo_sdk.RobotManage, arg0: arcs::common_interface::OperationalModeType)
-> int
```

Lua function prototype

```
setOperationalMode(mode: number) -> number
```

Lua example

```
num = setOperationalMode("Manual")
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.RobotManage.setOperationalMode","params":["Manual"],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":0}
```

C++ example

```
auto rpc_cli = std::make_shared<RpcClient>();
auto robot_name = rpc_cli->getRobotNames().front();
rpc_cli->getRobotInterface(robot_name)->getRobotManage()->setOperationalMode(OperationalModeType::Automatic);
```

8.6.8.2.29 setSim()

```
int arcs::common_interface::RobotManage::setSim (
    bool enable)
```

Generated by Doxygen

Initiate robot enter/exit simulation mode request

Parameters

enable	
--------	--

Returns

Returns 0 on success; error code on failure AUBO_BUSY AUBO_BAD_STATE -AUBO_INVL↔
_ARGUMENT -AUBO_BAD_STATE

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
setSim(self: pyaubo_sdk.RobotManage, arg0: bool) -> int
```

Lua function prototype

```
setSim(enable: boolean) -> number
```

Lua example

```
num = setSim(true)
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.RobotManage.setSim","params":[true],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":0}
```

C++ example

```
auto rpc_cli = std::make_shared<RpcClient>();
auto robot_name = rpc_cli->getRobotNames().front();
rpc_cli->getRobotInterface(robot_name)->getRobotManage()->setSim(true);
```

8.6.8.2.30 setUnlockProtectiveStop()

```
int arcs::common_interface::RobotManage::setUnlockProtectiveStop ()
```

Generated by Doxygen

Clear protective stop, including collision stop

Returns

Exceptions

<code>arcs::common_interface::AuboException</code>	
--	--

Python function prototype

```
setUnlockProtectiveStop(self: pyaubo_sdk.RobotManage) -> int
```

Lua function prototype

```
setUnlockProtectiveStop() -> number
```

Lua example

```
num = setUnlockProtectiveStop()
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.RobotManage.setUnlockProtectiveStop","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":0}
```

C++ example

```
auto rpc_cli = std::make_shared<RpcClient>();
auto robot_name = rpc_cli->getRobotNames().front();
rpc_cli->getRobotInterface(robot_name)->getRobotManage()->setUnlockProtectiveStop();
```

8.6.8.2.31 startRecord()

```
int arcs::common_interface::RobotManage::startRecord (
    const std::string & file_name)
```

Start real-time trajectory recording

Parameters

<code>file_name</code>	
------------------------	--

Returns

Returns 0 on success; error code on failure AUBO_BUSY AUBO_BAD_STATE AUBO_INVALID_ARGUMENT AUBO_BAD_STATE

Exceptions

arcs::common_interface::AuboException	
---	--

Lua function prototype

```
startRecord(fiel_name: string) -> nil
```

Lua example

```
startRecord("traje.csv")
```

8.6.8.2.32 startRecordCache()

```
int arcs::common_interface::RobotManage::startRecordCache (
    const std::string & name)
```

Start real-time trajectory recording to memory cache (no file output)

Parameters

name	Cache name used to index recorded data in memory. Empty string returns invalid argument.
------	--

Returns

Returns 0 on success; negative error code on failure

Exceptions

arcs::common_interface::AuboException	
---	--

Lua function prototype

```
startRecordCache(name: string) -> nil
```

Lua example

```
startRecordCache("rec")
```

Exceptions

<code>arcs::common_interface::AuboException</code>	
--	--

Python function prototype

```
startup(self: pyaubo_sdk.RobotManage) -> int
```

Lua function prototype

```
startup() -> number
```

Lua example

```
num = startup()
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.RobotManage.startup","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":0}
```

C++ example

```
auto rpc_cli = std::make_shared<RpcClient>();
auto robot_name = rpc_cli->getRobotNames().front();
rpc_cli->getRobotInterface(robot_name)->getRobotManage()->startup();
```

8.6.8.2.34 stopRecord()

```
int arcs::common_interface::RobotManage::stopRecord ()
```

Stop real-time recording

Returns

Returns 0 on success; error code on failure AUBO_BUSY AUBO_BAD_STATE -AUBO_BAD↵
_STATE

Exceptions

arcs::common_interface::AuboException	
---	--

Lua function prototype

```
stopRecord() -> nil
```

Lua example

```
stopRecord()
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.RobotManage.stopRecord","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":0}
```

8.6.8.2.35 stopRecordCache()

```
int arcs::common_interface::RobotManage::stopRecordCache ()
```

Stop current real-time trajectory recording to memory cache

Returns

Returns 0 on success; negative error code on failure

Exceptions

arcs::common_interface::AuboException	
---	--

Lua function prototype

```
stopRecordCache() -> nil
```

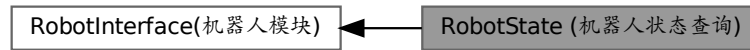
Lua example

```
stopRecordCache()
```

8.6.9 RobotState ()

[RobotState](#).

Collaboration diagram for RobotState ():



Functions

- [RobotModeType](#) `arcs::common_interface::RobotState::getRobotModeType ()`
Get the robot mode state
- [SafetyModeType](#) `arcs::common_interface::RobotState::getSafetyModeType ()`
Get the safety mode
- `bool` [arcs::common_interface::RobotState::isPowerOn \(\)](#)
Get the robot power-on state
- `bool` [arcs::common_interface::RobotState::isSteady \(\)](#)
Whether the robot has stopped
- `bool` [arcs::common_interface::RobotState::isCollisionOccurred \(\)](#)
Whether a collision has occurred
- `bool` [arcs::common_interface::RobotState::isWithinSafetyLimits \(\)](#)
Whether the robot is within safety limits
- `std::vector< double >` [arcs::common_interface::RobotState::getTcpPose \(\)](#)
Get the current TCP pose.
- `std::vector< double >` [arcs::common_interface::RobotState::getActualTcpOffset \(\)](#)
Get the current TCP offset, which is the TCP offset used by the pose returned from `getTcpPose`
- `std::vector< double >` [arcs::common_interface::RobotState::getTargetTcpPose \(\)](#)
Get the next target waypoint.
- `std::vector< double >` [arcs::common_interface::RobotState::getToolPose \(\)](#)
Get the tool pose (without TCP offset)
- `std::vector< double >` [arcs::common_interface::RobotState::getTcpSpeed \(\)](#)
Get the TCP speed
- `std::vector< double >` [arcs::common_interface::RobotState::getTcpForce \(\)](#)
Get the TCP force/torque
- `std::vector< double >` [arcs::common_interface::RobotState::getElbowPosition \(\)](#)
Get the position of the elbow
- `std::vector< double >` [arcs::common_interface::RobotState::getElbowVelocity \(\)](#)
Get the elbow velocity
- `std::vector< double >` [arcs::common_interface::RobotState::getBaseForce \(\)](#)
Get the base force/torque
- `std::vector< double >` [arcs::common_interface::RobotState::getTcpTargetPose \(\)](#)
Get the last sent TCP target pose
- `std::vector< double >` [arcs::common_interface::RobotState::getTcpTargetSpeed \(\)](#)
Get the TCP target speed

- `std::vector< double > arcs::common_interface::RobotState::getTcpTargetForce ()`
Get the TCP target force/torque
- `std::vector< JointStateType > arcs::common_interface::RobotState::getJointState ()`
Get the joint state of the manipulator
- `std::vector< JointServoModeType > arcs::common_interface::RobotState::getJointServoMode ()`
Get the servo state of the joints
- `std::vector< double > arcs::common_interface::RobotState::getJointPositions ()`
Get the joint positions of the manipulator
- `std::vector< double > arcs::common_interface::RobotState::getJointPositionsHistory (int steps)`
Get the historical joint positions of the manipulator
- `std::vector< double > arcs::common_interface::RobotState::getJointSpeeds ()`
Get the joint speeds of the manipulator
- `std::vector< double > arcs::common_interface::RobotState::getJointAccelerations ()`
Get the joint accelerations of the manipulator
- `std::vector< double > arcs::common_interface::RobotState::getJointTorqueSensors ()`
Get the joint torques of the manipulator
- `std::vector< double > arcs::common_interface::RobotState::getJointContactTorques ()`
Get the joint contact torques (external torques) of the manipulator
- `std::vector< double > arcs::common_interface::RobotState::getJointGravityTorques ()`
Get the joint gravity torque of the manipulator
- `std::vector< double > arcs::common_interface::RobotState::getTcpForceSensors ()`
Get the TCP force sensor readings
- `std::vector< double > arcs::common_interface::RobotState::getJointCurrents ()`
Get the joint currents of the manipulator
- `std::vector< double > arcs::common_interface::RobotState::getJointVoltages ()`
Get the joint voltages of the manipulator
- `std::vector< double > arcs::common_interface::RobotState::getJointTemperatures ()`
Get the joint temperatures of the manipulator
- `std::vector< std::string > arcs::common_interface::RobotState::getJointUniqueIds ()`
Get the globally unique IDs of the joints
- `std::vector< int > arcs::common_interface::RobotState::getJointFirmwareVersions ()`
Get the joint firmware versions
- `std::vector< int > arcs::common_interface::RobotState::getJointHardwareVersions ()`
Get the joint hardware versions
- `std::string arcs::common_interface::RobotState::getMasterBoardUniqueId ()`
Get the globally unique ID of the MasterBoard
- `int arcs::common_interface::RobotState::getMasterBoardFirmwareVersion ()`
Get the MasterBoard firmware version
- `int arcs::common_interface::RobotState::getMasterBoardHardwareVersion ()`
Get the MasterBoard hardware version
- `std::string arcs::common_interface::RobotState::getSlaveBoardUniqueId ()`
Get the globally unique ID of the SlaveBoard
- `int arcs::common_interface::RobotState::getSlaveBoardFirmwareVersion ()`
Get the SlaveBoard firmware version
- `int arcs::common_interface::RobotState::getSlaveBoardHardwareVersion ()`
Get the SlaveBoard hardware version
- `std::string arcs::common_interface::RobotState::getToolUniqueId ()`
Get the globally unique ID of the tool
- `int arcs::common_interface::RobotState::getToolFirmwareVersion ()`
Get the tool firmware version
- `int arcs::common_interface::RobotState::getToolHardwareVersion ()`

- Get the tool hardware version
 - `int arcs::common_interface::RobotState::getToolCommMode ()`
- Get the tool communication mode
 - `std::string arcs::common_interface::RobotState::getPedestalUniqueId ()`
- Get the globally unique ID of the pedestal
 - `int arcs::common_interface::RobotState::getPedestalFirmwareVersion ()`
- Get the pedestal firmware version
 - `int arcs::common_interface::RobotState::getPedestalHardwareVersion ()`
- Get the pedestal hardware version
 - `std::vector< double > arcs::common_interface::RobotState::getJointTargetPositions ()`
- Get the target joint positions (angles) of the manipulator
 - `std::vector< double > arcs::common_interface::RobotState::getJointTargetSpeeds ()`
- Get the target joint speeds of the manipulator
 - `std::vector< double > arcs::common_interface::RobotState::getJointTargetAccelerations ()`
- Get the target joint accelerations of the manipulator
 - `std::vector< double > arcs::common_interface::RobotState::getJointTargetTorques ()`
- Get the target joint torques of the manipulator
 - `std::vector< double > arcs::common_interface::RobotState::getJointTargetCurrents ()`
- Get the target joint currents of the manipulator
 - `bool arcs::common_interface::RobotState::isTeachPendantEnabled ()`
- Get whether the teach pendant is enabled.
 - `bool arcs::common_interface::RobotState::isToolFlangeEnabled ()`
- Get whether the tool flange is enabled.
 - `double arcs::common_interface::RobotState::getControlBoxTemperature ()`
- Get the control box temperature
 - `double arcs::common_interface::RobotState::getControlBoxHumidity ()`
- Get the control box humidity
 - `double arcs::common_interface::RobotState::getMainVoltage ()`
- Get the main bus voltage
 - `double arcs::common_interface::RobotState::getMainCurrent ()`
- Get the main bus current
 - `double arcs::common_interface::RobotState::getRobotVoltage ()`
- Get the robot voltage
 - `double arcs::common_interface::RobotState::getRobotCurrent ()`
- Get the robot current
 - `int arcs::common_interface::RobotState::getSlowDownLevel ()`
- Get the robot slow down level
 - `bool arcs::common_interface::RobotState::getTcpForceSensorStatus (const std::string &name)`
- Get the communication status of the tool force sensor

8.6.9.1 Detailed Description

[RobotState](#).

8.6.9.2 Function Documentation

8.6.9.2.1 getActualTcpOffset()

```
std::vector< double > arcs::common_interface::RobotState::getActualTcpOffset ()
```

Get the current TCP offset, which is the TCP offset used by the pose returned from getTcpPose

Returns

The current TCP offset

Lua function prototype

```
getActualTcpOffset() -> table
```

Lua example

```
ActualTcpOffset = getActualTcpOffset()
```

8.6.9.2.2 getBaseForce()

```
std::vector< double > arcs::common_interface::RobotState::getBaseForce ()
```

Get the base force/torque

Returns

Base force/torque

Exceptions

arcs::common_interface::AuboException	
---------------------------------------	--

Python function prototype

```
getBaseForce(self: pyaubo_sdk.RobotState) -> List[float]
```

Lua function prototype

```
getBaseForce() -> table
```

Lua example

```
BaseForce = getBaseForce()
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.RobotState.getBaseForce","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":[0.0,0.0,0.0,0.0,0.0,0.0]}
```

8.6.9.2.3 getControlBoxHumidity()

double arcs::common_interface::RobotState::getControlBoxHumidity ()

Get the control box humidity

Returns

Control box humidity

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

getControlBoxHumidity(self: pyaubo_sdk.RobotState) -> float

Lua function prototype

[getControlBoxHumidity\(\)](#) -> number

Lua example

ControlBoxHumidity = [getControlBoxHumidity\(\)](#)

JSON-RPC request example

```
{"jsonrpc": "2.0", "method": "rob1.RobotState.getControlBoxHumidity", "params": [], "id": 1}
```

JSON-RPC response example

```
{"id": 1, "jsonrpc": "2.0", "result": 20.0}
```

8.6.9.2.4 getControlBoxTemperature()

double arcs::common_interface::RobotState::getControlBoxTemperature ()

Get the control box temperature

Returns

Control box temperature

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
getControlBoxTemperature(self: pyaubo_sdk.RobotState) -> float
```

Lua function prototype

```
getControlBoxTemperature() -> number
```

Lua example

```
ControlBoxTemperature = getControlBoxTemperature()
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.RobotState.getControlBoxTemperature","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":25.0}
```

8.6.9.2.5 getElbowPosistion()

```
std::vector< double > arcs::common_interface::RobotState::getElbowPosistion ()
```

Get the position of the elbow

Returns

The position of the elbow

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
getElbowPosistion(self: pyaubo_sdk.RobotState) -> List[float]
```

Lua function prototype

```
getElbowPosistion() -> table
```

Lua example

```
ElbowPosistion = getElbowPosistion()
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.RobotState.getElbowPosistion","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":[0.07355755887512408,-0.1325,0.43200874126125227,-1.5707963267948968,0.433006344376404,0.0]}
```

8.6.9.2.6 getElbowVelocity()

```
std::vector< double > arcs::common_interface::RobotState::getElbowVelocity ()
```

Get the elbow velocity

Returns

Elbow velocity

Exceptions

arcs::common_interface::AuboException	
---------------------------------------	--

Python function prototype

```
getElbowVelocity(self: pyaubo_sdk.RobotState) -> List[float]
```

Lua function prototype

```
getElbowVelocity() -> table
```

Lua example

```
ElbowVelocity = getElbowVelocity()
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.RobotState.getElbowVelocity","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":[0.0,0.0,0.0,0.0,0.0,0.0]}
```

8.6.9.2.7 getJointAccelerations()

```
std::vector< double > arcs::common_interface::RobotState::getJointAccelerations ()
```

Get the joint accelerations of the manipulator

Returns

Joint accelerations of the manipulator

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
getJointAccelerations(self: pyaubo_sdk.RobotState) -> List[float]
```

Lua function prototype

```
getJointAccelerations\(\) -> table
```

Lua example

```
JointAccelerations = getJointAccelerations\(\)
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.RobotState.getJointAccelerations","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":[0.0,0.0,0.0,0.0,0.0,0.0]}
```

8.6.9.2.8 getJointContactTorques()

```
std::vector< double > arcs::common_interface::RobotState::getJointContactTorques ()
```

Get the joint contact torques (external torques) of the manipulator

Returns

Joint contact torques of the manipulator

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
getJointContactTorques(self: pyaubo_sdk.RobotState) -> List[float]
```

Lua function prototype

```
getJointContactTorques() -> table
```

Lua example

```
JointContactTorques = getJointContactTorques()
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.RobotState.getJointContactTorques","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":[0.0,0.0,0.0,0.0,0.0,0.0]}
```

8.6.9.2.9 getJointCurrents()

```
std::vector< double > arcs::common_interface::RobotState::getJointCurrents ()
```

Get the joint currents of the manipulator

Returns

Joint currents of the manipulator

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
getJointCurrents(self: pyaubo_sdk.RobotState) -> List[float]
```


Lua function prototype

`getJointCurrents()` -> table

Lua example

`JointCurrents = getJointCurrents()`

JSON-RPC request example

```
{ "jsonrpc": "2.0", "method": "rob1.RobotState.getJointCurrents", "params": [], "id": 1 }
```

JSON-RPC response example

```
{ "id": 1, "jsonrpc": "2.0", "result": [0.0, 1.25885009765625, -1.5289306640625, 0.71868896484375, 0.1007080078125, 0.3021240234375] }
```

8.6.9.2.10 getJointFirmwareVersions()

`std::vector< int > arcs::common_interface::RobotState::getJointFirmwareVersions ()`

Get the joint firmware versions

Returns

Joint firmware versions

Exceptions

<code>arcs::common_interface::AuboException</code>	
--	--

Python function prototype

`getJointFirmwareVersions(self: pyaubo_sdk.RobotState) -> List[int]`

Lua function prototype

`getJointFirmwareVersions()` -> table

Lua example

`JointFirmwareVersions = getJointFirmwareVersions()`

JSON-RPC request example

```
{ "jsonrpc": "2.0", "method": "rob1.RobotState.getJointFirmwareVersions", "params": [], "id": 1 }
```

JSON-RPC response example

```
{ "id": 1, "jsonrpc": "2.0", "result": [1000010, 1000010, 1000010, 1000010, 1000010, 1000010] }
```

8.6.9.2.11 `getJointGravityTorques()`

```
std::vector< double > arcs::common_interface::RobotState::getJointGravityTorques ()
```

Get the joint gravity torque of the manipulator

Returns

Joint gravity torque of the manipulator, containing torque values for 6 joints (unit: Nm)

Exceptions

<code>arcs::common_interface::AuboException</code>	
--	--

Python function prototype

```
getJointGravityTorques(self: pyaubo_sdk.RobotState) -> List[float]
```

Lua function prototype

```
getJointGravityTorques() -> table
```

Lua example

```
local jointGravityTorques = getJointGravityTorques()
```

JSON-RPC request example

```
{ "jsonrpc": "2.0", "method": "rob1.RobotState.getJointGravityTorques", "params": [], "id": 1 }
```

JSON-RPC response example

```
{ "id": 1, "jsonrpc": "2.0", "result": [1.23, -2.34, 3.45, -4.56, 0.78, -0.12] }
```

8.6.9.2.12 `getJointHardwareVersions()`

```
std::vector< int > arcs::common_interface::RobotState::getJointHardwareVersions ()
```

Get the joint hardware versions

Returns

Joint hardware versions

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
getJointHardwareVersions(self: pyaubo_sdk.RobotState) -> List[int]
```

Lua function prototype

```
getJointHardwareVersions() -> table
```

Lua example

```
JointHardwareVersions = getJointHardwareVersions()
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.RobotState.getJointHardwareVersions","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":[1000000,1000000,1004000,1004000,1004000,1004000]}
```

8.6.9.2.13 getJointPositions()

```
std::vector< double > arcs::common_interface::RobotState::getJointPositions ()
```

Get the joint positions of the manipulator

Returns

Joint positions of the manipulator

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
getJointPositions(self: pyaubo_sdk.RobotState) -> List[float]
```

Lua function prototype

```
getJointPositions() -> table
```

Lua example

```
JointPositions = getJointPositions()
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.RobotState.getJointPositions","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":[0.0,-0.26199241371495835,1.7418102574563423,0.4330197667082982,1.5707963267948966,0.0]}
```

8.6.9.2.14 getJointPositionsHistory()

```
std::vector< double > arcs::common_interface::RobotState::getJointPositionsHistory (
    int steps)
```

Get the historical joint positions of the manipulator

Parameters

steps	Number of steps
-------	-----------------

Returns

Historical joint positions of the manipulator

8.6.9.2.15 getJointServoMode()

```
std::vector< JointServoModeType > arcs::common_interface::RobotState::getJointServoMode ()
```

Get the servo state of the joints

Returns

The servo state of the joints

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
getJointServoMode(self: pyaubo_sdk.RobotState) -> List[arcs::common_interface::JointServoModeType]↵
```

Lua function prototype

```
getJointServoMode() -> table
```

Lua example

```
JointServoMode = getJointServoMode()
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.RobotState.getJointServoMode","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":["Position","Position","Position","Position","Position","Position","Position"]}↵
```

8.6.9.2.16 getJointSpeeds()

```
std::vector< double > arcs::common_interface::RobotState::getJointSpeeds ()
```

Get the joint speeds of the manipulator

Returns

Joint speeds of the manipulator

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
getJointSpeeds(self: pyaubo_sdk.RobotState) -> List[float]
```

Lua function prototype

```
getJointSpeeds() -> table
```

Lua example

```
JointSpeeds = getJointSpeeds()
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.RobotState.getJointSpeeds","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":[0.0,0.0,0.0,0.0,0.0,0.0]}
```

8.6.9.2.17 getJointState()

```
std::vector< JointStateType > arcs::common_interface::RobotState::getJointState ()
```

Get the joint state of the manipulator

Returns

Joint state of the manipulator

Exceptions

arcs::common_interface::AuboException	
---------------------------------------	--

Python function prototype

```
getJointState(self: pyaubo_sdk.RobotState) -> List[arcs::common_interface::JointStateType]
```

Lua function prototype

```
getJointState() -> table
```

Lua example

```
JointState = getJointState()
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.RobotState.getJointState","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":["Running","Running","Running","Running","Running","Running"]}↵
}
```

8.6.9.2.18 getJointTargetAccelerations()

```
std::vector< double > arcs::common_interface::RobotState::getJointTargetAccelerations ()
```

Get the target joint accelerations of the manipulator

Returns

Target joint accelerations of the manipulator

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
getJointTargetAccelerations(self: pyaubo_sdk.RobotState) -> List[float]
```

Lua function prototype

```
getJointTargetAccelerations\(\) -> table
```

Lua example

```
JointTargetAccelerations = getJointTargetAccelerations\(\)
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.RobotState.getJointTargetAccelerations","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":[0.0,-0.6737932929246071,-12.610253240108449,0.0,0.0,0.0]}
```

8.6.9.2.19 getJointTargetCurrents()

```
std::vector< double > arcs::common_interface::RobotState::getJointTargetCurrents ()
```

Get the target joint currents of the manipulator

Returns

Target joint currents of the manipulator

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
getJointTargetCurrents(self: pyaubo_sdk.RobotState) -> List[float]
```

Lua function prototype

```
getJointTargetCurrents() -> table
```

Lua example

```
JointTargetCurrents = getJointTargetCurrents()
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.RobotState.getJointTargetCurrents","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":[0.0,0.0,0.0,0.0,0.0,0.0]}
```

8.6.9.2.20 getJointTargetPositions()

```
std::vector< double > arcs::common_interface::RobotState::getJointTargetPositions ()
```

Get the target joint positions (angles) of the manipulator

Returns

Target joint positions (angles) of the manipulator

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
getJointTargetPositions(self: pyaubo_sdk.RobotState) -> List[float]
```


Lua function prototype

```
getJointTargetPositions() -> table
```

Lua example

```
JointTargetPositions = getJointTargetPositions()
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.RobotState.getJointTargetPositions","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":[0.0,-0.2619944355631239,1.7418124015308052,0.4330219266665035,1.↵  
5707963267948966,0.0]}
```

8.6.9.2.21 getJointTargetSpeeds()

```
std::vector< double > arcs::common_interface::RobotState::getJointTargetSpeeds ()
```

Get the target joint speeds of the manipulator

Returns

Target joint speeds of the manipulator

Exceptions

arcs::common_interface::AuboException	
---------------------------------------	--

Python function prototype

```
getJointTargetSpeeds(self: pyaubo_sdk.RobotState) -> List[float]
```

Lua function prototype

```
getJointTargetSpeeds() -> table
```

Lua example

```
JointTargetSpeeds = getJointTargetSpeeds()
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.RobotState.getJointTargetSpeeds","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":[0.0,0.00024227101509399773,0.0016521760307419697,0.0026521060731088397,0.↵  
0.0]}
```

8.6.9.2.22 getJointTargetTorques()

`std::vector< double > arcs::common_interface::RobotState::getJointTargetTorques ()`

Get the target joint torques of the manipulator

Returns

Target joint torques of the manipulator

Exceptions

<code>arcs::common_interface::AuboException</code>	
--	--

Python function prototype

`getJointTargetTorques(self: pyaubo_sdk.RobotState) -> List[float]`

Lua function prototype

`getJointTargetTorques() -> table`

Lua example

`JointTargetTorques = getJointTargetTorques()`

JSON-RPC request example

```
{"jsonrpc": "2.0", "method": "rob1.RobotState.getJointTargetTorques", "params": [], "id": 1}
```

JSON-RPC response example

```
{"id": 1, "jsonrpc": "2.0", "result": [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]}
```

8.6.9.2.23 getJointTemperatures()

`std::vector< double > arcs::common_interface::RobotState::getJointTemperatures ()`

Get the joint temperatures of the manipulator

Returns

Joint temperatures of the manipulator

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
getJointTemperatures(self: pyaubo_sdk.RobotState) -> List[float]
```

Lua function prototype

```
getJointTemperatures() -> table
```

Lua example

```
JointTemperatures = getJointTemperatures()
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.RobotState.getJointTemperatures","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":[38.0,38.0,38.0,39.0,38.0,39.0]}
```

8.6.9.2.24 getJointTorqueSensors()

```
std::vector< double > arcs::common_interface::RobotState::getJointTorqueSensors ()
```

Get the joint torques of the manipulator

Returns

Joint torques of the manipulator

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
getJointTorqueSensors(self: pyaubo_sdk.RobotState) -> List[float]
```

Lua function prototype

```
getJointTorqueSensors() -> table
```

Lua example

```
JointTorqueSensors = getJointTorqueSensors()
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.RobotState.getJointTorqueSensors","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":[0.0,6275.367736816406,-7704.2816162109375,3586.9766235351563,503.0364990234375,1506.0882568359375]}
```

8.6.9.2.25 getJointUniqueIds()

```
std::vector< std::string > arcs::common_interface::RobotState::getJointUniqueIds ()
```

Get the globally unique IDs of the joints

Returns

Globally unique IDs of the joints

Exceptions

arcs::common_interface::AuboException	
---------------------------------------	--

Python function prototype

```
getJointUniqueIds(self: pyaubo_sdk.RobotState) -> List[str]
```

Lua function prototype

```
getJointUniqueIds() -> table
```

Lua example

```
JointUniqueIds = getJointUniqueIds()
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.RobotState.getJointUniqueIds","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":["00800020ffffff31405153","00800020ffffff3e3f5153","00800020ffffff414b5153","00800020ffffff414b5153"]}
```

8.6.9.2.26 getJointVoltages()

```
std::vector< double > arcs::common_interface::RobotState::getJointVoltages ()
```

Get the joint voltages of the manipulator

Returns

Joint voltages of the manipulator

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
getJointVoltages(self: pyaubo_sdk.RobotState) -> List[float]
```

Lua function prototype

```
getJointVoltages\(\) -> table
```

Lua example

```
JointVoltages = getJointVoltages\(\)
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.RobotState.getJointVoltages","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":[2.0,2.5,3.0,2.0,2.5,2.0]}
```

8.6.9.2.27 getMainCurrent()

```
double arcs::common_interface::RobotState::getMainCurrent ()
```

Get the main bus current

Returns

Main bus current

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
getMainCurrent(self: pyaubo_sdk.RobotState) -> float
```

Lua function prototype

```
getMainCurrent() -> number
```

Lua example

```
MainCurrent = getMainCurrent()
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.RobotState.getMainCurrent","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":0.3204345703125}
```

8.6.9.2.28 getMainVoltage()

```
double arcs::common_interface::RobotState::getMainVoltage ()
```

Get the main bus voltage

Returns

Main bus voltage

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
getMainVoltage(self: pyaubo_sdk.RobotState) -> float
```

Lua function prototype

```
getMainVoltage() -> number
```

Lua example

```
MainVoltage = getMainVoltage()
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.RobotState.getMainVoltage","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":52.75}
```

8.6.9.2.29 getMasterBoardFirmwareVersion()

```
int arcs::common_interface::RobotState::getMasterBoardFirmwareVersion ()
```

Get the MasterBoard firmware version

Returns

MasterBoard firmware version

Exceptions

arcs::common_interface::AuboException	
---------------------------------------	--

Python function prototype

```
getMasterBoardFirmwareVersion(self: pyaubo_sdk.RobotState) -> int
```

Lua function prototype

```
getMasterBoardFirmwareVersion() -> number
```

Lua example

```
MasterBoardFirmwareVersion = getMasterBoardFirmwareVersion()
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.RobotState.getMasterBoardFirmwareVersion","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":1000004}
```

8.6.9.2.30 `getMasterBoardHardwareVersion()`

```
int arcs::common_interface::RobotState::getMasterBoardHardwareVersion ()
```

Get the MasterBoard hardware version

Returns

MasterBoard hardware version

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
getMasterBoardHardwareVersion(self: pyaubo_sdk.RobotState) -> int
```

Lua function prototype

```
getMasterBoardHardwareVersion\(\) -> number
```

Lua example

```
MasterBoardHardwareVersion = getMasterBoardHardwareVersion\(\)
```

JSON-RPC request example

```
{ "jsonrpc": "2.0", "method": "rob1.RobotState.getMasterBoardHardwareVersion", "params": [], "id": 1 }
```

JSON-RPC response example

```
{ "id": 1, "jsonrpc": "2.0", "result": 1000000 }
```

8.6.9.2.31 `getMasterBoardUniqueId()`

```
std::string arcs::common_interface::RobotState::getMasterBoardUniqueId ()
```

Get the globally unique ID of the MasterBoard

Returns

Globally unique ID of the MasterBoard

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
getMasterBoardUniqueId(self: pyaubo_sdk.RobotState) -> str
```

Lua function prototype

```
getMasterBoardUniqueId() -> string
```

Lua example

```
MasterBoardUniqueId = getMasterBoardUniqueId()
```

JSON-RPC request example

```
{"jsonrpc": "2.0", "method": "rob1.RobotState.getMasterBoardUniqueId", "params": [], "id": 1}
```

JSON-RPC response example

```
{"id": 1, "jsonrpc": "2.0", "result": "001e0044510f343037323637"}
```

8.6.9.2.32 getPedestalFirmwareVersion()

```
int arcs::common_interface::RobotState::getPedestalFirmwareVersion ()
```

Get the pedestal firmware version

Returns

Pedestal firmware version

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
getPedestalFirmwareVersion(self: pyaubo_sdk.RobotState) -> int
```

Lua function prototype

```
getPedestalFirmwareVersion() -> number
```

Lua example

```
PedestalFirmwareVersion = getPedestalFirmwareVersion()
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.RobotState.getPedestalFirmwareVersion","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":1000004}
```

8.6.9.2.33 getPedestalHardwareVersion()

```
int arcs::common_interface::RobotState::getPedestalHardwareVersion ()
```

Get the pedestal hardware version

Returns

Pedestal hardware version

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
getPedestalHardwareVersion(self: pyaubo_sdk.RobotState) -> int
```

Lua function prototype

```
getPedestalHardwareVersion() -> number
```

Lua example

```
PedestalHardwareVersion = getPedestalHardwareVersion()
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.RobotState.getPedestalHardwareVersion","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":1007000}
```

8.6.9.2.34 getPedestalUniqueId()

```
std::string arcs::common_interface::RobotState::getPedestalUniqueId ()
```

Get the globally unique ID of the pedestal

Returns

Globally unique ID of the pedestal

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
getPedestalUniqueId(self: pyaubo_sdk.RobotState) -> str
```

Lua function prototype

```
getPedestalUniqueId\(\) -> string
```

Lua example

```
PedestalUniqueId = getPedestalUniqueId\(\)
```

JSON-RPC request example

```
{"jsonrpc": "2.0", "method": "rob1.RobotState.getPedestalUniqueId", "params": [], "id": 1}
```

JSON-RPC response example

```
{"id": 1, "jsonrpc": "2.0", "result": "205257533543065248544339"}
```

8.6.9.2.35 getRobotCurrent()

```
double arcs::common_interface::RobotState::getRobotCurrent ()
```

Get the robot current

Returns

Robot current

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
getRobotCurrent(self: pyaubo_sdk.RobotState) -> float
```

Lua function prototype

```
getRobotCurrent() -> number
```

Lua example

```
RobotCurrent = getRobotCurrent()
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.RobotState.getRobotCurrent","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":0.3204345703125}
```

8.6.9.2.36 getRobotModeType()

[RobotModeType](#) [arcs::common_interface::RobotState::getRobotModeType](#) ()

Get the robot mode state

Returns

The robot's mode state

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
getRobotModeType(self: pyaubo_sdk.RobotState) -> arcs::common\_interface::RobotModeType
```

Lua function prototype

```
getRobotModeType() -> number
```

Lua example

```
RobotModeType = getRobotModeType()
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.RobotState.getRobotModeType","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":"Running"}
```

8.6.9.2.37 getRobotVoltage()

double arcs::common_interface::RobotState::getRobotVoltage ()

Get the robot voltage

Returns

Robot voltage

Exceptions

arcs::common_interface::AuboException	
---------------------------------------	--

Python function prototype

```
getRobotVoltage(self: pyaubo_sdk.RobotState) -> float
```

Lua function prototype

```
getRobotVoltage() -> number
```

Lua example

```
RobotVoltage = getRobotVoltage()
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.RobotState.getRobotVoltage","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":52.75}
```

8.6.9.2.38 `getSafetyModeType()`

`SafetyModeType` `arcs::common_interface::RobotState::getSafetyModeType ()`

Get the safety mode

Returns

The safety mode

Exceptions

<code>arcs::common_interface::AuboException</code>	
--	--

Python function prototype

`getSafetyModeType(self: pyaubo_sdk.RobotState) -> arcs::common_interface::SafetyModeType`

Lua function prototype

`getSafetyModeType() -> number`

Lua example

`SafetyModeType = getSafetyModeType()`

JSON-RPC request example

```
{"jsonrpc": "2.0", "method": "rob1.RobotState.getSafetyModeType", "params": [], "id": 1}
```

JSON-RPC response example

```
{"id": 1, "jsonrpc": "2.0", "result": "Normal"}
```

8.6.9.2.39 `getSlaveBoardFirmwareVersion()`

`int arcs::common_interface::RobotState::getSlaveBoardFirmwareVersion ()`

Get the SlaveBoard firmware version

Returns

SlaveBoard firmware version

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
getSlaveBoardFirmwareVersion(self: pyaubo_sdk.RobotState) -> int
```

Lua function prototype

```
getSlaveBoardFirmwareVersion() -> number
```

Lua example

```
SlaveBoardFirmwareVersion = getSlaveBoardFirmwareVersion()
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.RobotState.getSlaveBoardFirmwareVersion","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":0}
```

8.6.9.2.40 getSlaveBoardHardwareVersion()

```
int arcs::common_interface::RobotState::getSlaveBoardHardwareVersion ()
```

Get the SlaveBoard hardware version

Returns

SlaveBoard hardware version

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
getSlaveBoardHardwareVersion(self: pyaubo_sdk.RobotState) -> int
```

Lua function prototype

```
getSlaveBoardHardwareVersion() -> number
```

Lua example

```
SlaveBoardHardwareVersion = getSlaveBoardHardwareVersion()
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.RobotState.getSlaveBoardHardwareVersion","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":6030098}
```

8.6.9.2.41 getSlaveBoardUniqueId()

std::string arcs::common_interface::RobotState::getSlaveBoardUniqueId ()

Get the globally unique ID of the SlaveBoard

Returns

Globally unique ID of the SlaveBoard

Exceptions

arcs::common_interface::AuboException	
---------------------------------------	--

Python function prototype

```
getSlaveBoardUniqueId(self: pyaubo_sdk.RobotState) -> str
```

Lua function prototype

```
getSlaveBoardUniqueId() -> string
```

Lua example

```
SlaveBoardUniqueId = getSlaveBoardUniqueId()
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.RobotState.getSlaveBoardUniqueId","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":"73657263008000000000000000000000"}
```


8.6.9.2.42 getSlowDownLevel()

```
int arcs::common_interface::RobotState::getSlowDownLevel ()
```

Get the robot slow down level

Returns

Robot slow down level

Exceptions

arcs::common_interface::AuboException	
---	--

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.RobotState.getSlowDownLevel","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":0}
```

8.6.9.2.43 getTargetTcpPose()

```
std::vector< double > arcs::common_interface::RobotState::getTargetTcpPose ()
```

Get the next target waypoint.

Note the difference from getTcpTargetPose; the definition here is ambiguous and the naming needs improvement.

The pose is represented as (x, y, z, rx, ry, rz). x, y, z are the target position of the Tool Center Point (TCP) in the base coordinate system, in meters. rx, ry, rz are the target orientation of the TCP in the base coordinate system, as ZYX Euler angles in radians.

Returns

The current target pose, in the form (x, y, z, rx, ry, rz)

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
getTargetTcpPose(self: pyaubo_sdk.RobotState) -> List[float]
```

Lua function prototype

```
getTargetTcpPose() -> table
```

Lua example

```
TargetTcpPose = getTargetTcpPose()
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.RobotState.getTargetTcpPose","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":[0.4173932217619493,-0.13250000000000012,0.43296496133045825,3.141577313781914,0.0,1.5707963267948963]}
```

8.6.9.2.44 getTcpForce()

```
std::vector< double > arcs::common_interface::RobotState::getTcpForce ()
```

Get the TCP force/torque

Returns

TCP force/torque

Python function prototype

```
getTcpForce(self: pyaubo_sdk.RobotState) -> List[float]
```

Lua function prototype

```
getTcpForce() -> table
```

Lua example

```
TcpForce = getTcpForce()
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.RobotState.getTcpForce","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":[0.0,0.0,0.0,0.0,0.0,0.0]}
```

Exceptions

<code>arcs::common_interface::AuboException</code>	
--	--

Python function prototype

```
getTcpForceSensors(self: pyaubo_sdk.RobotState) -> List[float]
```

Lua function prototype

```
getTcpForceSensors() -> table
```

Lua example

```
TcpForceSensors = getTcpForceSensors()
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.RobotState.getTcpForceSensors","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":[0.0,0.0,0.0,0.0,0.0,0.0]}
```

8.6.9.2.46 getTcpForceSensorStatus()

```
bool arcs::common_interface::RobotState::getTcpForceSensorStatus (
    const std::string & name)
```

Get the communication status of the tool force sensor

Parameters

name	force sensor name, it is consistent with the parameters of 'selectTcpForceSensor'
------	---

Returns

Returns true if communication is normal; otherwise returns false

Exceptions

arcs::common_interface::AuboException	
---	--

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.RobotState.getSlowDownLevel","params":["tool.KWR75A"],
"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":0}
```

8.6.9.2.47 getTcpPose()

```
std::vector< double > arcs::common_interface::RobotState::getTcpPose ()
```

Get the current TCP pose.

The TCP offset can be obtained by getActualTcpOffset.

The pose is represented as (x, y, z, rx, ry, rz). x, y, z are the position of the Tool Center Point (TCP) in the base coordinate system, in meters. rx, ry, rz are the orientation of the TCP in the base coordinate system, as ZYX Euler angles in radians.

Returns

The TCP pose, in the form (x, y, z, rx, ry, rz)

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
getTcpPose(self: pyaubo_sdk.RobotState) -> List[float]
```

Lua function prototype

```
getTcpPose\(\) -> table
```

Lua example

```
TcpPose = getTcpPose\(\)
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.RobotState.getTcpPose","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":[0.41777839846910425,-0.132500000000000012,0.20928451364415995,3.1415792312578987,0.0,1.5707963267948963]}
```

8.6.9.2.48 getTcpSpeed()

```
std::vector< double > arcs::common_interface::RobotState::getTcpSpeed ()
```

Get the TCP speed

Returns

TCP speed

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
getTcpSpeed(self: pyaubo_sdk.RobotState) -> List[float]
```

Lua function prototype

```
getTcpSpeed() -> table
```

Lua example

```
TcpSpeed = getTcpSpeed()
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.RobotState.getTcpSpeed","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":[0.0,0.0,0.0,0.0,0.0,0.0]}
```

8.6.9.2.49 getTcpTargetForce()

```
std::vector< double > arcs::common_interface::RobotState::getTcpTargetForce ()
```

Get the TCP target force/torque

Returns

TCP target force/torque

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
getTcpTargetForce(self: pyaubo_sdk.RobotState) -> List[float]
```

Lua function prototype

```
getTcpTargetForce() -> table
```

Lua example

```
TcpTargetForce = getTcpTargetForce()
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.RobotState.getTcpTargetForce","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":[0.0,0.0,0.0,0.0,0.0,0.0]}
```

8.6.9.2.50 getTcpTargetPose()

```
std::vector< double > arcs::common_interface::RobotState::getTcpTargetPose ()
```

Get the last sent TCP target pose

Returns

TCP target pose

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
getTcpTargetPose(self: pyaubo_sdk.RobotState) -> List[float]
```

Lua function prototype

`getTcpTargetPose()` -> table

Lua example

`TcpTargetPose = getTcpTargetPose()`

JSON-RPC request example

```
{ "jsonrpc": "2.0", "method": "rob1.RobotState.getTcpTargetPose", "params": [], "id": 1 }
```

JSON-RPC response example

```
{ "id": 1, "jsonrpc": "2.0", "result": [0.41777829240862013, -0.132500000000000012, 0.2092832117232601, 3.1415812372223217, 0.0, 1.5707963267948963] }
```

8.6.9.2.51 getTcpTargetSpeed()

`std::vector< double > arcs::common_interface::RobotState::getTcpTargetSpeed ()`

Get the TCP target speed

Returns

TCP target speed

Exceptions

<code>arcs::common_interface::AuboException</code>	
--	--

Python function prototype

`getTcpTargetSpeed(self: pyaubo_sdk.RobotState) -> List[float]`

Lua function prototype

`getTcpTargetSpeed()` -> table

Lua example

`TcpTargetSpeed = getTcpTargetSpeed()`

JSON-RPC request example

```
{ "jsonrpc": "2.0", "method": "rob1.RobotState.getTcpTargetSpeed", "params": [], "id": 1 }
```

JSON-RPC response example

```
{ "id": 1, "jsonrpc": "2.0", "result": [0.0, 0.0, 0.0, 0.0, 0.0, 0.0] }
```

8.6.9.2.52 `getToolCommMode()`

```
int arcs::common_interface::RobotState::getToolCommMode ()
```

Get the tool communication mode

Returns

Tool communication mode 0: No serial port 1: [Serial](#) port only 2: With force sensor and serial port

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
getToolCommMode(self: pyaubo_sdk.RobotState) -> int
```

Lua function prototype

```
getToolCommMode\(\) -> number
```

Lua example

```
ToolCommMode = getToolCommMode\(\)
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.RobotState.getToolCommMode","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":1}
```

8.6.9.2.53 `getToolFirmwareVersion()`

```
int arcs::common_interface::RobotState::getToolFirmwareVersion ()
```

Get the tool firmware version

Returns

Tool firmware version

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
getToolFirmwareVersion(self: pyaubo_sdk.RobotState) -> int
```

Lua function prototype

```
getToolFirmwareVersion() -> number
```

Lua example

```
ToolFirmwareVersion = getToolFirmwareVersion()
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.RobotState.getToolFirmwareVersion","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":1001003}
```

8.6.9.2.54 getToolHardwareVersion()

```
int arcs::common_interface::RobotState::getToolHardwareVersion ()
```

Get the tool hardware version

Returns

```
Tool hardware version
```

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
getToolHardwareVersion(self: pyaubo_sdk.RobotState) -> int
```

Lua function prototype

```
getToolHardwareVersion() -> number
```

Lua example

```
ToolHardwareVersion = getToolHardwareVersion()
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.RobotState.getToolHardwareVersion","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":1000000}
```

8.6.9.2.55 getToolPose()

```
std::vector< double > arcs::common_interface::RobotState::getToolPose ()
```

Get the tool pose (without TCP offset)

The pose is represented as (x, y, z, rx, ry, rz). x, y, z are the target position of the flange center in the base coordinate system, in meters. rx, ry, rz are the target orientation of the flange center in the base coordinate system, as ZYX Euler angles in radians.

Returns

The tool pose, in the form (x, y, z, rx, ry, rz)

Exceptions

arcs::common_interface::AuboException	
---------------------------------------	--

Python function prototype

```
getToolPose(self: pyaubo_sdk.RobotState) -> List[float]
```

Lua function prototype

```
getToolPose() -> table
```

Lua example

```
ToolPose = getToolPose()
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.RobotState.getToolPose","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":[0.41777820858878617,-0.132500000000000012,0.20928410288421018,3.141579231257899,0.0,1.5707963267948963]}
```

8.6.9.2.56 getToolUniqueId()

```
std::string arcs::common_interface::RobotState::getToolUniqueId ()
```

Get the globally unique ID of the tool

Returns

Globally unique ID of the tool

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
getToolUniqueId(self: pyaubo_sdk.RobotState) -> str
```

Lua function prototype

```
getToolUniqueId\(\) -> string
```

Lua example

```
ToolUniqueId = getToolUniqueId\(\)
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.RobotState.getToolUniqueId","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":"397d4e5331541252314d3042"}
```

8.6.9.2.57 isCollisionOccurred()

```
bool arcs::common_interface::RobotState::isCollisionOccurred ()
```

Whether a collision has occurred

Returns

Returns true if a collision occurred; otherwise returns false

Exceptions

arcs::common_interface::AuboException	
---	--

Lua function prototype

```
isCollisionOccurred() -> boolean
```

Lua example

```
CollisionOccurred = isCollisionOccurred()
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.RobotState.isCollisionOccurred","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":false}
```

8.6.9.2.58 isPowerOn()

```
bool arcs::common_interface::RobotState::isPowerOn ()
```

Get the robot power-on state

Returns

Returns true if the robot is powered on; otherwise returns false

Exceptions

arcs::common_interface::AuboException	
---	--

Lua function prototype

```
isPowerOn() -> boolean
```

Lua example

```
PowerOn = isPowerOn()
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.RobotState.isPowerOn","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":true}
```

8.6.9.2.59 isSteady()

```
bool arcs::common_interface::RobotState::isSteady ()
```

Whether the robot has stopped

Returns

Returns true if stopped; otherwise returns false

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
isSteady(self: pyaubo_sdk.RobotState) -> bool
```

Lua function prototype

```
isSteady() -> boolean
```

Lua exaple

```
Steady = isSteady()
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.RobotState.isSteady","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":true}
```

8.6.9.2.60 isTeachPendantEnabled()

```
bool arcs::common_interface::RobotState::isTeachPendantEnabled ()
```

Get whether the teach pendant is enabled.

Indicates whether the enable button of teach pendant is in the pressed state.

Returns

Returns false if the enable button of teach pendant is pressed; otherwise returns true

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
isTeachPendantEnabled(self: pyaubo_sdk.RobotState) -> bool
```

Lua function prototype

```
isTeachPendantEnabled() -> boolean
```

Lua example

```
TeachPendantEnabled = isTeachPendantEnabled()
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.RobotState.isTeachPendantEnabled","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":true}
```

8.6.9.2.61 isToolFlangeEnabled()

```
bool arcs::common_interface::RobotState::isToolFlangeEnabled ()
```

Get whether the tool flange is enabled.

Returns

Returns false if the tool flange is disabled; otherwise returns true

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
isToolFlangeEnabled(self: pyaubo_sdk.RobotState) -> bool
```

Lua function prototype

```
isToolFlangeEnabled() -> boolean
```

Lua example

```
toolEnabled = isToolFlangeEnabled()
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.RobotState.isToolFlangeEnabled","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":true}
```

8.6.9.2.62 isWithinSafetyLimits()

bool arcs::common_interface::RobotState::isWithinSafetyLimits ()

Whether the robot is within safety limits

Returns

Returns true if within safety limits; otherwise returns false

Exceptions

arcs::common_interface::AuboException	
---------------------------------------	--

Python function prototype

```
isWithinSafetyLimits(self: pyaubo_sdk.RobotState) -> bool
```

Lua function prototype

```
isWithinSafetyLimits() -> boolean
```

Lua example

```
WithinSafetyLimits = isWithinSafetyLimits()
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.RobotState.isWithinSafetyLimits","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":true}
```

8.7 RuntimeMachine ()

The [RuntimeMachine](#) class.

Functions

- int [arcs::common_interface::RuntimeMachine::newTask](#) (bool daemon=false)
Returns the task_id
- int [arcs::common_interface::RuntimeMachine::deleteTask](#) (int tid)
Delete a task, which will terminate any ongoing motion.
- int [arcs::common_interface::RuntimeMachine::detachTask](#) (int tid)
Wait for the task to finish naturally
- bool [arcs::common_interface::RuntimeMachine::isTaskAlive](#) (int tid)
Check if the task is alive
- int [arcs::common_interface::RuntimeMachine::getTaskQueueSize](#) (int tid)
Get the number of cached instructions in the task
- int [arcs::common_interface::RuntimeMachine::switchTask](#) (int tid)
Switch the current thread.
- int [arcs::common_interface::RuntimeMachine::setLabel](#) (int lineno, const std::string &comment)
Mark the line number and comment of the recorded instruction
- ARCS_DEPRECATED int [arcs::common_interface::RuntimeMachine::setPlanContext](#) (int tid, int lineno, const std::string &comment)
Add a comment to the aubo_control log Use setLabel instead
- int [arcs::common_interface::RuntimeMachine::nop](#) ()
No operation
- std::tuple< std::string, std::string > [arcs::common_interface::RuntimeMachine::getExecutionStatus](#) ()
Get the execution status of time-consuming interfaces (INST), such as setPersistentParameters
- int [arcs::common_interface::RuntimeMachine::gotoLine](#) (int lineno)
Jump to the specified line number
- std::tuple< int, int, std::string > [arcs::common_interface::RuntimeMachine::getPlanContext](#) (int tid=-1)
Get the current runtime context
- std::tuple< int, int, std::string > [arcs::common_interface::RuntimeMachine::getAdvancePlanContext](#) (int tid=-1)
Get the context information of the advance planner
- int [arcs::common_interface::RuntimeMachine::getAdvancePtr](#) (int tid=-1)
Get the program pointer of AdvanceRun
- int [arcs::common_interface::RuntimeMachine::getMainPtr](#) (int tid=-1)
Get the program pointer of robot motion
- int [arcs::common_interface::RuntimeMachine::getInterpPtr](#) (int tid)
Get the pointer of the most recently interpreted instruction
- int [arcs::common_interface::RuntimeMachine::loadProgram](#) (const std::string &program)
Load a local project file.
- int [arcs::common_interface::RuntimeMachine::preloadProgram](#) (int index, const std::string &program)
Preload project file
- std::string [arcs::common_interface::RuntimeMachine::getPreloadProgram](#) (int index)
Get the name of the preloaded project file.
- int [arcs::common_interface::RuntimeMachine::clearPreloadPrograms](#) ()

- Clear all preloaded project files.
- `int arcs::common_interface::RuntimeMachine::runProgram ()`
Run the already loaded project file
- `int arcs::common_interface::RuntimeMachine::start ()`
Start the runtime
- `int arcs::common_interface::RuntimeMachine::stop ()`
Stop the runtime, i.e., stop script execution.
- `int arcs::common_interface::RuntimeMachine::abort ()`
Abort robot operation.
- `int arcs::common_interface::RuntimeMachine::pause ()`
Pause the interpreter
- `int arcs::common_interface::RuntimeMachine::step ()`
Execute a single step
- `int arcs::common_interface::RuntimeMachine::resume ()`
Resume the interpreter
- `int arcs::common_interface::RuntimeMachine::arbitraryResume ()`
Resume the interpreter
- `int arcs::common_interface::RuntimeMachine::setResumeWait (bool wait)`
Wait for the previous sequence to complete before resuming the interpreter
- `int arcs::common_interface::RuntimeMachine::enterCritical (double timeout)`
The abort command is deferred during critical sections to avoid interrupting internal instructions.
- `int arcs::common_interface::RuntimeMachine::exitCritical ()`
Exit the critical section
- `ARCS_DEPRECATED RuntimeState arcs::common_interface::RuntimeMachine::getStatus ()`
Get the status of the planner
- `int arcs::common_interface::RuntimeMachine::setBreakPoint (int lineno)`
Set a breakpoint
- `int arcs::common_interface::RuntimeMachine::removeBreakPoint (int lineno)`
Remove a breakpoint
- `int arcs::common_interface::RuntimeMachine::clearBreakPoints ()`
Clear all breakpoints
- `int arcs::common_interface::RuntimeMachine::timerStart (const std::string &name)`
Start the timer
- `int arcs::common_interface::RuntimeMachine::timerStop (const std::string &name)`
Stop the timer
- `int arcs::common_interface::RuntimeMachine::timerReset (const std::string &name)`
Reset the timer
- `int arcs::common_interface::RuntimeMachine::timerDelete (const std::string &name)`
Delete the timer
- `double arcs::common_interface::RuntimeMachine::getTimer (const std::string &name)`
Get the timer value
- `int arcs::common_interface::RuntimeMachine::triggBegin (double distance, double delay)`
Start configuring trigger
- `int arcs::common_interface::RuntimeMachine::triggEnd ()`
End configuring trigger
- `int arcs::common_interface::RuntimeMachine::triggInterrupt (double distance, double delay)`
Returns the automatically assigned interrupt number
- `std::vector< int > arcs::common_interface::RuntimeMachine::getTriggInterrupts ()`
Get the list of all interrupt numbers

8.7.1 Detailed Description

The [RuntimeMachine](#) class.

8.7.2 Function Documentation

8.7.2.1 abort()

```
int arcs::common_interface::RuntimeMachine::abort ()
```

Abort robot operation.

If you only want to stop the runtime, you can call the [RuntimeMachine::stop](#) interface.

If the script runtime is in the Running state, aborts the runtime; if the runtime is Stopped and the robot is moving, stops the robot motion; if force control is enabled, stops force control.

Returns

Python function prototype

```
abort(self: pyaubo_sdk.RuntimeMachine) -> int
```

Lua function prototype

```
abort\(\) -> number
```

JSON-RPC Request Example

```
{"jsonrpc":"2.0","method":"RuntimeMachine.abort","params":[],"id":1}
```

JSON-RPC Response Example

```
{"id":1,"jsonrpc":"2.0","result":0}
```

8.7.2.2 arbitraryResume()

```
int arcs::common_interface::RuntimeMachine::arbitraryResume ()
```

Resume the interpreter

Returns

Python function prototype

```
arbitraryResume(self: pyaubo_sdk.RuntimeMachine) -> int
```

Lua function prototype

```
arbitraryResume\(\) -> number
```

JSON-RPC Request Example

```
{"jsonrpc":"2.0","method":"RuntimeMachine.arbitraryResume","params":[],"id":1}
```

JSON-RPC Response Example

```
{"id":1,"jsonrpc":"2.0","result":0}
```

8.7.2.3 clearBreakPoints()

```
int arcs::common_interface::RuntimeMachine::clearBreakPoints ()
```

Clear all breakpoints

Returns

Python function prototype

```
clearBreakPoints(self: pyaubo_sdk.RuntimeMachine) -> int
```

Lua function prototype

```
clearBreakPoints() -> number
```

JSON-RPC Request Example

```
{"jsonrpc":"2.0","method":"RuntimeMachine.clearBreakPoints","params":[],"id":1}
```

JSON-RPC Response Example

```
{"id":1,"jsonrpc":"2.0","result":0}
```

8.7.2.4 clearPreloadPrograms()

```
int arcs::common_interface::RuntimeMachine::clearPreloadPrograms ()
```

Clear all preloaded project files.

Calling this method will release all project indices and their associated program names that were previously preloaded via preloadProgram.

Returns

Returns 0 on success;

8.7.2.5 deleteTask()

```
int arcs::common_interface::RuntimeMachine::deleteTask (
    int tid)
```

Delete a task, which will terminate any ongoing motion.

JSON-RPC Request Example

```
Generated by Doxygen {"jsonrpc":"2.0","method":"RuntimeMachine.deleteTask","params":[26],"id":1}
```

JSON-RPC Response Example

Parameters

tid	
-----	--

Returns

JSON-RPC Request Example

```
{ "jsonrpc": "2.0", "method": "RuntimeMachine.detachTask", "params": [26], "id": 1 }
```

JSON-RPC Response Example

```
{ "id": 1, "jsonrpc": "2.0", "result": 0 }
```

8.7.2.7 enterCritical()

```
int arcs::common_interface::RuntimeMachine::enterCritical (
    double timeout)
```

The abort command is deferred during critical sections to avoid interrupting internal instructions.

Parameters

timeout	(seconds, 0~5): max deferral time for abort. Exceeding it forces exit from critical section and triggers abort.
---------	---

Returns

Python function prototype

```
enterCritical(self: pyaubo_sdk.RuntimeMachine, arg0: double) -> int
```

Lua function prototype

```
enterCritical(timeout: number) -> number
```

JSON-RPC Request Example

```
{ "jsonrpc": "2.0", "method": "RuntimeMachine.enterCritical", "params": [5.0], "id": 1 }
```

Generated by Doxygen

JSON-RPC Response Example

Parameters

--	--

return

Python function prototype

```
exitCritical(self: pyaubo_sdk.RuntimeMachine) -> int
```

Lua function prototype

```
exitCritical() -> number
```

JSON-RPC Request Example

```
{"jsonrpc":"2.0","method":"RuntimeMachine.exitCritical","params":[],"id":1}
```

JSON-RPC Response Example

```
{"id":1,"jsonrpc":"2.0","result":0}
```

8.7.2.9 getAdvancePlanContext()

```
std::tuple< int, int, std::string > arcs::common_interface::RuntimeMachine::getAdvancePlanContext (
    int tid = -1)
```

Get the context information of the advance planner

Parameters

tid	Task ID If specified (not -1), returns the context information of the advance planner for the corresponding task; if not specified (is -1), returns the context information of the advance planner for the currently running thread
-----	---

Returns

JSON-RPC Request Example

```
{"jsonrpc":"2.0","method":"RuntimeMachine.getAdvancePlanContext","params":[-1],"id":1}
```

JSON-RPC Response Example

```
{"id":1,"jsonrpc":"2.0","result":[-1,-1,""]}
```

8.7.2.10 getAdvancePtr()

```
int arcs::common_interface::RuntimeMachine::getAdvancePtr (
    int tid = -1)
```

Get the program pointer of AdvanceRun

Returns

JSON-RPC Request Example

```
{"jsonrpc":"2.0","method":"RuntimeMachine.getAdvancePtr","params":[-1],"id":1}
```

JSON-RPC Response Example

```
{"id":1,"jsonrpc":"2.0","result":-1}
```

8.7.2.11 getExecutionStatus()

```
std::tuple< std::string, std::string > arcs::common_interface::RuntimeMachine::getExecutionStatus ()
```

Get the execution status of time-consuming interfaces (INST), such as setPersistentParameters

Returns

Instruction name, execution status Execution status: EXECUTING/FINISHED

Python function prototype

```
getExecutionStatus(self: pyaubo_sdk.RuntimeMachine) -> Tuple[str, str, int]
```

Lua function prototype

```
getExecutionStatus\(\) -> string, string, number
```

JSON-RPC Request Example

```
{"jsonrpc":"2.0","method":"RuntimeMachine.getExecutionStatus","params":[],"id":1}
```

JSON-RPC Response Example

```
{"id":1,"jsonrpc":"2.0","result":["confirmSafetyParameters","FINISHED"]}
```

8.7.2.12 getInterpPtr()

```
int arcs::common_interface::RuntimeMachine::getInterpPtr (
    int tid)
```

Parameters

tid	
-----	--

Returns

JSON-RPC Request Example

```
{"jsonrpc":"2.0","method":"RuntimeMachine.getInterpPtr","params":[26],"id":1}
```

JSON-RPC Response Example

```
{"id":1,"jsonrpc":"2.0","result":-1}
```

8.7.2.13 getMainPtr()

```
int arcs::common_interface::RuntimeMachine::getMainPtr (
    int tid = -1)
```

Get the program pointer of robot motion

Parameters

tid	Task ID If specified (not -1), returns the program pointer of the corresponding task; if not specified (is -1), returns the program pointer of the currently running thread
-----	---

Returns

JSON-RPC Request Example

```
{"jsonrpc":"2.0","method":"RuntimeMachine.getMainPtr","params":[-1],"id":1}
```

JSON-RPC Response Example

```
{"id":1,"jsonrpc":"2.0","result":-1}
```

8.7.2.14 getPlanContext()

Generated by Doxygen

```
std::tuple< int, int, std::string > arcs::common_interface::RuntimeMachine::getPlanContext (
    int tid = -1)
```

Get the current runtime context

Parameters

tid	Task ID If specified (not -1), returns the runtime context of the corresponding task; if not specified (is -1), returns the runtime context of the currently running thread
-----	---

Returns

Python function prototype

```
getPlanContext(self: pyaubo_sdk.RuntimeMachine) -> Tuple[int, int, str]
```

Lua function prototype

```
getPlanContext() -> number
```

JSON-RPC Request Example

```
{"jsonrpc":"2.0","method":"RuntimeMachine.getPlanContext","params":[-1],"id":1}
```

JSON-RPC Response Example

```
{"id":1,"jsonrpc":"2.0","result":[-1,0,""]}
```

8.7.2.15 getPreloadProgram()

```
std::string arcs::common_interface::RuntimeMachine::getPreloadProgram (
    int index)
```

Get the name of the preloaded project file.

Returns an empty string if not loaded or index is out of range.

Parameters

index	Project index number (0~99)
-------	-----------------------------

Returns

Project file name

8.7.2.16 getStatus()

ARCS_DEPRECATED [RuntimeState](#) arcs::common_interface::RuntimeMachine::getStatus ()

Get the status of the planner

Returns

Python function prototype

```
getStatus(self: pyaubo_sdk.RuntimeMachine) -> arcs::common\_interface::RuntimeState
```

Lua function prototype

```
getStatus\(\) -> number
```

JSON-RPC Request Example

```
{ "jsonrpc": "2.0", "method": "RuntimeMachine.getStatus", "params": [], "id": 1 }
```

JSON-RPC Response Example

```
{ "id": 1, "jsonrpc": "2.0", "result": "Running" }
```

8.7.2.17 getTaskQueueSize()

```
int arcs::common_interface::RuntimeMachine::getTaskQueueSize (
    int tid)
```

Get the number of cached instructions in the task

Parameters

tid	
-----	--

Returns

8.7.2.18 getTimer()

```
double arcs::common_interface::RuntimeMachine::getTimer (
    Generated by Doxygen
    const std::string & name)
```

Get the timer value

Parameters

name	
------	--

Returns

Python function prototype

```
getTimer(self: pyaubo_sdk.RuntimeMachine, arg0: str) -> float
```

Lua function prototype

```
getTimer(name: string) -> number
```

JSON-RPC Request Example

```
{"jsonrpc":"2.0","method":"RuntimeMachine.getTimer","params":["timer"],"id":1}
```

JSON-RPC Response Example

```
{"id":1,"jsonrpc":"2.0","result":25.409769612}
```

8.7.2.19 getTriggInterrupts()

```
std::vector< int > arcs::common_interface::RuntimeMachine::getTriggInterrupts ()
```

Get the list of all interrupt numbers

Returns

JSON-RPC Request Example

```
{"jsonrpc":"2.0","method":"RuntimeMachine.getTriggInterrupts","params":[],"id":1}
```

JSON-RPC Response Example

```
{"id":1,"jsonrpc":"2.0","result":[]}
```

8.7.2.20 gotoLine()

```
int arcs::common_interface::RuntimeMachine::gotoLine (
    int lineno)
```

Jump to the specified line number

Parameters

lineno	
--------	--

Returns

Python function prototype

```
gotoLine(self: pyaubo_sdk.RuntimeMachine, arg0: int) -> int
```

Lua function prototype

```
gotoLine(lineno: number) -> number
```

JSON-RPC Request Example

```
{"jsonrpc": "2.0", "method": "RuntimeMachine.gotoLine", "params": [10], "id": 1}
```

JSON-RPC Response Example

```
{"id": 1, "jsonrpc": "2.0", "result": 0}
```

8.7.2.21 isTaskAlive()

```
bool arcs::common_interface::RuntimeMachine::isTaskAlive (  
    int tid)
```

Check if the task is alive

Parameters

tid	
-----	--

Returns

JSON-RPC Request Example

```
{"jsonrpc": "2.0", "method": "RuntimeMachine.isTaskAlive", "params": [26], "id": 1}
```

JSON-RPC Response Example

```
{"id": 1, "jsonrpc": "2.0", "result": true}
```

Parameters

program	
---------	--

Returns

JSON-RPC Request Example

```
{"jsonrpc": "2.0", "method": "RuntimeMachine.loadProgram", "params": ["demo"], "id": 1}
```

JSON-RPC Response Example

```
{"id": 1, "jsonrpc": "2.0", "result": 0}
```

8.7.2.23 newTask()

```
int arcs::common_interface::RuntimeMachine::newTask (  
    bool daemon = false)
```

Returns the task_id

JSON-RPC Request Example

```
{"jsonrpc": "2.0", "method": "RuntimeMachine.newTask", "params": [false], "id": 1}
```

JSON-RPC Response Example

```
{"id": 1, "jsonrpc": "2.0", "result": 26}
```

8.7.2.24 nop()

```
int arcs::common_interface::RuntimeMachine::nop ()
```

No operation

Returns

JSON-RPC Request Example

```
{"jsonrpc": "2.0", "method": "RuntimeMachine.nop", "params": [], "id": 1}
```

JSON-RPC Response Example

```
{"id": 1, "jsonrpc": "2.0", "result": 0}
```

8.7.2.25 pause()

```
int arcs::common_interface::RuntimeMachine::pause ()
```

Pause the interpreter

Returns

Python function prototype

```
pause(self: pyaubo_sdk.RuntimeMachine) -> int
```

Lua function prototype

```
pause() -> number
```

JSON-RPC Request Example

```
{"jsonrpc":"2.0","method":"RuntimeMachine.pause","params":[],"id":1}
```

JSON-RPC Response Example

```
{"id":1,"jsonrpc":"2.0","result":0}
```

8.7.2.26 preloadProgram()

```
int arcs::common_interface::RuntimeMachine::preloadProgram (
    int index,
    const std::string & program)
```

Preload project file

Parameters

index	Project index number (0~99)
program	Project name

Returns

8.7.2.27 removeBreakPoint()

Generated by Doxygen

```
int arcs::common_interface::RuntimeMachine::removeBreakPoint (
    int lineno)
```

Parameters

lineno	
--------	--

Returns

Python function prototype

```
removeBreakPoint(self: pyaubo_sdk.RuntimeMachine, arg0: int) -> int
```

Lua function prototype

```
removeBreakPoint(lineno: number) -> number
```

JSON-RPC Request Example

```
{"jsonrpc":"2.0","method":"RuntimeMachine.removeBreakPoint","params":[15],"id":1}
```

JSON-RPC Response Example

```
{"id":1,"jsonrpc":"2.0","result":0}
```

8.7.2.28 resume()

```
int arcs::common_interface::RuntimeMachine::resume ()
```

Resume the interpreter

Returns

Python function prototype

```
resume(self: pyaubo_sdk.RuntimeMachine) -> int
```

Lua function prototype

```
resume() -> number
```

JSON-RPC Request Example

```
{"jsonrpc":"2.0","method":"RuntimeMachine.resume","params":[],"id":1}
```

JSON-RPC Response Example

```
{"id":1,"jsonrpc":"2.0","result":0}
```

8.7.2.29 runProgram()

```
int arcs::common_interface::RuntimeMachine::runProgram ()
```

Run the already loaded project file

Returns

JSON-RPC Request Example

```
{"jsonrpc":"2.0","method":"RuntimeMachine.runProgram","params":[],"id":1}
```

JSON-RPC Response Example

```
{"id":1,"jsonrpc":"2.0","result":0}
```

8.7.2.30 setBreakPoint()

```
int arcs::common_interface::RuntimeMachine::setBreakPoint (  
    int lineno)
```

Set a breakpoint

Parameters

lineno	
--------	--

Returns

Python function prototype

```
setBreakPoint(self: pyaubo_sdk.RuntimeMachine, arg0: int) -> int
```

Lua function prototype

```
setBreakPoint(lineno: number) -> number
```

JSON-RPC Request Example

```
{"jsonrpc":"2.0","method":"RuntimeMachine.setBreakPoint","params":[15],"id":1}
```

JSON-RPC Response Example

```
{"id":1,"jsonrpc":"2.0","result":0}
```

Parameters

lineno	
comment	

Returns

JSON-RPC Request Example

```
{"jsonrpc":"2.0","method":"RuntimeMachine.setLabel","params":[5,"moveJoint"],"id":1}
```

JSON-RPC Response Example

```
{"id":1,"jsonrpc":"2.0","result":0}
```

8.7.2.32 setPlanContext()

```
ARCS_DEPRECATED int arcs::common_interface::RuntimeMachine::setPlanContext (
    int tid,
    int lineno,
    const std::string & comment)
```

Add a comment to the aubo__control log Use setLabel instead

Parameters

tid	Thread ID of the instruction
lineno	Line number
comment	Comment

Returns

Python function prototype

```
setPlanContext(self: pyaubo_sdk.RuntimeMachine, arg0: int, arg1: int, arg2: str) -> int
```

Lua function prototype

```
setPlanContext(tid: number, lineno: number, comment: string) -> number
```

JSON-RPC Request Example

```
{"jsonrpc":"2.0","method":"RuntimeMachine.setPlanContext","params":[26,3,"moveJoint"],"id":1}
```

JSON-RPC Response Example

```
{"id":1,"jsonrpc":"2.0","result":0}
```


8.7.2.33 setResumeWait()

```
int arcs::common_interface::RuntimeMachine::setResumeWait (
    bool wait)
```

Wait for the previous sequence to complete before resuming the interpreter

Parameters

wait	
------	--

Returns

JSON-RPC Request Example

```
{"jsonrpc":"2.0","method":"RuntimeMachine.setResumeWait","params":[true],"id":1}
```

JSON-RPC Response Example

```
{"id":1,"jsonrpc":"2.0","result":0}
```

8.7.2.34 start()

```
int arcs::common_interface::RuntimeMachine::start ()
```

Start the runtime

Returns

Python function prototype

```
start(self: pyaubo_sdk.RuntimeMachine) -> int
```

Lua function prototype

```
start() -> number
```

JSON-RPC Request Example

```
{"jsonrpc":"2.0","method":"RuntimeMachine.start","params":[],"id":1}
```

JSON-RPC Response Example

```
{"id":1,"jsonrpc":"2.0","result":0}
```

8.7.2.35 `step()`

```
int arcs::common_interface::RuntimeMachine::step ()
```

Execute a single step

Returns

Python function prototype

```
step(self: pyaubo_sdk.RuntimeMachine) -> int
```

Lua function prototype

```
step() -> number
```

JSON-RPC Request Example

```
{"jsonrpc":"2.0","method":"RuntimeMachine.step","params":[],"id":1}
```

JSON-RPC Response Example

```
{"id":1,"jsonrpc":"2.0","result":0}
```

8.7.2.36 `stop()`

```
int arcs::common_interface::RuntimeMachine::stop ()
```

Stop the runtime, i.e., stop script execution.

Cannot stop robot motion when the runtime state is Stopped.

If you want to stop all robot motion, use the [RuntimeMachine::abort](#) interface.

Returns

Python function prototype

```
stop(self: pyaubo_sdk.RuntimeMachine) -> int
```

Lua function prototype

```
stop() -> number
```

JSON-RPC Request Example

```
{"jsonrpc":"2.0","method":"RuntimeMachine.stop","params":[],"id":1}
```

Parameters

tid	
-----	--

Returns

JSON-RPC Request Example

```
{"jsonrpc": "2.0", "method": "RuntimeMachine.switchTask", "params": [26], "id": 1}
```

JSON-RPC Response Example

```
{"id": 1, "jsonrpc": "2.0", "result": 0}
```

8.7.2.38 timerDelete()

```
int arcs::common_interface::RuntimeMachine::timerDelete (  
    const std::string & name)
```

Delete the timer

Parameters

name	
------	--

Returns

Python function prototype

```
timerDelete(self: pyaubo_sdk.RuntimeMachine, arg0: str) -> int
```

Lua function prototype

```
timerDelete(name: string) -> nil
```

JSON-RPC Request Example

```
{"jsonrpc": "2.0", "method": "RuntimeMachine.timerDelete", "params": ["timer"], "id": 1}
```

JSON-RPC Response Example

```
{"id": 1, "jsonrpc": "2.0", "result": 0}
```

Parameters

name	
------	--

Returns

Python function prototype

```
timerReset(self: pyaubo_sdk.RuntimeMachine, arg0: str) -> int
```

Lua function prototype

```
timerReset(name: string) -> nil
```

JSON-RPC Request Example

```
{"jsonrpc": "2.0", "method": "RuntimeMachine.timerReset", "params": ["timer"], "id": 1}
```

JSON-RPC Response Example

```
{"id": 1, "jsonrpc": "2.0", "result": 0}
```

8.7.2.40 timerStart()

```
int arcs::common_interface::RuntimeMachine::timerStart (  
    const std::string & name)
```

Start the timer

Parameters

name	
------	--

Returns

Python function prototype

```
timerStart(self: pyaubo_sdk.RuntimeMachine, arg0: str) -> int
```

Lua function prototype

```
timerStart(name: string) -> nil
```

JSON-RPC Request Example

```
{"jsonrpc": "2.0", "method": "RuntimeMachine.timerStart", "params": ["timer"], "id": 1}
```

JSON-RPC Response Example

```
{"id": 1, "jsonrpc": "2.0", "result": 0}
```

8.7.2.41 timerStop()

```
int arcs::common_interface::RuntimeMachine::timerStop (  
    const std::string & name)
```

Stop the timer

Parameters

name	
------	--

Returns

Python function prototype

```
timerStop(self: pyaubo_sdk.RuntimeMachine, arg0: str) -> int
```

Lua function prototype

```
timerStop(name: string) -> nil
```

JSON-RPC Request Example

```
{"jsonrpc": "2.0", "method": "RuntimeMachine.timerStop", "params": ["timer"], "id": 1}
```

JSON-RPC Response Example

```
{"id": 1, "jsonrpc": "2.0", "result": 0}
```

8.7.2.42 triggBegin()

```
int arcs::common_interface::RuntimeMachine::triggBegin (  
    double distance,  
    double delay)
```

Start configuring trigger

Parameters

distance	
----------	--

delay	
-------	--

Returns

JSON-RPC Request Example

```
{"jsonrpc":"2.0","method":"RuntimeMachine.triggBegin","params":[],"id":1}
```

JSON-RPC Response Example

```
{"id":1,"jsonrpc":"2.0","result":0}
```

8.7.2.43 triggEnd()

```
int arcs::common_interface::RuntimeMachine::triggEnd ()
```

End configuring trigger

Returns

JSON-RPC Request Example

```
{"jsonrpc":"2.0","method":"RuntimeMachine.triggEnd","params":[],"id":1}
```

JSON-RPC Response Example

```
{"id":1,"jsonrpc":"2.0","result":0}
```

8.7.2.44 triggInterrupt()

```
int arcs::common_interface::RuntimeMachine::triggInterrupt (
    double distance,
    double delay)
```

Returns the automatically assigned interrupt number

Parameters

distance	
delay	
intnum	

Returns

8.8 Serial ()

[Serial](#) control.

Functions

- int [arcs::common_interface::Serial::serialOpen](#) (const std::string &device, int baud, float stop_bits, int even, const std::string &serial_name="serial_0")
Open TCP/IP ethernet communication serial
- int [arcs::common_interface::Serial::serialClose](#) (const std::string &serial_name="serial_0")
Close TCP/IP serial communication Close down the serial connection to the server.
- int [arcs::common_interface::Serial::serialReadByte](#) (const std::string &variable, const std::string &serial_name="serial_0")
Reads a number of bytes from the serial.
- int [arcs::common_interface::Serial::serialReadByteList](#) (int number, const std::string &variable, const std::string &serial_name="serial_0")
Reads a number of bytes from the serial.
- int [arcs::common_interface::Serial::serialReadString](#) (const std::string &variable, const std::string &serial_name="serial_0", const std::string &prefix="", const std::string &suffix="", bool interpret_escape=false)
Reads all data from the serial and returns the data as a string.
- int [arcs::common_interface::Serial::serialSendByte](#) (char value, const std::string &serial_name="serial_0")
Sends a byte to the server Sends the byte through the serial.
- int [arcs::common_interface::Serial::serialSendInt](#) (int value, const std::string &serial_name="serial_0")
Sends an int (int32_t) to the server Sends the int through the serial.
- int [arcs::common_interface::Serial::serialSendLine](#) (const std::string &str, const std::string &serial_name="serial_0")
Sends a string with a newline character to the server Sends the string <str> through the serial in ASCII coding, appending a newline at the end.
- int [arcs::common_interface::Serial::serialSendString](#) (const std::string &str, const std::string &serial_name="serial_0")
Sends a string to the server Sends the string <str> through the serial in ASCII coding.
- int [arcs::common_interface::Serial::serialSendAllString](#) (bool is_check, const std::vector< char > &str, const std::string &serial_name="serial_0")

8.8.1 Detailed Description

[Serial](#) control.

8.8.2 Function Documentation

8.8.2.1 serialClose()

```
int arcs::common_interface::Serial::serialClose (
    const std::string & serial_name = "serial_0")
```

Close TCP/IP serial communication Close down the serial connection to the server.

Parameters

serial_name	
-------------	--

Returns

Python function prototype

```
serialClose(self: pyaubo_sdk.Serial, arg0: str) -> int
```

Lua function prototype

```
serialClose(serial_name: string) -> nil
```

8.8.2.2 serialOpen()

```
int arcs::common_interface::Serial::serialOpen (
    const std::string & device,
    int baud,
    float stop_bits,
    int even,
    const std::string & serial_name = "serial_0")
```

Open TCP/IP ethernet communication serial

Parameters

device	
baud	
stop_bits	
even	
serial_name	

Returns

Python function prototype

```
serialOpen(self: pyaubo_sdk.Serial, arg0: str, arg1: int, arg2: float, arg3: int, arg4: str) -> int
```

Lua function prototype

```
serialOpen(device: string, baud: number, stop_bits: number, even: number, serial_name: string)  
-> nil
```

8.8.2.3 serialReadByte()

```
int arcs::common_interface::Serial::serialReadByte (  
    const std::string & variable,  
    const std::string & serial_name = "serial_0")
```

Reads a number of bytes from the serial.

Bytes are in network byte order. A maximum of 30 values can be read in one command.

Parameters

variable	
serial_name	

Returns

Python function prototype

```
serialReadByte(self: pyaubo_sdk.Serial, arg0: str, arg1: str) -> int
```

Lua function prototype

```
serialReadByte(variable: string, serial_name: string) -> number
```

8.8.2.4 serialReadByteList()

```
int arcs::common_interface::Serial::serialReadByteList (
    int number,
    const std::string & variable,
    const std::string & serial_name = "serial_0")
```

Reads a number of bytes from the serial.

Bytes are in network byte order. A maximum of 30 values can be read in one command. A list of numbers read (list of ints, length=number+1)

Parameters

number	Number of bytes to read
variable	
serial_name	Serial port name

Returns

Return value

Python function prototype

```
serialReadByteList(self: pyaubo_sdk.Serial, arg0: int, arg1: str, arg2: str) -> int
```

Lua function prototype

```
serialReadByteList(number: number, variable: string, serial_name: string) -> number
```

8.8.2.5 serialReadString()

```
int arcs::common_interface::Serial::serialReadString (
    const std::string & variable,
    const std::string & serial_name = "serial_0",
    const std::string & prefix = "",
    const std::string & suffix = "",
    bool interpret_escape = false)
```

Reads all data from the serial and returns the data as a string.

Bytes are in network byte order.

The optional parameters "prefix" and "suffix", can be used to express what is extracted from the serial. The "prefix" specifies the start of the substring (message) extracted from the serial. The data up to the end of the "prefix" will be ignored and removed from the serial. The "suffix" specifies the end of the substring (message) extracted from the serial. Any remaining data on the serial, after the "suffix", will be preserved. E.g. if the serial server sends a string "noise>hello<", the controller can receive the "hello" by calling this script function with the prefix=">" and suffix="<". By using the "prefix" and "suffix" it is also possible send multiple string to the controller at once, because the suffix defines where the message ends. E.g. sending ">hello<>world<"

Parameters

variable	
serial_name	
prefix	
suffix	
interpret_escape	

Returns

Python function prototype

```
serialReadString(self: pyaubo_sdk.Serial, arg0: str, arg1: str, arg2: str, arg3: str, arg4: bool) ->
int
```

Lua function prototype

```
serialReadString(variable: string, serial_name: string, prefix: string, suffix: string, interpret_↵
escape: boolean) -> number
```

8.8.2.6 serialSendAllString()

```
int arcs::common_interface::Serial::serialSendAllString (
    bool is_check,
    const std::vector< char > & str,
    const std::string & serial_name = "serial_0")
```

Parameters

is_check	Whether to check
str	Array of strings
serial_name	Serial port name

Returns

Return value

Python function prototype

```
serialSendAllString(self: pyaubo_sdk.Serial, arg0: bool, arg1: List[str], arg2: str) -> int
```

Lua function prototype

```
serialSendAllString(is_check: boolean, str: table, serial_name: string) -> nil
```

8.8.2.7 serialSendByte()

```
int arcs::common_interface::Serial::serialSendByte (  
    char value,  
    const std::string & serial_name = "serial_0")
```

Sends a byte to the server Sends the byte
through the serial.

Expects no response. Can be used to send special ASCII characters; 10 is newline, 2 is start of text, 3 is end of text.

Parameters

value	
serial_name	

Returns

Python function prototype

```
serialSendByte(self: pyaubo_sdk.Serial, arg0: str, arg1: str) -> int
```

Lua function prototype

```
serialSendByte(value: string, serial_name: string) -> nil
```

8.8.2.8 serialSendInt()

```
int arcs::common_interface::Serial::serialSendInt (  
    int value,  
    const std::string & serial_name = "serial_0")
```

Sends an int (int32_t) to the server Sends the int
through the serial.

Send in network byte order. Expects no response.

Parameters

value	
serial_name	

Returns

Python function prototype

```
serialSendInt(self: pyaubo_sdk.Serial, arg0: int, arg1: str) -> int
```

Lua function prototype

```
serialSendInt(value: number, serial_name: string) -> nil
```

8.8.2.9 serialSendLine()

```
int arcs::common_interface::Serial::serialSendLine (  
    const std::string & str,  
    const std::string & serial_name = "serial_0")
```

Sends a string with a newline character to the server Sends the string <str> through the serial in ASCII coding, appending a newline at the end.

Expects no response.

Parameters

str	
serial_name	

Returns

Python function prototype

```
serialSendLine(self: pyaubo_sdk.Serial, arg0: str, arg1: str) -> int
```

Lua function prototype

```
serialSendLine(str: string, serial_name: string) -> nil
```

Parameters

str	
serial_name	

Returns

Python function prototype

```
serialSendString(self: pyaubo_sdk.Serial, arg0: str, arg1: str) -> int
```

Lua function prototype

```
serialSendString(str: string, serial_name: string) -> nil
```

8.9 Socket (socket)

[Socket](#) control.

Functions

- int [arcs::common_interface::Socket::socketOpen](#) (const std::string &address, int port, const std::string &socket_name="socket_0")
Open TCP/IP ethernet communication socket
- int [arcs::common_interface::Socket::socketClose](#) (const std::string &socket_name="socket_0")
Closes TCP/IP socket communication Closes down the socket connection to the server.
- int [arcs::common_interface::Socket::socketReadAsciiFloat](#) (int number, const std::string &variable, const std::string &socket_name="socket_0")
Reads a number of ascii formatted floats from the socket.
- int [arcs::common_interface::Socket::socketReadBinaryInteger](#) (int number, const std::string &variable, const std::string &socket_name="socket_0")
Reads a number of 32 bit integers from the socket.
- int [arcs::common_interface::Socket::socketReadByteList](#) (int number, const std::string &variable, const std::string &socket_name="socket_0")
Reads a number of bytes from the socket.
- int [arcs::common_interface::Socket::socketReadString](#) (const std::string &variable, const std::string &socket_name="socket_0", const std::string &prefix="", const std::string &suffix="", bool interpret_escape=false)
Reads all data from the socket and returns the data as a string.
- int [arcs::common_interface::Socket::socketReadAllString](#) (const std::string &variable, const std::string &socket_name="socket_0")
Reads all data from the socket and returns the data as a vector of chars.
- int [arcs::common_interface::Socket::socketSendByte](#) (char value, const std::string &socket_name="socket_0")

- Sends a byte to the server Sends the byte through the socket.
- int [arcs::common_interface::Socket::socketSendInt](#) (int value, const std::string &socket_name="socket_0")
Sends an int (int32_t) to the server Sends the int through the socket.
- int [arcs::common_interface::Socket::socketSendLine](#) (const std::string &str, const std::string &socket_name="socket_0")
Sends a string with a newline character to the server Sends the string <str> through the socket in ASCII coding.
- int [arcs::common_interface::Socket::socketSendString](#) (const std::string &str, const std::string &socket_name="socket_0")
Sends a string to the server Sends the string <str> through the socket in ASCII coding.
- int [arcs::common_interface::Socket::socketSendAllString](#) (bool is_check, const std::vector< char > &str, const std::string &socket_name="socket_0")
Sends all data in the given vector of chars to the server.
- bool [arcs::common_interface::Socket::socketHasConnected](#) (const std::string &socket_name="socket_0")
Check if the socket is connected.

8.9.1 Detailed Description

[Socket](#) control.

8.9.2 Function Documentation

8.9.2.1 socketClose()

```
int arcs::common_interface::Socket::socketClose (
    const std::string & socket_name = "socket_0")
```

Closes TCP/IP socket communication Closes down the socket connection to the server.

Instruction

Parameters

socket_name	
-------------	--

Returns

Python function prototype

```
socketClose(self: pyaubo_sdk.Socket, arg0: str) -> int
```

Lua function prototype

```
socketClose(socket_name: string) -> nil
```

JSON-RPC request example

```
{"jsonrpc": "2.0", "method": "Socket.socketClose", "params": ["socket_0"], "id": 1}
```

JSON-RPC response example

```
{"id": 1, "jsonrpc": "2.0", "result": 0}
```

8.9.2.2 socketHasConnected()

```
bool arcs::common_interface::Socket::socketHasConnected (
    const std::string & socket_name = "socket_0")
```

Check if the socket is connected.

socketHasConnected

Parameters

socket_name	
-------------	--

Returns

Python

```
socketHasConnected(self: pyaubo_sdk.Socket, arg0: str) -> bool
```

Lua

```
socketHasConnected(socket_name: string) -> boolean
```

8.9.2.3 socketOpen()

```
int arcs::common_interface::Socket::socketOpen (
    const std::string & address,
    int port,
    const std::string & socket_name = "socket_0")
```

Open TCP/IP ethernet communication socket

Instruction

Parameters

address	
port	
socket_name	

Returns

Python function prototype

```
socketOpen(self: pyaubo_sdk.Socket, arg0: str, arg1: int, arg2: str) -> int
```

Lua function prototype

```
socketOpen(address: string, port: number, socket_name: string) -> nil
```

JSON-RPC request example

```
{"jsonrpc": "2.0", "method": "Socket.socketOpen", "params": ["172.16.26.248", 8000, "socket_0"],  
  "id": 1}
```

JSON-RPC response example

```
{"id": 1, "jsonrpc": "2.0", "result": 0}
```

8.9.2.4 socketReadAllString()

```
int arcs::common_interface::Socket::socketReadAllString (  
    const std::string & variable,  
    const std::string & socket_name = "socket_0")
```

Reads all data from the socket and returns the data as a vector of chars.

Instruction `std::vector<char>`

Parameters

variable	
socket_name	

Returns

Python function prototype

```
socketReadAllString(self: pyaubo_sdk.Socket, arg0: str, arg1: str) -> int
```

Lua function prototype

```
socketReadAllString(variable: string, socket_name: string) -> number
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"Socket.socketReadAllString","params":["camera","socket_0"],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":0}
```

8.9.2.5 socketReadAsciiFloat()

```
int arcs::common_interface::Socket::socketReadAsciiFloat (
    int number,
    const std::string & variable,
    const std::string & socket_name = "socket_0")
```

Reads a number of ascii formatted floats from the socket.

A maximum of 30 values can be read in one command. A list of numbers read (list of floats, length=number+1)

Result will be stored in a register named reg_key. Use getFloatVec to retrieve data

Parameters

number	
variable	
socket_name	

Returns

Python function prototype

```
socketReadAsciiFloat(self: pyaubo_sdk.Socket, arg0: int, arg1: str, arg2: str) -> int
```

Lua function prototype

```
socketReadAsciiFloat(number: number, variable: string, socket_name: string) -> number
```

8.9.2.6 socketReadBinaryInteger()

```
int arcs::common_interface::Socket::socketReadBinaryInteger (
    int number,
    const std::string & variable,
    const std::string & socket_name = "socket_0")
```

Reads a number of 32 bit integers from the socket.

Bytes are in network byte order. A maximum of 30 values can be read in one command. A list of numbers read (list of ints, length=number+1)

Instruction

std::vector<int>

Parameters

number	
variable	
socket_name	

Returns

Python function prototype

```
socketReadBinaryInteger(self: pyaubo_sdk.Socket, arg0: int, arg1: str, arg2: str) -> int
```

Lua function prototype

```
socketReadBinaryInteger(number: number, variable: string, socket_name: string) -> number
```

8.9.2.7 socketReadByteList()

```
int arcs::common_interface::Socket::socketReadByteList (
    int number,
    const std::string & variable,
    const std::string & socket_name = "socket_0")
```

Reads a number of bytes from the socket.

Bytes are in network byte order. A maximum of 30 values can be read in one command. A list of numbers read (list of ints, length=number+1)

Instruction

std::vector<char>

Parameters

number	
variable	
socket_name	

Returns

Python function prototype

```
socketReadByteList(self: pyaubo_sdk.Socket, arg0: int, arg1: str, arg2: str) -> int
```

Lua function prototype

```
socketReadByteList(number: number, variable: string, socket_name: string) -> number
```

8.9.2.8 socketReadString()

```
int arcs::common_interface::Socket::socketReadString (
    const std::string & variable,
    const std::string & socket_name = "socket_0",
    const std::string & prefix = "",
    const std::string & suffix = "",
    bool interpret_escape = false)
```

Reads all data from the socket and returns the data as a string.

Bytes are in network byte order.

The optional parameters "prefix" and "suffix", can be used to express what is extracted from the socket. The "prefix" specifies the start of the substring (message) extracted from the socket. The data up to the end of the "prefix" will be ignored and removed from the socket. The "suffix" specifies the end of the substring (message) extracted from the socket. Any remaining data on the socket, after the "suffix", will be preserved. E.g. if the socket server sends a string "noise>hello<", the controller can receive the "hello" by calling this script function with the prefix=">" and suffix="<". By using the "prefix" and "suffix" it is also possible send multiple string to the controller at once, because the suffix defines where the message ends. E.g. sending ">hello<>world<"

Instruction

```
std::string
```

Parameters

variable	
socket_name	
prefix	
suffix	
interpret_escape	

Returns

Python function prototype

```
socketReadString(self: pyaubo_sdk.Socket, arg0: str, arg1: str, arg2: str, arg3: str, arg4: bool) ->
int
```

Lua function prototype

```
socketReadString(variable: string, socket_name: string, prefix: string, suffix: string, interpret_↵
escape: boolean) -> number
```

JSON-RPC request example

```
{ "jsonrpc": "2.0", "method": "Socket.socketReadString", "params": [ "camera", "socket_0", "", "", false ] ↵
, "id": 1 }
```

JSON-RPC response example

```
{ "id": 1, "jsonrpc": "2.0", "result": 0 }
```

8.9.2.9 socketSendAllString()

```
int arcs::common_interface::Socket::socketSendAllString (
    bool is_check,
    const std::vector< char > & str,
    const std::string & socket_name = "socket_0")
```

Sends all data in the given vector of chars to the server.

Parameters

is_check	Whether to check the sending status
----------	-------------------------------------

str	The data to send as a vector of chars
socket_name	The name of the socket

Returns

Status code

Python function prototype

```
socketSendAllString(self: pyaubo_sdk.Socket, arg0: bool, arg1: List[str], arg2: str) -> int
```

Lua function prototype

```
socketSendAllString(is_check: boolean, str: table, socket_name: string) -> nil
```

8.9.2.10 socketSendByte()

```
int arcs::common_interface::Socket::socketSendByte (
    char value,
    const std::string & socket_name = "socket_0")
```

Sends a byte to the server Sends the byte
through the socket.

Expects no response. Can be used to send special ASCII characters; 10 is newline, 2 is start of text, 3 is end of text.

Instruction

Parameters

value	
socket_name	

Returns

Python function prototype

```
socketSendByte(self: pyaubo_sdk.Socket, arg0: str, arg1: str) -> int
```

Lua function prototype

```
socketSendByte(value: string, socket_name: string) -> nil
```

8.9.2.11 socketSendInt()

```
int arcs::common_interface::Socket::socketSendInt (
    int value,
    const std::string & socket_name = "socket_0")
```

Sends an int (int32_t) to the server Sends the int through the socket.

Send in network byte order. Expects no response

Instruction

Parameters

value	
socket_name	

Returns

Python function prototype

```
socketSendInt(self: pyaubo_sdk.Socket, arg0: int, arg1: str) -> int
```

Lua function prototype

```
socketSendInt(value: number, socket_name: string) -> nil
```

8.9.2.12 socketSendLine()

```
int arcs::common_interface::Socket::socketSendLine (
    const std::string & str,
    const std::string & socket_name = "socket_0")
```

Sends a string with a newline character to the server Sends the string <str> through the socket in ASCII coding.

Expects no response.

Instruction

Parameters

str	
socket_name	

Returns

Python function prototype

```
socketSendLine(self: pyaubo_sdk.Socket, arg0: str, arg1: str) -> int
```

Lua function prototype

```
socketSendLine(str: string, socket_name: string) -> nil
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"Socket.socketSendLine","params":["abcd","socket_0"],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":0}
```

8.9.2.13 socketSendString()

```
int arcs::common_interface::Socket::socketSendString (
    const std::string & str,
    const std::string & socket_name = "socket_0")
```

Sends a string to the server Sends the string <str> through the socket in ASCII coding.

Expects no response.

Instruction

Parameters

str	
socket_name	

Returns

Python function prototype

```
socketSendString(self: pyaubo_sdk.Socket, arg0: str, arg1: str) -> int
```

Lua function prototype

```
socketSendString(str: string, socket_name: string) -> nil
```

JSON-RPC request example

```
{"jsonrpc": "2.0", "method": "Socket.socketSendString", "params": ["abcd", "socket_0"], "id": 1}
```

JSON-RPC response example

```
{"id": 1, "jsonrpc": "2.0", "result": 0}
```

8.10 SyncMove ()

Functions

- int [arcs::common_interface::SyncMove::syncMoveOn](#) (const std::string &syncident, const [TaskSet](#) &taskset)
syncMoveOn is used to start synchronized movement mode.
- bool [arcs::common_interface::SyncMove::syncMoveSegment](#) (int id)
Set the ID for the synchronized path segment.
- int [arcs::common_interface::SyncMove::syncMoveOff](#) (const std::string &syncident)
syncMoveOff is used to end synchronized movement mode.
- int [arcs::common_interface::SyncMove::syncMoveUndo](#) ()
syncMoveUndo is used to turn off synchronized movements, even if not all the other task programs execute the syncMoveUndo instruction.
- int [arcs::common_interface::SyncMove::waitSyncTasks](#) (const std::string &syncident, const [TaskSet](#) &taskset)
waitSyncTasks is used to synchronize several task programs at a specific point in the program.
- bool [arcs::common_interface::SyncMove::isSyncMoveOn](#) ()
isSyncMoveOn is used to tell if the mechanical unit group is in synchronized movement mode.
- int [arcs::common_interface::SyncMove::syncMoveSuspend](#) ()
Suspend synchronized movement mode.
- int [arcs::common_interface::SyncMove::syncMoveResume](#) ()
Resume synchronized movement mode.
- int [arcs::common_interface::SyncMove::frameAdd](#) (const std::string &name, const std::vector<double> &pose, const std::string &ref_name)
Add a frame with the name "name" initialized at the specified pose expressed in the ref_frame coordinate frame.
- int [arcs::common_interface::SyncMove::frameAttach](#) (const std::string &child, const std::string &parent)
Attaches the child frame to the parent world model object.

- `int arcs::common_interface::SyncMove::frameDeleteAll ()`
Delete all frames that have been added to the world model.
- `int arcs::common_interface::SyncMove::frameDelete (const std::string &name)`
Delete the frame with name from the world model.
- `int arcs::common_interface::SyncMove::frameMove (const std::string &name, const std::vector< double > &pose, const std::string &ref_name)`
Changes the placement of the coordinate frame named name to the new placement given by pose that is defined in the ref_name coordinate frame.
- `std::vector< double > arcs::common_interface::SyncMove::frameGetPose (const std::string &name, const std::string &rel_frame, const std::string &ref_frame)`
Get the pose of the name frame relative to the rel_frame frame but expressed in the coordinates of the ref_frame frame.
- `std::vector< double > arcs::common_interface::SyncMove::frameConvertPose (const std::vector< double > &pose, const std::string &from_frame, const std::string &to_frame)`
Convert pose from from_frame to to_frame.
- `bool arcs::common_interface::SyncMove::frameExist (const std::string &name)`
Queries for the existence of a frame by the given name.
- `std::string arcs::common_interface::SyncMove::frameGetParent (const std::string &name)`
Get the parent of the frame named name in the world model.
- `std::vector< std::string > arcs::common_interface::SyncMove::frameGetChildren (const std::string &name)`
Returns a list of immediate child object frame names.
- `int arcs::common_interface::SyncMove::axisGroupAdd (const std::string &name, const std::vector< double > &pose, const std::string &ref_frame)`
Adds a new axis group with the given name to the world model.
- `int arcs::common_interface::SyncMove::axisGroupDelete (const std::string &name)`
Deletes the axis group with the given name from the world model.
- `int arcs::common_interface::SyncMove::axisGroupAddAxis (const std::string &group_name, const std::string &name, const std::string &parent, const std::vector< double > &pose)`
Adds an external axis with the given name to the axis group named group_name.
- `int arcs::common_interface::SyncMove::axisGroupUpdateAxis (const std::string &name, const std::vector< double > &pose)`
Updates the corresponding properties of axis with name.
- `int arcs::common_interface::SyncMove::axisGroupGetAxisIndex (const std::string &name)`
Returns the index of the axis with given axis_name in the RTDE target positions and actual positions arrays.
- `std::string arcs::common_interface::SyncMove::axisGroupGetAxisName (int index)`
Returns the name of the axis with the given axis_index.
- `std::vector< double > arcs::common_interface::SyncMove::axisGroupGetTargetPositions (const std::string &group_name)`
Returns the current target positions of the axis group with group_name.
- `std::vector< double > arcs::common_interface::SyncMove::axisGroupGetActualPositions (const std::string &group_name)`
Returns the current actual positions of the axis group with group_name.
- `int arcs::common_interface::SyncMove::axisGroupOffsetPositions (const std::string &group_name, const std::vector< double > &offset)`
Shifts the target and actual positions of the axis group group_name by the given offset.
- `int arcs::common_interface::SyncMove::axisGroupMoveJoint (const std::string &group_name, const std::vector< double > &q, double a, double v)`
Moves the axes of axis group named group_name to new positions q, using a trapezoidal velocity profile.
- `int arcs::common_interface::SyncMove::axisGroupSpeedJoint (const std::string &group_name, const std::vector< double > &qd, double a, double t)`
Accelerates the axes of axis group named group_name up to the target velocities qd.

8.10.1 Detailed Description

8.10.2 Function Documentation

8.10.2.1 axisGroupAdd()

```
int arcs::common_interface::SyncMove::axisGroupAdd (
    const std::string & name,
    const std::vector< double > & pose,
    const std::string & ref_frame)
```

Adds a new axis group with the given name to the world model.

It is placed at the given pose in the reference coordinate frame defined by ref_frame.

An axis group can only be attached to the world coordinate frame.

Each axis group has a coordinate frame attached to its base, which can be used as an argument to other world model functions by referring the name of the group.

At most 6 axis groups can be added to the world model.

Parameters

name	(string) Name of the axis group to add. The name cannot be an empty string. Names used by world model objects (e.g., frame, axis group, axis, etc.) must be unique.
pose	(pose) Pose of the axis group's base, in the reference coordinate frame.
ref_frame	(optional): (string) Name of the reference coordinate frame that pose is defined in. This can be any world model entity with a coordinate system (e.g., frame, axis group, axis, etc.). The default value "base" refers to the robot's base frame.

Returns

8.10.2.2 axisGroupAddAxis()

```
int arcs::common_interface::SyncMove::axisGroupAddAxis (
    const std::string & group_name,
    const std::string & name,
    const std::string & parent,
    const std::vector< double > & pose)
```

Adds an external axis with the given name to the axis group named group_name.

The axis is attached at the given pose in the reference coordinate frame defined by parent when its axis position is 0. The type, max velocity, max acceleration, position limits, and index of this axis are defined by type, v_limit, a_limit, q_limits, and axis_index, respectively. The pose parameter is typically obtained from a calibration process when the external axis is commissioned. This function will fail if this axis group is under control of another function, or if the kinematic chain created by the attachment forms a closed chain.

Parameters

group_↵ name	Name of the axis group this new axis is added to. The axis group would have been created using axis_group_add(). Axis group with such name must exist.
name	Name of the new axis. The name cannot be an empty string. Names used by world model objects (e.g., frame, axis group, axis, etc.) must be unique.
parent	Name of the parent axis. If it's empty or the same as group_name, the new axis will be attached to the base of the axis group. Axis with such name must exist in the axis group.
pose	The zero-position pose, in the parent coordinate frame, this axis will be placed and attached to. This is the pose the axis will be (relative to its parent) when its axis position is 0. If type is 0 (rotary), then the z axis of the frame corresponds to the axis of rotation. If type is 1 (linear), then the z axis is the axis of translation.
type	Axis type, 0 for rotary, 1 for linear.
v_limit	Maximum velocity.
a_limit	Maximum acceleration.
q_limits	Position limits.
axis_index	Axis index.

8.10.2.3 axisGroupDelete()

```
int arcs::common_interface::SyncMove::axisGroupDelete (
    const std::string & name)
```

Deletes the axis group with the given name from the world model.

All attached axes are also disabled (if live) and deleted.

This function will fail, if this axis group is under control by another function.

Parameters

name	(string) Name of the axis group to delete. Axis group with such name must exist.
------	--

Returns

8.10.2.4 axisGroupGetActualPositions()

```
std::vector< double > arcs::common_interface::SyncMove::axisGroupGetActualPositions (
    const std::string & group_name)
```

Returns the current actual positions of the axis group with group_name.

Generated by Doxygen

If group_name is not provided, the actual positions of all external axes will be returned.

This function will fail, if the external axis bus is disabled.

Parameters

group_↵ name	(optional): (string) Name of the axis group in query. Axis group with such name must exist.
-----------------	---

Returns

Double[]: Actual positions of the involved axes, in the order of their external axis indices.

8.10.2.5 axisGroupGetAxisIndex()

```
int arcs::common_interface::SyncMove::axisGroupGetAxisIndex (
    const std::string & name)
```

Returns the index of the axis with given axis_name in the RTDE target positions and actual positions arrays.

Parameters

axis_name	(string) Name of the axis in query. Axis with such name must exist.
-----------	---

Returns

integer: Index of the axis in the RTDE target positions and actual positions arrays.

8.10.2.6 axisGroupGetAxisName()

```
std::string arcs::common_interface::SyncMove::axisGroupGetAxisName (
    int index)
```

Returns the name of the axis with the given axis_index.

Parameters

axis_index	(integer) Index of the axis in query. Axis with such index must exist.
------------	--

Returns

string: Name of the axis.

Parameters

group_↔ name	(optional): (string) Name of the axis group in query. Axis group with such name must REALLY exist.
-----------------	--

Returns

Double[]: Target positions of the involved axes, in the order of their external axis indices.

8.10.2.8 axisGroupMoveJoint()

```
int arcs::common_interface::SyncMove::axisGroupMoveJoint (
    const std::string & group_name,
    const std::vector< double > & q,
    double a,
    double v)
```

Moves the axes of axis group named group_name to new positions q, using a trapezoidal velocity profile.

Factor a specifying the percentage of the max profile accelerations out of the acceleration limits of each axes. Factor v specifying the percentage of the max profile velocities out of the velocity limits of each axes.

The actual accelerations and velocities are determined by the most constraining axis, so that all the axes complete the acceleration, cruise, and deceleration phases at the same time.

Parameters

group_↔ name	(string) Name of the axis group to move. Axis group with such name must exist.
q	(float[]) Target positions in rad (rotary) or in m (linear). If the target exceeds the position limits, then it is set to the nearest limit. The involved axes are ordered increasingly by their axis indices. The size of q must match the number of axes attached to the given group.
a	(float) Factor specifying the max accelerations of this move out of the acceleration limits. a must be in range of (0,1].
v	(float) Factor specifying the max velocities of this move out of the velocity limits. v must be in range of (0,1].

Return: n/a

8.10.2.9 axisGroupOffsetPositions()

```
int arcs::common_interface::SyncMove::axisGroupOffsetPositions (
    const std::string & group_name,
    const std::vector< double > & offset)
```

Generated by Doxygen

Shifts the target and actual positions of the axis group group_name by the given offset.

This is a software shift that happens in the controller only, it does not affect external axis drives. The

Parameters

group_↔ name	(string) Name of the axis group to apply the offset positions to. Axis group with such name must exist.
offset	(float[]) Offsets that the target and actual positions should be shifted by. The size of offset must match the number of axes attached to the given group.

Returns

8.10.2.10 axisGroupSpeedJoint()

```
int arcs::common_interface::SyncMove::axisGroupSpeedJoint (
    const std::string & group_name,
    const std::vector< double > & qd,
    double a,
    double t)
```

Accelerates the axes of axis group named group_name up to the target velocities qd.

Factor a specifying the percentage of the max accelerations out of the acceleration limits of each axes. The function will run for a period of t seconds.

Parameters

group_↔ name	(string) Name of the external axis group to control. Axis group with such name must exist.
qd	(float[]) Target velocities for the axes in the axis group. If the target exceeds the velocity limits, then it is set to the limit. The involved axes are ordered increasingly by their axis indices. The size of qd must match the number of axes attached to the given group.
a	(float) Factor specifying the max accelerations of this move out of the acceleration limits. a must be in range of (0,1].
t	(optional): (float) Duration in seconds before the function returns. If $t < 0$, then the function will return when the target velocities are reached. if $t = 0$, then the function will return after this duration, regardless of what the achieved axes velocities are.

8.10.2.11 axisGroupUpdateAxis()

```
int arcs::common_interface::SyncMove::axisGroupUpdateAxis (
    const std::string & name,
    const std::vector< double > & pose)
```

Updates the corresponding properties of axis with name.

Generated by Doxygen

The pose parameter is typically obtained from a calibration process when the external axis is commissioned. See here for a guide on a basic routine for calibrating a single rotary axis.

This function will fail, if the axis group the axis attached to is already being controlled by another

Parameters

name	(string) Name of the axis to update. Axis with such name must exist.
pose	(optional): (pose) New zero-position pose, in the coordinate frame of the parent axis (or axis group), of the axis. This is the pose of the axis when its axis position is 0.

8.10.2.12 frameAdd()

```
int arcs::common_interface::SyncMove::frameAdd (
    const std::string & name,
    const std::vector< double > & pose,
    const std::string & ref_name)
```

Add a frame with the name "name" initialized at the specified pose expressed in the ref_frame coordinate frame.

This command only adds a frame to the world, it does not attach it to the ref_frame coordinate frame. Use [frameAttach\(\)](#) to attach the newly added frame to ref_frame if desired.

Parameters

name	name of the frame to be added. The name must not be the same as any existing world model object (frame, axis, or axis group), otherwise an exception is thrown
pose	initial pose of the new object
ref_frame	name of the world model object whose coordinate frame the pose is expressed in. If nothing is provided here, the default is the robot "base" frame.

Returns

8.10.2.13 frameAttach()

```
int arcs::common_interface::SyncMove::frameAttach (
    const std::string & child,
    const std::string & parent)
```

Attaches the child frame to the parent world model object.

The relative transform between the parent and child will be set such that the child does not move in the world when the attachment occurs.

The child cannot be "world", "flange", "tcp", or the same as parent.

This will fail if child or parent is not an existing frame, or this makes the attachments form a closed chain.

Generated by Doxygen

If being used with the MotionPlus, the parent argument can be the name of an external axis or axis group.

Parameters

child	name of the frame to be attached. The name must not be “world”, “flange”, or “tcp”.
parent	name of the object that the child frame will be attached to.

Returns

8.10.2.14 frameConvertPose()

```
std::vector< double > arcs::common_interface::SyncMove::frameConvertPose (
    const std::vector< double > & pose,
    const std::string & from_frame,
    const std::string & to_frame)
```

Convert pose from from_frame to to_frame.

This will fail if either coordinate system argument is not an existing frame.

If being used with MotionPlus, all three arguments can also be the names of external axes or axis groups.

Parameters

pose	pose to be converted
from_frame	name of reference frame at origin of old coordinate system
to_frame	name of reference frame at origin of new coordinate system

Returns

Value of pose expressed in the coordinates of to_frame.

8.10.2.15 frameDelete()

```
int arcs::common_interface::SyncMove::frameDelete (
    const std::string & name)
```

Delete the frame with name from the world model.

The “world”, “base”, “flange”, and “tcp” frames cannot be deleted.

Any frames that are attached to the deleted frame will be attached to the “world” frame with new frame offsets set such that the detached frame does not move in the world.

This command will fail if the frame does not exist.

Parameters

name	name of the frame to be deleted
------	---------------------------------

Returns

8.10.2.16 frameDeleteAll()

```
int arcs::common_interface::SyncMove::frameDeleteAll ()
```

Delete all frames that have been added to the world model.

The “world”, “base”, “flange”, and “tcp” frames cannot be deleted.

Any frames that are attached to the deleted frames will be attached to the “world” frame with new frame offsets set such that the detached frames do not move in the world.

Returns

8.10.2.17 frameExist()

```
bool arcs::common_interface::SyncMove::frameExist (
    const std::string & name)
```

Queries for the existence of a frame by the given name.

Parameters

name	name of the frame to be queried.
------	----------------------------------

Returns

Returns true if there is a frame by the given name, false if not.

8.10.2.18 frameGetChildren()

```
std::vector< std::string > arcs::common_interface::SyncMove::frameGetChildren (
    const std::string & name)
```

Returns a list of immediate child object frame names.

Generated by Doxygen

Parent-child relationships are defined by world model attachments. If being used with MotionPlus, the child objects may also be an axis group or an axis.

Parameters

name	the name of the parent object.
------	--------------------------------

Returns

a list of immediate child object frame names

8.10.2.19 frameGetParent()

```
std::string arcs::common_interface::SyncMove::frameGetParent (
    const std::string & name)
```

Get the parent of the frame named name in the world model.

If the frame is not attached to another frame, then “world” is the parent.

Parameters

name	the frame being queried
------	-------------------------

Returns

name of the parent as a string

8.10.2.20 frameGetPose()

```
std::vector< double > arcs::common_interface::SyncMove::frameGetPose (
    const std::string & name,
    const std::string & rel_frame,
    const std::string & ref_frame)
```

Get the pose of the name frame relative to the rel_frame frame but expressed in the coordinates of the ref_frame frame.

If ref_frame is not provided, then this returns the pose of the name frame relative to and expressed in the same frame as rel_frame.

This will fail if any arguments are not an existing frame.

If being used with MotionPlus, all three arguments can also be the names of external axes or axis groups.

Parameters

name	name of the frame to query.
rel_frame	short for “relative frame” is the frame where the pose is computed relative to
ref_frame	short for “reference frame” is the frame to express the coordinates of resulting relative pose in. If this is not provided, then it will default to match the value of rel_frame.

Returns

The pose of the frame expressed in the ref_frame coordinates.

8.10.2.21 frameMove()

```
int arcs::common_interface::SyncMove::frameMove (
    const std::string & name,
    const std::vector< double > & pose,
    const std::string & ref_name)
```

Changes the placement of the coordinate frame named name to the new placement given by pose that is defined in the ref_name coordinate frame.

This will fail if name is “world”, “flange”, “tcp”, or if the frame does not exist. Note: to move the “tcp” frame, use the set_tcp() command instead.

If being used with the MotionPlus, the ref_name argument can be the name of an external axis or axis group.

Parameters

name	the name of the frame to move
pose	the new placement
ref_name	the coordinate frame that pose is expressed in. The default value is the robot’s “base” frame.

Returns

8.10.2.22 isSyncMoveOn()

```
bool arcs::common_interface::SyncMove::isSyncMoveOn ()
```

isSyncMoveOn is used to tell if the mechanical unit group is in synchronized movement mode.

A task that does not control any mechanical unit can find out if the mechanical units defined in the parameter Use Mechanical Unit Group are in synchronized movement mode.

Returns

Python prototype

```
isSyncMoveOn(self: pyaubo_sdk.SyncMove) -> bool
```

Lua prototype

```
isSyncMoveOn() -> boolean
```

8.10.2.23 syncMoveOff()

```
int arcs::common_interface::SyncMove::syncMoveOff (
    const std::string & syncident)
```

syncMoveOff is used to end synchronized movement mode.

A syncMoveOff instruction will wait for the other task programs. When all task programs have reached the syncMoveOff, they will continue their execution in unsynchronized mode. A stop point must be programmed before the syncMoveOff instruction.

Parameters

syncident	
-----------	--

Returns

Python prototype

```
syncMoveOff(self: pyaubo_sdk.SyncMove, arg0: str) -> int
```

Lua prototype

```
syncMoveOff(syncident: string) -> nil @endcode
```

8.10.2.24 syncMoveOn()

Parameters

syncident	
taskset	

Returns

Python function prototype

```
syncMoveOn(self: pyaubo_sdk.SyncMove, arg0: str, arg1: Set[str]) -> int
```

Lua function prototype

```
syncMoveOn(syncident: string, taskset: table) -> nil @endcode
```

8.10.2.25 syncMoveResume()

```
int arcs::common_interface::SyncMove::syncMoveResume ()
```

Resume synchronized movement mode.

Returns

Python prototype

```
syncMoveResume(self: pyaubo_sdk.SyncMove) -> int
```

Lua prototype

```
syncMoveResume() -> nil
```

8.10.2.26 syncMoveSegment()

```
bool arcs::common_interface::SyncMove::syncMoveSegment (
    int id)
```

Set the ID for the synchronized path segment.

In synchronized movements mode, all or none of the simultaneous move instructions must be programmed with corner zones. This means that move instructions with the same ID must either all have corner zones, or all have stop points. If a move instruction with a corner zone and a move instruction with a stop point are synchronously executed in their respective task program, an error will occur. Synchronously executed move instructions can have corner zones of different sizes (e.g. one uses z10 and one uses z50).

Parameters

id	
----	--

Returns

Python prototype

```
syncMoveSegment(self: pyaubo_sdk.SyncMove, arg0: int) -> bool
```

Lua prototype

```
syncMoveSegment(id: number) -> boolean @endcoe
```

8.10.2.27 syncMoveSuspend()

```
int arcs::common_interface::SyncMove::syncMoveSuspend ()
```

Suspend synchronized movement mode.

Returns

Python prototype

```
syncMoveSuspend(self: pyaubo_sdk.SyncMove) -> int
```

Lua prototype

```
syncMoveSuspend\(\) -> nil
```

8.10.2.28 syncMoveUndo()

```
int arcs::common_interface::SyncMove::syncMoveUndo ()
```

syncMoveUndo is used to turn off synchronized movements, even if not all the other task programs execute the syncMoveUndo instruction.

syncMoveUndo is intended for UNDO handlers. When the program pointer is moved from the procedure, syncMoveUndo is used to turn off the synchronization.

Returns

Python prototype

```
syncMoveUndo(self: pyaubo_sdk.SyncMove) -> int
```

Lua prototype

```
syncMoveUndo\(\) -> nil @endcoe
```

8.10.2.29 waitSyncTasks()

```
int arcs::common_interface::SyncMove::waitSyncTasks (
    const std::string & syncident,
    const TaskSet & taskset)
```

waitSyncTasks is used to synchronize several task programs at a specific point in the program.

A waitSyncTasks instruction will wait for the other task programs. When all task programs have reached the waitSyncTasks instruction, they will continue their execution.

Parameters

syncident	
taskset	

Returns

Python prototype

```
waitSyncTasks(self: pyaubo_sdk.SyncMove, arg0: str, arg1: Set[str]) -> int
```

Lua prototype

```
waitSyncTasks(syncident: string, taskset: table) -> nil @endcode
```

8.11 SystemInfo ()

SystemInfo.

Functions

- int [arcs::common_interface::SystemInfo::getControlSoftwareVersionCode](#) ()
Get the controller software version code
- std::string [arcs::common_interface::SystemInfo::getControlSoftwareFullVersion](#) ()
Get the full controller software version
- int [arcs::common_interface::SystemInfo::getInterfaceVersionCode](#) ()
Get the interface version code
- std::string [arcs::common_interface::SystemInfo::getControlSoftwareBuildDate](#) ()
Get the controller software build date
- std::string [arcs::common_interface::SystemInfo::getControlSoftwareVersionHash](#) ()
Get the controller software git version
- uint64_t [arcs::common_interface::SystemInfo::getControlSystemTime](#) ()
Get the system time (software start time in nanoseconds)

8.11.1 Detailed Description

[SystemInfo](#).

8.11.2 Function Documentation

8.11.2.1 getControlSoftwareBuildDate()

`std::string arcs::common_interface::SystemInfo::getControlSoftwareBuildDate ()`

Get the controller software build date

Returns

Returns the controller software build date

Python prototype

```
getControlSoftwareBuildDate(self: pyaubo_sdk.SystemInfo) -> str
```

Lua prototype

```
getControlSoftwareBuildDate\(\) -> string
```

C++ example

```
std::string build_date =  
rpc_cli->getSystemInfo()->getControlSoftwareBuildDate();
```

JSON-RPC request example

```
{"jsonrpc": "2.0", "method": "SystemInfo.getControlSoftwareBuildDate", "params": [], "id": 1}
```

JSON-RPC response example

```
{"id": 1, "jsonrpc": "2.0", "result": "2024-3-5 07:03:20"}
```

8.11.2.2 getControlSoftwareFullVersion()

`std::string arcs::common_interface::SystemInfo::getControlSoftwareFullVersion ()`

Get the full controller software version

Returns

Returns the full controller software version

Python prototype

```
getControlSoftwareFullVersion(self: pyaubo_sdk.SystemInfo) -> str
```

Lua prototype

```
getControlSoftwareFullVersion() -> string
```

C++ example

```
std::string control_version =
rpc_cli->getSystemInfo()->getControlSoftwareFullVersion();
```

JSON-RPC request example

```
{"jsonrpc": "2.0", "method": "SystemInfo.getControlSoftwareFullVersion", "params": [], "id": 1}
```

JSON-RPC response example

```
{"id": 1, "jsonrpc": "2.0", "result": "0.31.0-alpha.16+20alc76"}
```

8.11.2.3 getControlSoftwareVersionCode()

`int arcs::common_interface::SystemInfo::getControlSoftwareVersionCode ()`

Get the controller software version code

Returns

Returns the controller software version code

Python prototype

```
getControlSoftwareVersionCode(self: pyaubo_sdk.SystemInfo) -> int
```

Lua prototype

```
getControlSoftwareVersionCode() -> number
```

C++ example

```
int control_version =
rpc_cli->getSystemInfo()->getControlSoftwareVersionCode();
```

JSON-RPC request example

```
{"jsonrpc": "2.0", "method": "SystemInfo.getControlSoftwareVersionCode", "params": [], "id": 1}
```

JSON-RPC response example

```
{"id": 1, "jsonrpc": "2.0", "result": 28003}
```

8.11.2.4 getControlSoftwareVersionHash()

```
std::string arcs::common_interface::SystemInfo::getControlSoftwareVersionHash ()
```

Get the controller software git version

Returns

Returns the controller software git version

Python prototype

```
getControlSoftwareVersionHash(self: pyaubo_sdk.SystemInfo) -> str
```

Lua prototype

```
getControlSoftwareVersionHash() -> string
```

C++ example

```
std::string git_version =
rpc_cli->getSystemInfo()->getControlSoftwareVersionHash();
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"SystemInfo.getControlSoftwareVersionHash","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":"fa4f64a"}
```

8.11.2.5 getControlSystemTime()

```
uint64_t arcs::common_interface::SystemInfo::getControlSystemTime ()
```

Get the system time (software start time in nanoseconds)

Returns

Returns the system time (software start time in nanoseconds)

Python prototype

```
getControlSystemTime(self: pyaubo_sdk.SystemInfo) -> int
```

Lua prototype

```
getControlSystemTime() -> number
```

C++ example

```
std::string system_time =
rpc_cli->getSystemInfo()->getControlSystemTime();
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"SystemInfo.getControlSystemTime","params":[],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":9287799079682}
```

8.11.2.6 `getInterfaceVersionCode()`

```
int arcs::common_interface::SystemInfo::getInterfaceVersionCode ()
```

Get the interface version code

Returns

Returns the interface version code

Python prototype

```
getInterfaceVersionCode(self: pyaubo_sdk.SystemInfo) -> int
```

Lua prototype

```
getInterfaceVersionCode\(\) -> number
```

C++ example

```
int interface_version =
rpc_cli->getSystemInfo()->getInterfaceVersionCode();
```

JSON-RPC request example

```
{ "jsonrpc": "2.0", "method": "SystemInfo.getInterfaceVersionCode", "params": [], "id": 1 }
```

JSON-RPC response example

```
{ "id": 1, "jsonrpc": "2.0", "result": 22003 }
```

8.12 `Trace ()`

provides a logging system for controller extension programs

Functions

- `int arcs::common_interface::Trace::alarm (TraceLevel level, int code, const std::vector< std::string > &args={})`
Injects alarm information into the aubo_control log.
- `int arcs::common_interface::Trace::textmsg (const std::string &msg)`
print message into log
- `int arcs::common_interface::Trace::popup (TraceLevel level, const std::string &title, const std::string &msg, int mode)`
Send a popup request to the connected RTDE client
- `RobotMsgVector arcs::common_interface::Trace::peek (size_t num, uint64_t last_time=0)`
peek the latest AlarmInfo (after the last retrieval)

8.12.1 Detailed Description

provides a logging system for controller extension programs

8.12.2 Function Documentation

8.12.2.1 alarm()

```
int arcs::common_interface::Trace::alarm (
    TraceLevel level,
    int code,
    const std::vector< std::string > & args = {})
```

Injects alarm information into the aubo_control log.

[TraceLevel](#):

0 - FATAL

1 - ERROR

2 - WARNING

3 - INFO

4 - DEBUG

Code definitions refer to [error_stack](#)

Parameters

level	
code	
args	

Returns

Python function prototype

```
alarm(self: pyaubo_sdk.Trace, arg0: arcs::common\_interface::TraceLevel, arg1: int, arg2: List↔
[str]) -> int
```

Lua function prototype

```
alarm(level: number, code: number, args: table) -> nil
```

JSON-RPC request example

```
{ "jsonrpc": "2.0", "method": "rob1.Trace.alarm", "params": [ "", 1, ["Error", "Trajectory planning
failed!", "1"] ], "id": 1 }
```

8.12.2.2 peek()

Parameters

num	
last_time	

Returns

Python function prototype

```
peek(self: pyaubo_sdk.Trace, arg0: int, arg1: int) -> List[arcs::common_interface::RobotMsg]
```

Lua function prototype

```
peek(num: number, last_time: number) -> table
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.Trace.peak","params":[1,0],"id":1}
```

JSON-RPC response example

```
{{"id":1,"jsonrpc":"2.0","result":{"args":["RobotModeType.Running"], "code":30045,"level":"↔  
INFO","source":"rob1","timestamp":5102883064300}}}
```

8.12.2.3 popup()

```
int arcs::common_interface::Trace::popup (
    TraceLevel level,
    const std::string & title,
    const std::string & msg,
    int mode)
```

Send a popup request to the connected RTDE client

Parameters

level	
title	
msg	

mode	mode
	0: normal mode
	1: blocking mode
	2: input mode bool
	3: input mode int
	4: input mode double
	5: input mode string

Returns

Python function prototype

```
popup(self: pyaubo_sdk.Trace, arg0: arcs::common_interface::TraceLevel, arg1: str, arg2: str,
arg3: int) -> int
```

Lua function prototype

```
popup(level: number, title: string, msg: string, mode: number) -> nil
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"rob1.Trace.popup","params":["","Error","Trajectory planning failed!"],1}↵
,"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":0}
```

8.12.2.4 textmsg()

```
int arcs::common_interface::Trace::textmsg (
    const std::string & msg)
```

print message into log

Parameters

msg	message information
-----	---------------------

Returns

Python function prototype

```
textmsg(self: pyaubo_sdk.Trace, arg0: str) -> int
```

Lua function prototype

```
textmsg(msg: string) -> nil
```

JSON-RPC request example

```
{"jsonrpc": "2.0", "method": "rob1.Trace.textmsg", "params": ["test"], "id": 1}
```

JSON-RPC response example

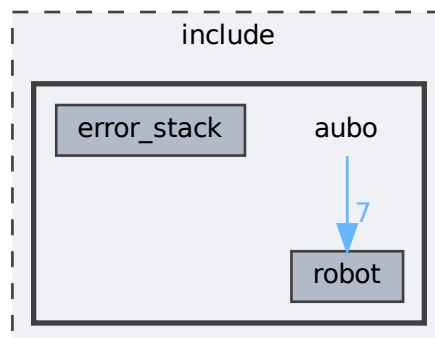
```
{"id": 1, "jsonrpc": "2.0", "result": 0}
```


Chapter 9

Directory Documentation

9.1 include/aubo Directory Reference

Directory dependency graph for aubo:



Directories

- directory [error_stack](#)
- directory [robot](#)

Files

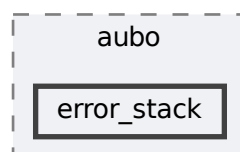
- file [aubo_api.h](#)
API for controlling the robot and external axis.
- file [axis_interface.h](#)
- file [gripper_interface.h](#)

- file [math.h](#)
Mathematic operation interface, such as euler to quaternion conversion, addition/subtraction of poses.
- file [register_control.h](#)
- file [robot_interface.h](#)
API
- file [runtime_machine.h](#)
Script interpreter runtime interface, allows pausing the script interpreter and setting/removing break-points.
- file [serial.h](#)
- file [socket.h](#)
socket
- file [sync_move.h](#)
- file [system_info.h](#)
- file [trace.h](#)
Interface for injecting logs into the controller's logging system.
- file [type_def.h](#)
enum type definitions

9.2 doc Directory Reference

9.3 include/aubo/error_stack Directory Reference

Directory dependency graph for error_stack:



Files

- file [error_stack.h](#)
- file [hal_error.h](#)
- file [rtm_error.h](#)
- file [system_error.h](#)

9.4 include Directory Reference

Directory dependency graph for include:

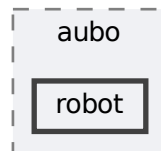


Directories

- directory [aubo](#)

9.5 include/aubo/robot Directory Reference

Directory dependency graph for robot:



Files

- file [force_control.h](#)
- file [io_control.h](#)
IO
- file [motion_control.h](#)
Motion control interface.
- file [robot_algorithm.h](#)
- file [robot_config.h](#)
DH
- file [robot_manage.h](#)
- file [robot_state.h](#)
/

Chapter 10

Namespace Documentation

10.1 arcs Namespace Reference

Namespaces

- namespace [common_interface](#)
- namespace [error_stack](#)

10.2 arcs::common_interface Namespace Reference

Classes

- class [AuboApi](#)
- class [AxisInterface](#)
- class [GripperInterface](#)
- class [Math](#)
- class [RegisterControl](#)
- class [ForceControl](#)
- class [IoControl](#)
- class [MotionControl](#)
- class [RobotAlgorithm](#)
- class [RobotConfig](#)
- class [RobotManage](#)
- class [RobotState](#)
- class [RobotInterface](#)
- class [RuntimeMachine](#)
- class [Serial](#)
- class [Socket](#)
- class [SyncMove](#)
- class [SystemInfo](#)
- class [Trace](#)
- struct [RobotSafetyParameterRange](#)
- struct [WObjectData](#)
- struct [VibrationRecalibrationParameter](#)
- struct [CircleParameters](#)

Circular motion parameters definition.

- struct [SpiralParameters](#)
- struct [Enveloping](#)
- struct [TrajConfig](#)
Trajectory configuration for payload identification.
- struct [RobotMsg](#)
- struct [GripperStatus](#)
- struct [RtdeRecipe](#)
RTDE menu.
- class [AuboException](#)
Custom exception class [AuboException](#).

Typedefs

- using [AuboApiPtr](#) = std::shared_ptr<[AuboApi](#)>
- using [AxisInterfacePtr](#) = std::shared_ptr<[AxisInterface](#)>
- using [GripperInterfacePtr](#) = std::shared_ptr<[GripperInterface](#)>
- using [MathPtr](#) = std::shared_ptr<[Math](#)>
- using [RegisterControlPtr](#) = std::shared_ptr<[RegisterControl](#)>
- using [ForceControlPtr](#) = std::shared_ptr<[ForceControl](#)>
- using [IoControlPtr](#) = std::shared_ptr<[IoControl](#)>
- using [MotionControlPtr](#) = std::shared_ptr<[MotionControl](#)>
- using [RobotAlgorithmPtr](#) = std::shared_ptr<[RobotAlgorithm](#)>
- using [RobotConfigPtr](#) = std::shared_ptr<[RobotConfig](#)>
- using [RobotManagePtr](#) = std::shared_ptr<[RobotManage](#)>
- using [RobotStatePtr](#) = std::shared_ptr<[RobotState](#)>
- using [RobotInterfacePtr](#) = std::shared_ptr<[RobotInterface](#)>
- using [RuntimeMachinePtr](#) = std::shared_ptr<[RuntimeMachine](#)>
- using [SerialPtr](#) = std::shared_ptr<[Serial](#)>
- using [SocketPtr](#) = std::shared_ptr<[Socket](#)>
- typedef std::unordered_set< std::string > [TaskSet](#)
- using [SyncMovePtr](#) = std::shared_ptr<[SyncMove](#)>
- using [SystemInfoPtr](#) = std::shared_ptr<[SystemInfo](#)>
- using [TracePtr](#) = std::shared_ptr<[Trace](#)>
- using [Vector3d](#) = std::array<double, 3>
- using [Vector4d](#) = std::array<double, 4>
- using [Vector3f](#) = std::array<float, 3>
- using [Vector4f](#) = std::array<float, 4>
- using [Vector6f](#) = std::array<float, 6>
- using [ResultWithErrno](#) = std::tuple<std::vector<double>, int>
- using [ResultWithErrno1](#) = std::tuple<std::vector<std::vector<double>>, int>
- using [ResultWithErrno2](#) = std::tuple<std::vector<std::string>, int>
- using [ResultWithErrno3](#) = std::tuple<int, int>
- using [Payload](#)
- using [ForceSensorCalibResult](#)
- using [ForceSensorCalibResultWithError](#)
- using [DynamicsModel](#)
- using [Box](#) = std::vector<double>
- using [Cylinder](#) = std::vector<double>
- using [Sphere](#) = std::vector<double>
- using [RobotMsgVector](#) = std::vector<[RobotMsg](#)>
- using [GripperStatusVector](#) = std::vector<[GripperStatus](#)>

Enumerations

- enum `PathBufferType` {
`PathBuffer_TOPPRA` = 1 , `PathBuffer_CubicSpline` , `PathBuffer_JointSpline` = 3 ,
`PathBuffer_JointSplineC` = 4 ,
`PathBuffer_JointBSpline` = 5 , `PathBuffer_JointBSplineC` = 6 }
 pathBuffer Type
- enum `AuboErrorCodes` : int { `ENUM_AuboErrorCodes_DECLARES` }
- enum class `RuntimeState` : int { `ENUM_RuntimeState_DECLARES` }
- enum class `RobotModeType` : int { `ENUM_RobotModeType_DECLARES` }
- enum class `AxisModeType` : int { `ENUM_AxisModeType_DECLARES` }
- enum class `SafetyModeType` : int { `ENUM_SafetyModeType_DECLARES` }
- Safety Mode.
 - enum class `OperationalModeType` : int { `ENUM_OperationalModeType_DECLARES` }
 - Operational Mode.
 - enum class `RobotControlModeType` : int { `ENUM_RobotControlModeType_DECLARES` }
 - Robot Control Mode.
 - enum class `JointServoModeType` : int { `ENUM_JointServoModeType_DECLARES` }
 - Joint Servo Mode.
 - enum class `JointStateType` : int { `ENUM_JointStateType_DECLARES` }
 - Joint State.
 - enum class `StandardOutputRunState` : int { `ENUM_StandardOutputRunState_DECLARES` }
 - Standard Output Run State.
 - enum class `StandardInputAction` : int { `ENUM_StandardInputAction_DECLARES` }
 - The `StandardInputAction` enum.
 - enum class `SafetyInputAction` : int { `ENUM_SafetyInputAction_DECLARES` }
 - enum class `SafetyOutputRunState` : int { `ENUM_SafetyOutputRunState_DECLARES` }
 - enum `TaskFrameType` { `ENUM_TaskFrameType_DECLARES` }
 - enum `EnvelopingShape` : int { `ENUM_EnvelopingShape_DECLARES` }
 - enum `PayloadIdentifyMoveAxis` : int { `ENUM_PayloadIdentifyMoveAxis_DECLARES` }
 - enum `TraceLevel` { `ENUM_TraceLevel_DECLARES` }
 - enum `SafeguardedStopType` : int { `ENUM_SafeguardedStopType_DECLARES` }
 - enum `RobotEmergencyStopType` : int { `ENUM_RobotEmergencyStopType_DECLARES` }
 - enum class `ForceControlState` { `Stopped` , `Starting` , `Stropping` , `Running` }
 - enum class `RefFrameType` { `None` , `Tool` , `Path` , `Base` }
 - enum `error_type` {
`parse_error` = -32700 , `invalid_request` = -32600 , `method_not_found` = -32601 , `invalid_params` = -32602 ,
`internal_error` = -32603 , `server_error` = -32500 , `invalid` = -32400 }
 Error type.
 - enum `ExceptionCode` {
`EC_DISCONNECTED` = -1 , `EC_NOT_LOGINED` = -2 , `EC_INVAL_SOCKET` = -3 ,
`EC_REQUEST_BUSY` = -4 ,
`EC_SEND_FAILED` = -5 , `EC_RECV_TIMEOUT` = -6 , `EC_RECV_ERROR` = -7 ,
`EC_PARSE_ERROR` = -8 ,
`EC_INVALID_REQUEST` = -9 , `EC_METHOD_NOT_FOUND` = -10 , `EC_INVALID_PARAMS` = -11 , `EC_INTERNAL_ERROR` = -12 ,
`EC_SERVER_ERROR` = -13 , `EC_INVALID` = -14 }
 Exception code.

Functions

- `std::ostream & operator<< (std::ostream &os, const RobotSafetyParameterRange &vd)`
- `std::ostream & operator<< (std::ostream &os, WObjectData p)`
- `std::ostream & operator<< (std::ostream &os, const VibrationRecalibrationParameter p)`
- `std::ostream & operator<< (std::ostream &os, CircleParameters p)`
- `std::ostream & operator<< (std::ostream &os, SpiralParameters p)`
- `std::ostream & operator<< (std::ostream &os, Enveloping p)`
- `std::ostream & operator<< (std::ostream &os, TrajConfig p)`
- `const char * returnValue2Str (int retval)`

10.2.1 Typedef Documentation

10.2.1.1 AuboApiPtr

using `arcs::common_interface::AuboApiPtr` = `std::shared_ptr<AuboApi>`

Definition at line 390 of file `aubo_api.h`.

10.2.1.2 AxisInterfacePtr

using `arcs::common_interface::AxisInterfacePtr` = `std::shared_ptr<AxisInterface>`

Definition at line 417 of file `axis_interface.h`.

10.2.1.3 Box

using `arcs::common_interface::Box` = `std::vector<double>`

Definition at line 814 of file `type_def.h`.

10.2.1.4 Cylinder

using `arcs::common_interface::Cylinder` = `std::vector<double>`

Definition at line 821 of file `type_def.h`.

10.2.1.5 DynamicsModel

using `arcs::common_interface::DynamicsModel`

Initial value:

`std::tuple<std::vector<double>, std::vector<double>, std::vector<double>`

Definition at line 805 of file `type_def.h`.

10.2.1.6 ForceControlPtr

using [arcs::common_interface::ForceControlPtr](#) = std::shared_ptr<[ForceControl](#)>

Definition at line 2618 of file [force_control.h](#).

10.2.1.7 ForceSensorCalibResult

using [arcs::common_interface::ForceSensorCalibResult](#)

Initial value:

```
std::tuple<std::vector<double>, std::vector<double>, double,  
          std::vector<double>
```

Definition at line 795 of file [type_def.h](#).

10.2.1.8 ForceSensorCalibResultWithError

using [arcs::common_interface::ForceSensorCalibResultWithError](#)

Initial value:

```
std::tuple<std::vector<double>, std::vector<double>, double,  
          std::vector<double>, double>
```

Definition at line 800 of file [type_def.h](#).

10.2.1.9 GripperInterfacePtr

using [arcs::common_interface::GripperInterfacePtr](#) = std::shared_ptr<[GripperInterface](#)>

Definition at line 287 of file [gripper_interface.h](#).

10.2.1.10 GripperStatusVector

using [arcs::common_interface::GripperStatusVector](#) = std::vector<[GripperStatus](#)>

Definition at line 864 of file [type_def.h](#).

10.2.1.11 IoControlPtr

using [arcs::common_interface::IoControlPtr](#) = std::shared_ptr<[IoControl](#)>

Definition at line 4984 of file [io_control.h](#).

10.2.1.12 MathPtr

using [arcs::common_interface::MathPtr](#) = std::shared_ptr<[Math](#)>

Definition at line 1073 of file [math.h](#).

10.2.1.13 MotionControlPtr

using [arcs::common_interface::MotionControlPtr](#) = std::shared_ptr<[MotionControl](#)>

Definition at line [6564](#) of file [motion_control.h](#).

10.2.1.14 Payload

using [arcs::common_interface::Payload](#)

Initial value:

```
std::tuple<double, std::vector<double>, std::vector<double>,
          std::vector<double>>
```

Definition at line [791](#) of file [type_def.h](#).

10.2.1.15 RegisterControlPtr

using [arcs::common_interface::RegisterControlPtr](#) = std::shared_ptr<[RegisterControl](#)>

Definition at line [3371](#) of file [register_control.h](#).

10.2.1.16 ResultWithErrno

using [arcs::common_interface::ResultWithErrno](#) = std::tuple<std::vector<double>, int>

Definition at line [785](#) of file [type_def.h](#).

10.2.1.17 ResultWithErrno1

using [arcs::common_interface::ResultWithErrno1](#) = std::tuple<std::vector<std::vector<double>>, int>

Definition at line [786](#) of file [type_def.h](#).

10.2.1.18 ResultWithErrno2

using [arcs::common_interface::ResultWithErrno2](#) = std::tuple<std::vector<std::string>, int>

Definition at line [787](#) of file [type_def.h](#).

10.2.1.19 ResultWithErrno3

using [arcs::common_interface::ResultWithErrno3](#) = std::tuple<int, int>

Definition at line [788](#) of file [type_def.h](#).

10.2.1.20 RobotAlgorithmPtr

using [arcs::common_interface::RobotAlgorithmPtr](#) = std::shared_ptr<[RobotAlgorithm](#)>

Definition at line [1693](#) of file [robot_algorithm.h](#).

10.2.1.21 RobotConfigPtr

using [arcs::common_interface::RobotConfigPtr](#) = std::shared_ptr<[RobotConfig](#)>

Definition at line [3808](#) of file [robot_config.h](#).

10.2.1.22 RobotInterfacePtr

using [arcs::common_interface::RobotInterfacePtr](#) = std::shared_ptr<[RobotInterface](#)>

Definition at line [391](#) of file [robot_interface.h](#).

10.2.1.23 RobotManagePtr

using [arcs::common_interface::RobotManagePtr](#) = std::shared_ptr<[RobotManage](#)>

Definition at line [1917](#) of file [robot_manage.h](#).

10.2.1.24 RobotMsgVector

using [arcs::common_interface::RobotMsgVector](#) = std::vector<[RobotMsg](#)>

Definition at line [845](#) of file [type_def.h](#).

10.2.1.25 RobotStatePtr

using [arcs::common_interface::RobotStatePtr](#) = std::shared_ptr<[RobotState](#)>

Definition at line [3069](#) of file [robot_state.h](#).

10.2.1.26 RuntimeMachinePtr

using [arcs::common_interface::RuntimeMachinePtr](#) = std::shared_ptr<[RuntimeMachine](#)>

Definition at line [1620](#) of file [runtime_machine.h](#).

10.2.1.27 SerialPtr

using [arcs::common_interface::SerialPtr](#) = std::shared_ptr<[Serial](#)>

Definition at line [407](#) of file [serial.h](#).

10.2.1.28 SocketPtr

using [arcs::common_interface::SocketPtr](#) = std::shared_ptr<[Socket](#)>

Definition at line 642 of file [socket.h](#).

10.2.1.29 Sphere

using [arcs::common_interface::Sphere](#) = std::vector<double>

Definition at line 826 of file [type_def.h](#).

10.2.1.30 SyncMovePtr

using [arcs::common_interface::SyncMovePtr](#) = std::shared_ptr<[SyncMove](#)>

Definition at line 1011 of file [sync_move.h](#).

10.2.1.31 SystemInfoPtr

using [arcs::common_interface::SystemInfoPtr](#) = std::shared_ptr<[SystemInfo](#)>

Definition at line 349 of file [system_info.h](#).

10.2.1.32 TaskSet

typedef std::unordered_set<std::string> [arcs::common_interface::TaskSet](#)

Definition at line 36 of file [sync_move.h](#).

10.2.1.33 TracePtr

using [arcs::common_interface::TracePtr](#) = std::shared_ptr<[Trace](#)>

Definition at line 232 of file [trace.h](#).

10.2.1.34 Vector3d

using [arcs::common_interface::Vector3d](#) = std::array<double, 3>

Definition at line 31 of file [type_def.h](#).

10.2.1.35 Vector3f

using [arcs::common_interface::Vector3f](#) = std::array<float, 3>

Definition at line 33 of file [type_def.h](#).

10.2.1.36 Vector4d

using [arcs::common_interface::Vector4d](#) = std::array<double, 4>

Definition at line 32 of file [type_def.h](#).

10.2.1.37 Vector4f

using [arcs::common_interface::Vector4f](#) = std::array<float, 4>

Definition at line 34 of file [type_def.h](#).

10.2.1.38 Vector6f

using [arcs::common_interface::Vector6f](#) = std::array<float, 6>

Definition at line 35 of file [type_def.h](#).

10.2.2 Enumeration Type Documentation

10.2.2.1 AuboErrorCodes

enum [arcs::common_interface::AuboErrorCodes](#) : int

Enumerator

ENUM_AuboErrorCodes_DECLARES	
------------------------------	--

Definition at line 547 of file [type_def.h](#).

10.2.2.2 AxisModeType

enum class [arcs::common_interface::AxisModeType](#) : int [strong]

Enumerator

ENUM_AxisModeType_DECLARES	
----------------------------	--

Definition at line 562 of file [type_def.h](#).

10.2.2.3 EnvelopingShape

enum [arcs::common_interface::EnvelopingShape](#) : int

Enumerator

ENUM_EnvelopingShape_DECLARES	
-------------------------------	--

Definition at line 639 of file [type_def.h](#).

10.2.2.4 error_type

enum [arcs::common_interface::error_type](#)

Error type.

Enumerator

parse_error	Parse error.
invalid_request	Invalid request.
method_not_found	Method not found.
invalid_params	Invalid parameters.
internal_error	Internal error.
server_error	Server error.
invalid	Invalid.

Definition at line 879 of file [type_def.h](#).

10.2.2.5 ExceptionCode

enum [arcs::common_interface::ExceptionCode](#)

Exception code.

Enumerator

EC_DISCONNECTED	Disconnected.
EC_NOT_LOGINED	Not logged in.
EC_INVAL_SOCKET	Invalid socket.
EC_REQUEST_BUSY	Request busy.
EC_SEND_FAILED	Send failed.
EC_RECV_TIMEOUT	Receive timeout.
EC_RECV_ERROR	Receive error.

EC_PARSE_ERROR	Parse error.
EC_INVALID_REQUEST	Invalid request.
EC_METHOD_NOT_FOUND	Method not found.
EC_INVALID_PARAMS	Invalid parameters.
EC_INTERNAL_ERROR	Internal error.
EC_SERVER_ERROR	Server error.
EC_INVALID	Invalid.

Definition at line 891 of file [type_def.h](#).

10.2.2.6 ForceControlState

enum class [arcs::common_interface::ForceControlState](#) [strong]

Enumerator

Stopped	
Starting	
Stropping	
Running	

Definition at line 702 of file [type_def.h](#).

10.2.2.7 JointServoModeType

enum class [arcs::common_interface::JointServoModeType](#) : int [strong]

Joint Servo Mode.

Enumerator

ENUM_JointServoModeType_DECLARES	
----------------------------------	--

Definition at line 595 of file [type_def.h](#).

10.2.2.8 JointStateType

enum class [arcs::common_interface::JointStateType](#) : int [strong]

Generated by Doxygen

Joint State.

Enumerator

ENUM_JointStateType_DECLARES	
------------------------------	--

Definition at line 603 of file [type_def.h](#).

10.2.2.9 OperationalModeType

```
enum class arcs::common_interface::OperationalModeType : int [strong]
```

Operational Mode.

Enumerator

ENUM_OperationalModeType_DECLARES	
-----------------------------------	--

Definition at line 579 of file [type_def.h](#).

10.2.2.10 PathBufferType

```
enum arcs::common_interface::PathBufferType
```

pathBuffer Type

The robot movements are programmed as pose-to-pose movements, that is move from the current position to a new position. The path between these two positions is then automatically calculated by the robot.

Enumerator

PathBuffer_TOPPRA	1: toppra time optimal path planning
PathBuffer_CubicSpline	2: cubic_spline (recorded trajectory)
PathBuffer_JointSpline	3: Joint B-spline interpolation, at least three points Deprecated, use 5 instead, currently it is joint space CUBIC_SPLINE
PathBuffer_JointSplineC	4: Joint B-spline interpolation, at least three points, but the input is Cartesian space pose 6 CUBIC_SPLINE 5: Joint B-spline interpolation, at least three points
PathBuffer_JointBSpline	
PathBuffer_JointBSplineC	6: Joint B-spline interpolation, at least three points, but the input is Cartesian space pose

Definition at line 34 of file [motion_control.h](#).

10.2.2.11 PayloadIdentifyMoveAxis

Enumerator

ENUM_PayloadIdentifyMoveAxis_DECLARES	
---------------------------------------	--

Definition at line 644 of file [type_def.h](#).

10.2.2.12 RefFrameType

enum class [arcs::common_interface::RefFrameType](#) [strong]

Enumerator

None	
Tool	Tool coordinate system.
Path	Trajectory coordinate system.
Base	Base coordinate system.

Definition at line 710 of file [type_def.h](#).

10.2.2.13 RobotControlModeType

enum class [arcs::common_interface::RobotControlModeType](#) : int [strong]

Robot Control Mode.

Enumerator

ENUM_RobotControlModeType_DECLARES	
------------------------------------	--

Definition at line 587 of file [type_def.h](#).

10.2.2.14 RobotEmergencyStopType

enum [arcs::common_interface::RobotEmergencyStopType](#) : int

Enumerator

ENUM_RobotEmergencyStopType_DECLARES	
--------------------------------------	--

Definition at line 659 of file [type_def.h](#).

10.2.2.15 RobotModeType

enum class [arcs::common_interface::RobotModeType](#) : int [strong]

Enumerator

ENUM_RobotModeType_DECLARES	
-----------------------------	--

Definition at line 557 of file [type_def.h](#).

10.2.2.16 RuntimeState

enum class [arcs::common_interface::RuntimeState](#) : int [strong]

Enumerator

ENUM_RuntimeState_DECLARES	
----------------------------	--

Definition at line 552 of file [type_def.h](#).

10.2.2.17 SafeguedStopType

enum [arcs::common_interface::SafeguedStopType](#) : int

Enumerator

ENUM_SafeguedStopType_DECLARES	
--------------------------------	--

Definition at line 654 of file [type_def.h](#).

10.2.2.18 SafetyInputAction

enum class [arcs::common_interface::SafetyInputAction](#) : int [strong]

Enumerator

ENUM_SafetyInputAction_DECLARES	
---------------------------------	--

Definition at line 624 of file [type_def.h](#).

10.2.2.19 SafetyModeType

enum class [arcs::common_interface::SafetyModeType](#) : int [strong]

Safety Mode.

Enumerator

ENUM_SafetyModeType_DECLARES	
------------------------------	--

Definition at line 571 of file [type_def.h](#).

10.2.2.20 SafetyOutputRunState

enum class [arcs::common_interface::SafetyOutputRunState](#) : int [strong]

Enumerator

ENUM_SafetyOutputRunState_DECLARES	
------------------------------------	--

Definition at line 629 of file [type_def.h](#).

10.2.2.21 StandardInputAction

enum class [arcs::common_interface::StandardInputAction](#) : int [strong]

The [StandardInputAction](#) enum.

Enumerator

ENUM_StandardInputAction_DECLARES	
-----------------------------------	--

Enumerator

ENUM_StandardOutputRunState_DECLARES	
--------------------------------------	--

Definition at line 611 of file [type_def.h](#).

10.2.2.23 TaskFrameType

enum [arcs::common_interface::TaskFrameType](#)

Enumerator

ENUM_TaskFrameType_DECLARES	
-----------------------------	--

Definition at line 634 of file [type_def.h](#).

10.2.2.24 TraceLevel

enum [arcs::common_interface::TraceLevel](#)

Enumerator

ENUM_TraceLevel_DECLARES	
--------------------------	--

Definition at line 649 of file [type_def.h](#).

10.2.3 Function Documentation

10.2.3.1 operator<<() [1/7]

```
std::ostream & arcs::common_interface::operator<< (
    std::ostream & os,
    CircleParameters p) [inline]
```

Definition at line 733 of file [type_def.h](#).

10.2.3.2 operator<<() [2/7]

```
std::ostream & arcs::common_interface::operator<< (
    std::ostream & os,
    const RobotSafetyParameterRange & vd) [inline]
```

Definition at line 180 of file [type_def.h](#).

10.2.3.3 operator<<() [3/7]

```
std::ostream & arcs::common_interface::operator<< (
    std::ostream & os,
    const VibrationRecalibrationParameter p) [inline]
```

Definition at line 223 of file [type_def.h](#).

10.2.3.4 operator<<() [4/7]

```
std::ostream & arcs::common_interface::operator<< (
    std::ostream & os,
    Enveloping p) [inline]
```

Definition at line 761 of file [type_def.h](#).

10.2.3.5 operator<<() [5/7]

```
std::ostream & arcs::common_interface::operator<< (
    std::ostream & os,
    SpiralParameters p) [inline]
```

Definition at line 747 of file [type_def.h](#).

10.2.3.6 operator<<() [6/7]

```
std::ostream & arcs::common_interface::operator<< (
    std::ostream & os,
    TrajConfig p) [inline]
```

Definition at line 779 of file [type_def.h](#).

10.2.3.7 operator<<() [7/7]

```
std::ostream & arcs::common_interface::operator<< (
    std::ostream & os,
    WObjectData p) [inline]
```

Definition at line 208 of file [type_def.h](#).

10.2.3.8 returnValue2Str()

```
const char * arcs::common_interface::returnValue2Str (
    int retval) [inline]
```

Definition at line 945 of file [type_def.h](#).

References [ENUM_AuboErrorCodes_DECLARES](#).

10.3 arcs::error_stack Namespace Reference

Enumerations

- enum [ErrorCodes](#) { [ARCS_ERROR_CODES](#) }

Functions

- constexpr int ARCS_ABI_EXPORT [codeCompose](#) (int aa, int bb, int cccc)
- constexpr int ARCS_ABI_EXPORT [mod](#) (int x)
- int [str2ErrorCode](#) (const char *err_code_name)
- const char * [errorCode2Str](#) (int err_code)
- std::ostream & [dump](#) (std::ostream &os)

10.3.1 Enumeration Type Documentation

10.3.1.1 ErrorCodes

enum [arcs::error_stack::ErrorCodes](#)

Enumerator

ARCS_ERROR_CODES	
----------------------------------	--

Definition at line [52](#) of file [error_stack.h](#).

10.3.2 Function Documentation

10.3.2.1 codeCompose()

```
int ARCS_ABI_EXPORT arcs::error_stack::codeCompose (
    int aa,
    int bb,
    int cccc)  [constexpr]
```

Definition at line [27](#) of file [error_stack.h](#).

10.3.2.2 dump()

```
std::ostream & arcs::error_stack::dump (
    std::ostream & os)  [inline]
```

Definition at line [99](#) of file [error_stack.h](#).

References [ARCS_ERROR_CODES](#).

10.3.2.3 errorCode2Str()

```
const char * arcs::error_stack::errorCode2Str (  
    int err_code)  [inline]
```

Definition at line 69 of file [error_stack.h](#).

References [ARCS_ERROR_CODES](#).

10.3.2.4 mod()

```
int ARCS_ABI_EXPORT arcs::error_stack::mod (  
    int x)  [constexpr]
```

Definition at line 32 of file [error_stack.h](#).

10.3.2.5 str2ErrorCode()

```
int arcs::error_stack::str2ErrorCode (  
    const char * err_code_name)  [inline]
```

Definition at line 59 of file [error_stack.h](#).

References [ARCS_ERROR_CODES](#).

Chapter 11

Class Documentation

11.1 arcs::common_interface::AuboApi Class Reference

```
#include <aubo_api.h>
```

Public Member Functions

- [AuboApi](#) ()
- virtual [~AuboApi](#) ()
- [MathPtr](#) [getMath](#) ()
 [Math](#) ()
 Get pure mathematic related API
- [SystemInfoPtr](#) [getSystemInfo](#) ()
 [SystemInfo](#) ()
 Get system info
- [RuntimeMachinePtr](#) [getRuntimeMachine](#) ()
 [RuntimeMachine](#) ()
 Get runtime api
- [RegisterControlPtr](#) [getRegisterControl](#) ()
 [RegisterControl](#) ()
 External registers api
- [std::vector< std::string >](#) [getRobotNames](#) ()
 Get robot list
- [RobotInterfacePtr](#) [getRobotInterface](#) (const [std::string](#) &name)
 [RobotInterface](#)()
 Get [RobotInterfacePtr](#) based on name
- [std::vector< std::string >](#) [getAxisNames](#) ()
 Get external axis list.
- [AxisInterfacePtr](#) [getAxisInterface](#) (const [std::string](#) &name)
 [AxisInterface](#) ()
 Get external axis interface
- [SocketPtr](#) [getSocket](#) ()
 [Socket](#) (socket)
 Get socket
- [SerialPtr](#) [getSerial](#) ()
 [Serial](#) ()
- [SyncMovePtr](#) [getSyncMove](#) (const [std::string](#) &name)

- [SyncMove \(\)](#) Get synchronous move interface
- [TracePtr getTrace \(const std::string &name\)](#)
[Trace \(\)](#) Get alert interface
- [GripperInterfacePtr getGripperInterface \(\)](#)
[GripperInterface \(\)](#) Get gripper interface

Protected Attributes

- void * [d_](#) { nullptr }

11.1.1 Detailed Description

Definition at line 27 of file [aubo_api.h](#).

11.1.2 Constructor & Destructor Documentation

11.1.2.1 AuboApi()

`arcs::common_interface::AuboApi::AuboApi ()`

11.1.2.2 ~AuboApi()

`virtual arcs::common_interface::AuboApi::~AuboApi ()` [virtual]

11.1.3 Member Data Documentation

11.1.3.1 d_

`void* arcs::common_interface::AuboApi::d_ { nullptr }` [protected]

Definition at line 388 of file [aubo_api.h](#).

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

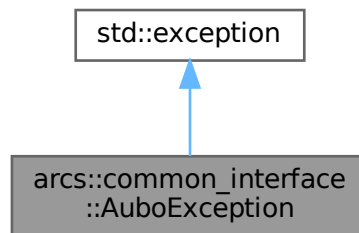
- include/aubo/[aubo_api.h](#)

11.2 arcs::common_interface::AboException Class Reference

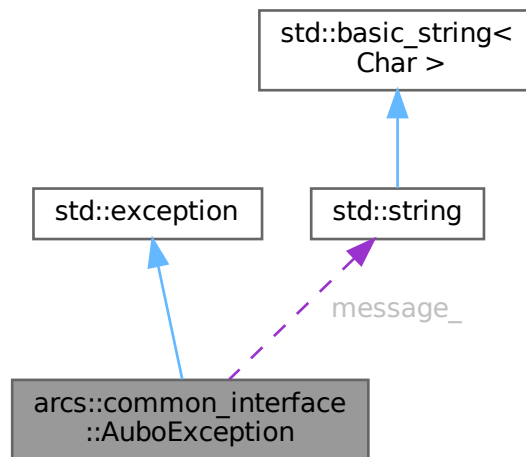
Custom exception class [AboException](#).

```
#include <type_def.h>
```

Inheritance diagram for arcs::common_interface::AboException:



Collaboration diagram for arcs::common_interface::AboException:



Public Member Functions

- [AboException](#) (int [code](#), const std::string &prefix, const std::string &message) noexcept
- [AboException](#) (int [code](#), const std::string &message) noexcept
- [error_type](#) type () const
- int [code](#) () const
- const char * [what](#) () const noexcept override

Private Attributes

- int [code__](#)
Exception code.
- std::string [message__](#)

11.2.1 Detailed Description

Custom exception class [AuboException](#).

Definition at line 911 of file [type_def.h](#).

11.2.2 Constructor & Destructor Documentation

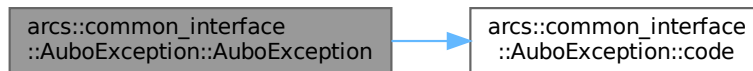
11.2.2.1 [AuboException\(\)](#) [1/2]

```
arcs::common_interface::AuboException::AuboException (
    int code,
    const std::string & prefix,
    const std::string & message)  [inline], [noexcept]
```

Definition at line 914 of file [type_def.h](#).

References [code\(\)](#).

Here is the call graph for this function:



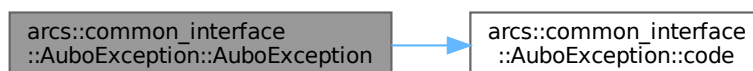
11.2.2.2 [AuboException\(\)](#) [2/2]

```
arcs::common_interface::AuboException::AuboException (
    int code,
    const std::string & message)  [inline], [noexcept]
```

Definition at line 920 of file [type_def.h](#).

References [code\(\)](#).

Here is the call graph for this function:



11.2.3 Member Function Documentation

11.2.3.1 code()

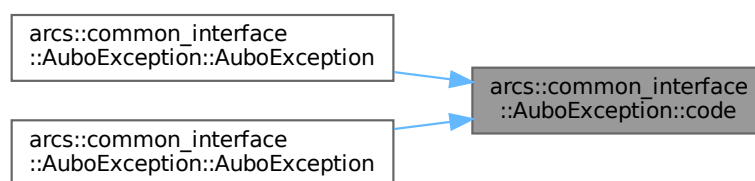
int arcs::common_interface::AuboException::code () const [inline]

Definition at line 937 of file [type_def.h](#).

References [code_](#).

Referenced by [AuboException\(\)](#), and [AuboException\(\)](#).

Here is the caller graph for this function:



11.2.3.2 type()

[error_type](#) arcs::common_interface::AuboException::type () const [inline]

Definition at line 925 of file [type_def.h](#).

References [code_](#), [arcs::common_interface::invalid](#), [arcs::common_interface::parse_error](#), and [arcs::common_interface::server_error](#).

11.2.3.3 what()

const char * arcs::common_interface::AuboException::what () const [inline], [override], [noexcept]

Definition at line 938 of file [type_def.h](#).

References [message_](#).

11.2.4 Member Data Documentation

11.2.4.1 code__

int arcs::common_interface::AuboException::code__ [private]

Exception code.

Definition at line 941 of file [type_def.h](#).

Referenced by [code\(\)](#), and [type\(\)](#).

11.2.4.2 message__

std::string arcs::common_interface::AuboException::message__ [private]

Definition at line 942 of file [type_def.h](#).

Referenced by [what\(\)](#).

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

- include/aubo/[type_def.h](#)

11.3 arcs::common_interface::AxisInterface Class Reference

```
#include <axis_interface.h>
```

Public Member Functions

- [AxisInterface](#) ()
- virtual [~AxisInterface](#) ()
- int [poweronExtAxis](#) ()
Power on.
- int [poweroffExtAxis](#) ()
Power off.
- int [enableExtAxis](#) ()
Enable.
- int [setExtAxisMountingPose](#) (const std::vector< double > &pose)
Set mounting pose of external axis (wrt world frame).
- int [moveExtJoint](#) (double pos, double v, double a, double duration)
move to pos, rotation or linear
- int [speedExtJoint](#) (double v, double a, double duration)
Set target speed, acceleration and duration
- int [stopExtJoint](#) (double a)
stop ext joint
- int [getExtAxisType](#) ()
Get external axis type: 0 for rotation, 1 for linear.
- [AxisModeType](#) [getAxisModeType](#) ()
Get external axis status
- std::vector< double > [getExtAxisMountingPose](#) ()
Get external axis mounting pose
- std::vector< double > [getExtAxisPose](#) ()
Get pose wrt mounting coordinate system, axis can be positioner or linear rail
- double [getExtAxisPosition](#) ()
Get external axis position
- double [getExtAxisVelocity](#) ()
Get external axis speed
- double [getExtAxisAcceleration](#) ()
Get external axis acceleration
- double [getExtAxisCurrent](#) ()

- Get external axis current
- double [getExtAxisTemperature](#) ()
 - Get external axis temperature
- double [getExtAxisBusVoltage](#) ()
 - Get external axis voltage
- double [getExtAxisBusCurrent](#) ()
 - Get external axis current
- double [getExtAxisMaxPosition](#) ()
 - Get external axis max position
- double [getExtMinPosition](#) ()
 - Get external axis min position
- double [getExtAxisMaxVelocity](#) ()
 - Get external axis max speed
- double [getExtAxisMaxAcceleration](#) ()
 - Get external axis max acceleration
- int [followAnotherAxis](#) (const std::string &target_name, double phase, double err)
 - Follow motion of another external axis not to be used during motion
- int [stopFollowAnotherAxis](#) ()
 - stopFollowAnotherAxis(not to be used during motion)
- int [getErrorCode](#) ()
 - Get external axis error code
- int [clearAxisError](#) ()
 - Reset axis error

Protected Attributes

- void * [d__](#)

11.3.1 Detailed Description

Definition at line 18 of file [axis_interface.h](#).

11.3.2 Constructor & Destructor Documentation

11.3.2.1 AxisInterface()

arcs::common_interface::AxisInterface::AxisInterface ()

11.3.2.2 ~AxisInterface()

virtual arcs::common_interface::AxisInterface::~AxisInterface () [virtual]

11.3.3 Member Data Documentation

11.3.3.1 d__

void* arcs::common_interface::AxisInterface::d__ [protected]

Definition at line 415 of file [axis_interface.h](#).

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

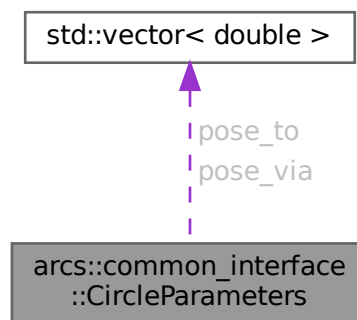
- include/aubo/[axis_interface.h](#)

11.4 arcs::common_interface::CircleParameters Struct Reference

Circular motion parameters definition.

```
#include <type_def.h>
```

Collaboration diagram for arcs::common_interface::CircleParameters:



Public Attributes

- std::vector< double > [pose_via](#)
Pose of the intermediate point in circular motion.
- std::vector< double > [pose_to](#)
Pose of the end point in circular motion.
- double [a](#)
Acceleration, unit: m/s².
- double [v](#)
Speed, unit: m/s.
- double [blend_radius](#)
Blending radius, unit: m.
- double [duration](#)
Running time, unit: s.
- double [helix](#)
- double [spiral](#)
- double [direction](#)
- int [loop_times](#)
Currently not supported.

11.4.1 Detailed Description

Circular motion parameters definition.

Definition at line 719 of file [type_def.h](#).

11.4.2 Member Data Documentation

11.4.2.1 a

double arcs::common_interface::CircleParameters::a

Acceleration, unit: m/s^2 .

Definition at line 723 of file [type_def.h](#).

11.4.2.2 blend_radius

double arcs::common_interface::CircleParameters::blend_radius

Blending radius, unit: m.

Definition at line 725 of file [type_def.h](#).

11.4.2.3 direction

double arcs::common_interface::CircleParameters::direction

Definition at line 729 of file [type_def.h](#).

11.4.2.4 duration

double arcs::common_interface::CircleParameters::duration

Running time, unit: s.

Definition at line 726 of file [type_def.h](#).

11.4.2.5 helix

double arcs::common_interface::CircleParameters::helix

Definition at line 727 of file [type_def.h](#).

11.4.2.6 loop_times

`int arcs::common_interface::CircleParameters::loop_times`

Currently not supported.

Definition at line 730 of file [type_def.h](#).

11.4.2.7 pose_to

`std::vector<double> arcs::common_interface::CircleParameters::pose_to`

Pose of the end point in circular motion.

Definition at line 722 of file [type_def.h](#).

11.4.2.8 pose_via

`std::vector<double> arcs::common_interface::CircleParameters::pose_via`

Pose of the intermediate point in circular motion.

Definition at line 721 of file [type_def.h](#).

11.4.2.9 spiral

`double arcs::common_interface::CircleParameters::spiral`

Definition at line 728 of file [type_def.h](#).

11.4.2.10 v

`double arcs::common_interface::CircleParameters::v`

Speed, unit: m/s.

Definition at line 724 of file [type_def.h](#).

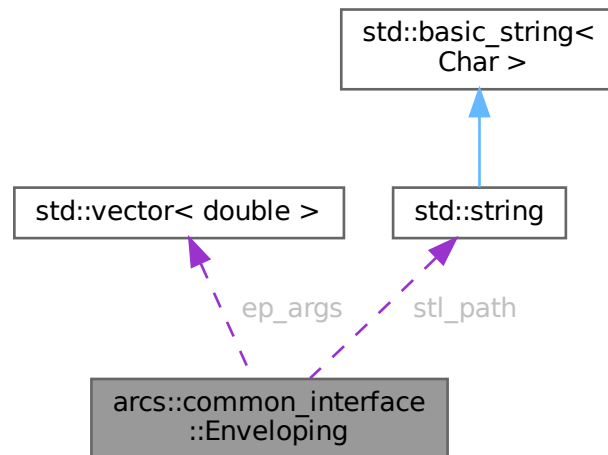
The documentation for this struct was generated from the following file:

- [include/aubo/type_def.h](#)

11.5 arcs::common_interface::Enveloping Struct Reference

```
#include <type_def.h>
```

Collaboration diagram for arcs::common_interface::Enveloping:



Public Attributes

- [EnvelopingShape](#) `shape`
- `std::vector< double >` `ep_args`
- `std::string` `stl_path`

11.5.1 Detailed Description

Definition at line 752 of file [type_def.h](#).

11.5.2 Member Data Documentation

11.5.2.1 ep_args

`std::vector<double>` `arcs::common_interface::Enveloping::ep_args`

Definition at line 755 of file [type_def.h](#).

11.5.2.2 shape

[EnvelopingShape](#) `arcs::common_interface::Enveloping::shape`

Definition at line 754 of file [type_def.h](#).

11.5.2.3 `stl_path`

`std::string arcs::common_interface::Enveloping::stl_path`

Definition at line 758 of file [type_def.h](#).

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

- [include/aubo/type_def.h](#)

11.6 `arcs::common_interface::ForceControl` Class Reference

```
#include <force_control.h>
```

Public Member Functions

- [ForceControl](#) ()
- virtual [~ForceControl](#) ()
- int [fcEnable](#) ()
Start force control
- int [fcDisable](#) ()
End force control
- bool [isFcEnabled](#) ()
Check if force control is enabled
- int [setTargetForce](#) (const std::vector< double > &feature, const std::vector< bool > &compliance, const std::vector< double > &wrench, const std::vector< double > &limits, [TaskFrameType](#) type=[TaskFrameType::FRAME_FORCE](#))
Set force control reference (target) value
- int [setDynamicModel1](#) (const std::vector< double > &env_stiff, const std::vector< double > &damp_scale, const std::vector< double > &stiff_scale)
Set force control dynamics model
- [DynamicsModel fcCalDynamicModel](#) (const std::vector< double > &env_stiff, const std::vector< double > &damp_scale, const std::vector< double > &stiff_scale)
Calculate force control dynamics model
- int [setDynamicModelSearch](#) (const std::vector< double > &damp_scale, const std::vector< double > &stiff_scale)
Set force control dynamics model for hole searching scenario
- int [setDynamicModelInsert](#) (const std::vector< double > &damp_scale, const std::vector< double > &stiff_scale)
Set force control dynamics model for insertion/extraction scenario
- int [setDynamicModelContact](#) (const std::vector< double > &env_stiff, const std::vector< double > &damp_scale, const std::vector< double > &stiff_scale)
Set force control dynamics model for contact scenario
- int [setDynamicModel](#) (const std::vector< double > &m, const std::vector< double > &d, const std::vector< double > &k)
Set force control dynamics model
- int [fcSetSensorThresholds](#) (const std::vector< double > &thresholds)
Set force control thresholds
- int [fcSetSensorLimits](#) (const std::vector< double > &limits)
Set force control maximum force limits

- `std::vector< double > getFcSensorThresholds ()`
Get force control thresholds
- `std::vector< double > getFcSensorLimits ()`
Get maximum force limits
- `DynamicsModel getDynamicModel ()`
Get force control dynamics model
- `int setCondForce (const std::vector< double > &min, const std::vector< double > &max, bool outside, double timeout)`
Set force control termination condition: Force.
- `int setCondOrient (const std::vector< double > &frame, double max_angle, double max_rot, bool outside, double timeout)`
setCondOrient is used to set up an end condition for the tool orientation.
- `int setCondPlane (const std::vector< double > &plane, double timeout)`
Specify a valid force control plane, x-y plane, z direction is valid
- `int setCondCylinder (const std::vector< double > &axis, double radius, bool outside, double timeout)`
Specify a valid force control cylinder by providing the central axis and cylinder radius, and specify whether the inside or outside of the cylinder is valid.
- `int setCondSphere (const std::vector< double > ¢er, double radius, bool outside, double timeout)`
Specify a valid force control sphere by providing the center and radius, and specify whether the inside or outside of the sphere is valid.
- `int setCondTcpSpeed (const std::vector< double > &min, const std::vector< double > &max, bool outside, double timeout)`
setCondTcpSpeed is used to setup an end condition for the TCP speed.
- `int setCondDistance (double distance, double timeout)`
Force control termination condition - distance
- `int setCondAdvanced (const std::string &type, const std::vector< double > &args, double timeout)`
Advanced force control termination condition
- `int setCondActive ()`
Activate force control termination condition
- `bool isCondFullfiled ()`
Check if the force control termination condition has been fulfilled
- `int setSupvForce (const std::vector< double > &min, const std::vector< double > &max)`
setSupvForce is used to set up force supervision in Force Control.
- `int setSupvOrient (const std::vector< double > &frame, double max_angle, double max_rot, bool outside)`
setSupvOrient is used to set up an supervision for the tool orientation.
- `int setSupvPosBox (const std::vector< double > &frame, const Box &box)`
setSupvPosBox is used to set up position supervision in Force Control.
- `int setSupvPosCylinder (const std::vector< double > &frame, const Cylinder &cylinder)`
- `int setSupvPosSphere (const std::vector< double > &frame, const Sphere &sphere)`
- `int setSupvReoriSpeed (const std::vector< double > &speed_limit, bool outside, double timeout)`
setSupvReoriSpeed is used to set up reorientation speed supervision in Force Control.
- `int setSupvTcpSpeed (const std::vector< double > &speed_limit, bool outside, double timeout)`
setSupvTcpSpeed is used to set up TCP speed supervision in Force Control.
- `int setLpFilter (const std::vector< double > &cutoff_freq)`
Set low-pass filter
- `int resetLpFilter ()`
Reset low-pass filter
- `int speedChangeEnable (double ref_force)`

The speedChangeEnable is used to activate FC SpeedChange function with desired reference and recover behavior.

- int [speedChangeDisable](#) ()
Deactivate FC SpeedChange function.
- int [speedChangeTune](#) (int speed_levels, double speed_ratio_min)
speedChangeTune is used to set FC SpeedChange system parameters to a new value.
- int [setDamping](#) (const std::vector< double > &damping, double ramp_time)
setDamping is used to tune the damping in the force control coordinate systems.
- int [resetDamping](#) ()
Reset damping parameters
- int [softFloatEnable](#) ()
Enable soft float function.
- int [softFloatDisable](#) ()
Disable soft float function.
- bool [isSoftFloatEnabled](#) ()
Returns whether soft float is enabled
- int [setSoftFloatParams](#) (bool joint_space, const std::vector< bool > &select, const std::vector< double > &stiff_percent, const std::vector< double > &stiff_damp_ratio, const std::vector< double > &force_threshold, const std::vector< double > &force_limit)
Set soft float parameters
- int [toolContact](#) (const std::vector< bool > &direction)
Detect contact between the tool and external objects
- std::vector< double > [getActualJointPositionsHistory](#) (int steps)

Protected Attributes

- void * [d_](#) { nullptr }

11.6.1 Detailed Description

Definition at line 94 of file [force_control.h](#).

11.6.2 Constructor & Destructor Documentation

11.6.2.1 ForceControl()

arcs::common_interface::ForceControl::ForceControl ()

11.6.2.2 ~ForceControl()

virtual arcs::common_interface::ForceControl::~~ForceControl () [virtual]

11.6.3 Member Data Documentation

11.6.3.1 d_

void* arcs::common_interface::ForceControl::d_ { nullptr } [protected]

Definition at line 2616 of file [force_control.h](#).

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

- [include/aubo/robot/force_control.h](#)

11.7 arcs::common_interface::GripperInterface Class Reference

```
#include <gripper_interface.h>
```

Public Member Functions

- [GripperInterface](#) ()
- virtual [~GripperInterface](#) ()
- std::vector< std::string > [gripperGetSupportedModels](#) ()
- [ResultWithErrno2 gripperScanDevices](#) (const std::string &model, const std::string &device_name)
- std::vector< std::string > [gripperGetNames](#) ()
- int [gripperAdd](#) (const std::string &name, const std::string &model)
- int [gripperDelete](#) (const std::string &name)
- int [gripperRename](#) (const std::string &name, const std::string &new_name)
- int [gripperConnect](#) (const std::string &name, const std::string &device_name)
- int [gripperDisconnect](#) (const std::string &name)
- bool [gripperIsConnected](#) (const std::string &name)
- int [gripperSetWorkMode](#) (const std::string &name, int work_mode)
- int [gripperGetWorkMode](#) (const std::string &name)
- int [gripperSetMountPose](#) (const std::string &name, const std::vector< double > &pose, bool enable_collision)
- std::vector< double > [gripperGetMountPose](#) (const std::string &name)
- int [gripperEnable](#) (const std::string &name, bool enable)
- bool [gripperIsEnabled](#) (const std::string &name)
- int [gripperSetPosition](#) (const std::string &name, const double position)
- int [gripperSetVelocity](#) (const std::string &name, const double velocity_percent)
- int [gripperSetForce](#) (const std::string &name, const double force)
- int [gripperSetAngle](#) (const std::string &name, const double angle)
- int [gripperSetRVelocity](#) (const std::string &name, const double r_velocity_percent)
- int [gripperSetTorque](#) (const std::string &name, const double torque_percent)
- int [gripperMove](#) (const std::string &name)
- int [gripperStop](#) (const std::string &name)
- std::string [gripperGetHardwareVersion](#) (const std::string &name)
- std::string [gripperGetSoftwareVersion](#) (const std::string &name)
- double [gripperGetPosition](#) (const std::string &name)
- double [gripperGetVelocity](#) (const std::string &name)
- double [gripperGetForce](#) (const std::string &name)
- double [gripperGetAngle](#) (const std::string &name)
- double [gripperGetRVelocity](#) (const std::string &name)
- double [gripperGetTorque](#) (const std::string &name)
- bool [gripperGetObjectDetection](#) (const std::string &name)
- bool [gripperGetMotionState](#) (const std::string &name)
- double [gripperGetVoltage](#) (const std::string &name)
- double [gripperGetTemperature](#) (const std::string &name)
- int [gripperResetSlaveId](#) (const std::string &name, const int slave_id)
- int [gripperGetStatusCode](#) (const std::string &name)

Protected Attributes

- void * [d__](#)

11.7.1 Detailed Description

Definition at line 17 of file [gripper_interface.h](#).

11.7.2 Constructor & Destructor Documentation

11.7.2.1 GripperInterface()

```
arcs::common_interface::GripperInterface::GripperInterface ()
```

11.7.2.2 ~GripperInterface()

```
virtual arcs::common_interface::GripperInterface::~~GripperInterface () [virtual]
```

11.7.3 Member Function Documentation

11.7.3.1 gripperGetAngle()

```
double arcs::common_interface::GripperInterface::gripperGetAngle (  
    const std::string & name)
```

11.7.3.2 gripperGetForce()

```
double arcs::common_interface::GripperInterface::gripperGetForce (  
    const std::string & name)
```

11.7.3.3 gripperGetMotionState()

```
bool arcs::common_interface::GripperInterface::gripperGetMotionState (  
    const std::string & name)
```

11.7.3.4 gripperGetMountPose()

```
std::vector< double > arcs::common_interface::GripperInterface::gripperGetMountPose (  
    const std::string & name)
```


Parameters

name	
------	--

Returns

11.7.3.5 gripperGetObjectDetection()

```
bool arcs::common_interface::GripperInterface::gripperGetObjectDetection (  
    const std::string & name)
```

11.7.3.6 gripperGetPosition()

```
double arcs::common_interface::GripperInterface::gripperGetPosition (  
    const std::string & name)
```

11.7.3.7 gripperGetRVelocity()

```
double arcs::common_interface::GripperInterface::gripperGetRVelocity (  
    const std::string & name)
```

11.7.3.8 gripperGetSoftwareVersion()

```
std::string arcs::common_interface::GripperInterface::gripperGetSoftwareVersion (  
    const std::string & name)
```

11.7.3.9 gripperGetTemperature()

```
double arcs::common_interface::GripperInterface::gripperGetTemperature (  
    const std::string & name)
```

11.7.3.10 gripperGetTorque()

```
double arcs::common_interface::GripperInterface::gripperGetTorque (  
    const std::string & name)
```

11.7.3.11 gripperGetVelocity()

```
double arcs::common_interface::GripperInterface::gripperGetVelocity (  
    const std::string & name)
```

11.7.3.12 gripperGetVoltage()

```
double arcs::common_interface::GripperInterface::gripperGetVoltage (  
    const std::string & name)
```

11.7.3.13 gripperSetAngle()

```
int arcs::common_interface::GripperInterface::gripperSetAngle (  
    const std::string & name,  
    const double angle)
```

11.7.3.14 gripperSetForce()

```
int arcs::common_interface::GripperInterface::gripperSetForce (  
    const std::string & name,  
    const double force)
```

11.7.3.15 gripperSetRVelocity()

```
int arcs::common_interface::GripperInterface::gripperSetRVelocity (  
    const std::string & name,  
    const double r_velocity_percent)
```

11.7.3.16 gripperSetTorque()

```
int arcs::common_interface::GripperInterface::gripperSetTorque (  
    const std::string & name,  
    const double torque_percent)
```

11.7.3.17 gripperSetVelocity()

```
int arcs::common_interface::GripperInterface::gripperSetVelocity (  
    const std::string & name,  
    const double velocity_percent)
```

11.7.4 Member Data Documentation

11.7.4.1 d_

```
void* arcs::common_interface::GripperInterface::d_ [protected]
```

Definition at line 285 of file [gripper_interface.h](#).

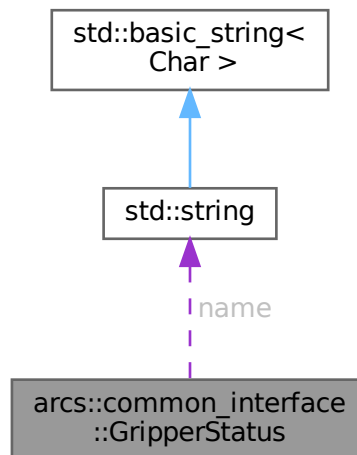
The documentation for this class was generated from the following file:

- [include/aubo/gripper_interface.h](#)

11.8 arcs::common_interface::GripperStatus Struct Reference

#include <type_def.h>

Collaboration diagram for arcs::common_interface::GripperStatus:



Public Attributes

- `std::string` [name](#)
- `bool` [is_connected](#)
- `bool` [is_enabled](#)
- `double` [position](#)
- `double` [velocity](#)
- `double` [force](#)
- `double` [angle](#)
- `double` [r_velocity](#)
- `double` [torque](#)
- `bool` [object_detection](#)
- `bool` [motion_state](#)
- `double` [voltage](#)
- `double` [temperature](#)
- `int` [status_code](#)

11.8.1 Detailed Description

Definition at line 847 of file [type_def.h](#).

11.8.2 Member Data Documentation

11.8.2.1 angle

double arcs::common_interface::GripperStatus::angle

Definition at line 855 of file [type_def.h](#).

11.8.2.2 force

double arcs::common_interface::GripperStatus::force

Definition at line 854 of file [type_def.h](#).

11.8.2.3 is_connected

bool arcs::common_interface::GripperStatus::is_connected

Definition at line 850 of file [type_def.h](#).

11.8.2.4 is_enabled

bool arcs::common_interface::GripperStatus::is_enabled

Definition at line 851 of file [type_def.h](#).

11.8.2.5 motion_state

bool arcs::common_interface::GripperStatus::motion_state

Definition at line 859 of file [type_def.h](#).

11.8.2.6 name

std::string arcs::common_interface::GripperStatus::name

Definition at line 849 of file [type_def.h](#).

11.8.2.7 object_detection

bool arcs::common_interface::GripperStatus::object_detection

Definition at line 858 of file [type_def.h](#).

11.8.2.8 position

double arcs::common_interface::GripperStatus::position

Definition at line 852 of file [type_def.h](#).

11.8.2.9 r_velocity

double arcs::common_interface::GripperStatus::r_velocity

Definition at line 856 of file [type_def.h](#).

11.8.2.10 status_code

int arcs::common_interface::GripperStatus::status_code

Definition at line 862 of file [type_def.h](#).

11.8.2.11 temperature

double arcs::common_interface::GripperStatus::temperature

Definition at line 861 of file [type_def.h](#).

11.8.2.12 torque

double arcs::common_interface::GripperStatus::torque

Definition at line 857 of file [type_def.h](#).

11.8.2.13 velocity

double arcs::common_interface::GripperStatus::velocity

Definition at line 853 of file [type_def.h](#).

11.8.2.14 voltage

double arcs::common_interface::GripperStatus::voltage

Definition at line 860 of file [type_def.h](#).

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

- [include/aubo/type_def.h](#)

11.9 arcs::common_interface::IoControl Class Reference

```
#include <io_control.h>
```

Public Member Functions

- [IoControl](#) ()
- virtual [~IoControl](#) ()
- int [getStandardDigitalInputNum](#) ()
Get the number of standard digital inputs.
- int [getToolDigitalInputNum](#) ()
Get the number of tool digital IOs (including digital inputs and outputs).
- int [getConfigurableDigitalInputNum](#) ()
Get the number of configurable digital inputs.
- int [getStandardDigitalOutputNum](#) ()
Get the number of standard digital outputs.
- int [getToolDigitalOutputNum](#) ()
Get the number of tool digital IOs (including digital inputs and outputs).
- int [setToolIoInput](#) (int index, bool input)
Set the specified tool digital IO as input or output.
- bool [isToolIoInput](#) (int index)
Determine whether the specified tool digital IO is configured as input.
- int [getConfigurableDigitalOutputNum](#) ()
Get the number of configurable digital outputs.
- int [getStandardAnalogInputNum](#) ()
Get the number of standard analog inputs.
- int [getToolAnalogInputNum](#) ()
Get the number of tool analog inputs.
- int [getStandardAnalogOutputNum](#) ()
Get the number of standard analog outputs.
- int [getToolAnalogOutputNum](#) ()
Get the number of tool analog outputs.
- int [setDigitalInputActionDefault](#) ()
Set all digital input actions to no trigger.
- int [setStandardDigitalInputAction](#) (int index, [StandardInputAction](#) action)
Set the trigger action for standard digital input.
- int [setToolDigitalInputAction](#) (int index, [StandardInputAction](#) action)
Set the trigger action for tool digital input.
- int [setConfigurableDigitalInputAction](#) (int index, [StandardInputAction](#) action)
Set the trigger action for configurable digital input.
- [StandardInputAction](#) [getStandardDigitalInputAction](#) (int index)
Get the trigger action for standard digital input.
- [StandardInputAction](#) [getToolDigitalInputAction](#) (int index)
Get the trigger action for tool digital input.
- [StandardInputAction](#) [getConfigurableDigitalInputAction](#) (int index)
Get the trigger action for configurable digital input.
- int [setDigitalOutputRunstateDefault](#) ()
Set all digital output runstates to None.
- int [setStandardDigitalOutputRunstate](#) (int index, [StandardOutputRunState](#) runstate)
Set the runstate for standard digital output.

- `int setToolDigitalOutputRunstate (int index, StandardOutputRunState runstate)`
Set the runstate for tool digital output.
- `int setConfigurableDigitalOutputRunstate (int index, StandardOutputRunState runstate)`
Set the runstate for configurable digital output.
- `StandardOutputRunState getStandardDigitalOutputRunstate (int index)`
Get the runstate for standard digital output.
- `StandardOutputRunState getToolDigitalOutputRunstate (int index)`
Get the runstate for tool digital output.
- `StandardOutputRunState getConfigurableDigitalOutputRunstate (int index)`
Get the runstate for configurable digital output.
- `int setStandardAnalogOutputRunstate (int index, StandardOutputRunState runstate)`
Set the runstate for standard analog output.
- `int setToolAnalogOutputRunstate (int index, StandardOutputRunState runstate)`
Set the runstate for tool analog output.
- `StandardOutputRunState getStandardAnalogOutputRunstate (int index)`
Get the runstate for standard analog output.
- `StandardOutputRunState getToolAnalogOutputRunstate (int index)`
Get the runstate for tool analog output.
- `int setDigitalOutputAfterEStopDefault ()`
Set all digital output states after emergency stop to default(no change)
- `int setAnalogOutputAfterEStopDefault ()`
Set all analog output states after emergency stop to default(no change)
- `int setStandardDigitalOutputAfterEStop (int index, bool value)`
Set the value of a standard digital output after emergency stop
- `int setConfigurableDigitalOutputAfterEStop (int index, bool value)`
Set the value of a configurable digital output after emergency stop
- `int setStandardAnalogOutputAfterEStop (int index, double value)`
Set the value of standard analog output after emergency stop
- `int setStandardAnalogInputDomain (int index, int domain)`
Set the range of standard analog input.
- `int setToolAnalogInputDomain (int index, int domain)`
Set the range of tool analog input.
- `int getStandardAnalogInputDomain (int index)`
Get the domain of standard analog input.
- `int getToolAnalogInputDomain (int index)`
Get the domain of tool analog input.
- `int setStandardAnalogOutputDomain (int index, int domain)`
Set the range of standard analog output.
- `int setToolAnalogOutputDomain (int index, int domain)`
Set the range of tool analog output.
- `int getStandardAnalogOutputDomain (int index)`
Get the domain of standard analog output.
- `int getToolAnalogOutputDomain (int index)`
Get the domain of tool analog output.
- `int setToolVoltageOutputDomain (int domain)`
Set the tool voltage output value (unit: V)
- `int getToolVoltageOutputDomain ()`
Get the tool voltage output value (unit: V)
- `int setStandardDigitalOutput (int index, bool value)`
Set the value of a standard digital output.
- `int setStandardDigitalOutputPulse (int index, bool value, double duration)`

- Set digital output pulse.
- int [setToolDigitalOutput](#) (int index, bool value)
Set the value of tool digital output.
- int [setToolDigitalOutputPulse](#) (int index, bool value, double duration)
Set tool digital output pulse.
- int [setConfigurableDigitalOutput](#) (int index, bool value)
Set the value of configurable digital output.
- int [setConfigurableDigitalOutputPulse](#) (int index, bool value, double duration)
Set configurable digital output pulse.
- int [setStandardAnalogOutput](#) (int index, double value)
Set the value of standard analog output.
- int [setToolAnalogOutput](#) (int index, double value)
Set the value of tool analog output.
- bool [getStandardDigitalInput](#) (int index)
Get the value of a standard digital input.
- uint32_t [getStandardDigitalInputs](#) ()
Get all standard digital input values.
- bool [getToolDigitalInput](#) (int index)
Get the value of tool digital input.
- uint32_t [getToolDigitalInputs](#) ()
Get all tool digital input values.
- bool [getConfigurableDigitalInput](#) (int index)
Get the value of configurable digital input.
- uint32_t [getConfigurableDigitalInputs](#) ()
Get all configurable digital input values.
- bool [getStandardDigitalOutput](#) (int index)
Get the value of a standard digital output.
- uint32_t [getStandardDigitalOutputs](#) ()
Get all standard digital output values.
- bool [getToolDigitalOutput](#) (int index)
Get the value of tool digital output.
- uint32_t [getToolDigitalOutputs](#) ()
Get all tool digital output values.
- bool [getConfigurableDigitalOutput](#) (int index)
Get the value of configurable digital output.
- uint32_t [getConfigurableDigitalOutputs](#) ()
Get all configurable digital output values.
- double [getStandardAnalogInput](#) (int index)
Get the value of standard analog input.
- double [getToolAnalogInput](#) (int index)
Get the value of tool analog input.
- double [getStandardAnalogOutput](#) (int index)
Get the value of standard analog output.
- double [getToolAnalogOutput](#) (int index)
Get the value of tool analog output.
- int [getStaticLinkInputNum](#) ()
Get the number of static link inputs.
- int [getStaticLinkOutputNum](#) ()
Get the number of static link outputs.
- uint32_t [getStaticLinkInputs](#) ()
Get all static link input values.

- `uint32_t getStaticLinkOutputs ()`
Get all static link output values.
- `bool hasEncoderSensor ()`
Whether the robot is equipped with an encoder.
- `int setEncDecoderType (int type, int range_id)`
Set the decoding method of the integrated encoder.
- `int setEncTickCount (int tick)`
Set the tick count of the integrated encoder.
- `int getEncDecoderType ()`
Get the decoder type of the encoder.
- `int getEncTickCount ()`
Get the tick count
- `int unwindEncDeltaTickCount (int delta_count)`
Prevent counting errors when the count exceeds the range
- `bool getToolButtonStatus ()`
Get the status of the tool button.
- `uint32_t getHandleIoStatus ()`
Get the status of handle buttons.
- `int getHandleType ()`
Get the handle type.

Protected Attributes

- `void * d_`

11.9.1 Detailed Description

Definition at line 46 of file [io_control.h](#).

11.9.2 Constructor & Destructor Documentation

11.9.2.1 IoControl()

`arcs::common_interface::IoControl::IoControl ()`

11.9.2.2 ~IoControl()

`virtual arcs::common_interface::IoControl::~~IoControl ()` [virtual]

11.9.3 Member Data Documentation

11.9.3.1 d_

`void* arcs::common_interface::IoControl::d_` [protected]

Definition at line 4982 of file [io_control.h](#).

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

- [include/aubo/robot/io_control.h](#)

11.10 arcs::common_interface::Math Class Reference

```
#include <math.h>
```

Public Member Functions

- [Math](#) ()
- virtual [~Math](#) ()
- [std::vector< double > poseAdd](#) (const [std::vector< double >](#) &p1, const [std::vector< double >](#) &p2)
Pose addition
- [std::vector< double > poseSub](#) (const [std::vector< double >](#) &p1, const [std::vector< double >](#) &p2)
Pose subtraction
- [std::vector< double > interpolatePose](#) (const [std::vector< double >](#) &p1, const [std::vector< double >](#) &p2, double alpha)
Calculate linear interpolation
- [std::vector< double > poseTrans](#) (const [std::vector< double >](#) &pose_from, const [std::vector< double >](#) &pose_to)
- [std::vector< double > poseTransInv](#) (const [std::vector< double >](#) &pose_from, const [std::vector< double >](#) &pose_to)
- [std::vector< double > poseInverse](#) (const [std::vector< double >](#) &pose)
Get the inverse of a pose
- [double poseDistance](#) (const [std::vector< double >](#) &p1, const [std::vector< double >](#) &p2)
Calculate distance between two poses
- [double poseAngleDistance](#) (const [std::vector< double >](#) &p1, const [std::vector< double >](#) &p2)
Calculate axis-angle difference between two poses
- [bool poseEqual](#) (const [std::vector< double >](#) &p1, const [std::vector< double >](#) &p2, double eps=5e-5)
Determine if two poses are equivalent
- [std::vector< double > transferRefFrame](#) (const [std::vector< double >](#) &F_b_a_old, const [Vector3d](#) &V_in_a, int type)
- [std::vector< double > poseRotation](#) (const [std::vector< double >](#) &pose, const [std::vector< double >](#) &rotrv)
Pose rotation
- [std::vector< double > rpyToQuaternion](#) (const [std::vector< double >](#) &rpy)
Euler angles to quaternions
- [std::vector< double > quaternionToRpy](#) (const [std::vector< double >](#) &quat)
Quaternions to euler angles
- [ResultWithErrno tcpOffsetIdentify](#) (const [std::vector< std::vector< double >](#) > &poses)
Four point method calibration for TCP offset
- [ResultWithErrno calibrateCoordinate](#) (const [std::vector< std::vector< double >](#) > &poses, int type)
Calibrate coordinate system with 3 points
- [ResultWithErrno calculateCircleFourthPoint](#) (const [std::vector< double >](#) &p1, const [std::vector< double >](#) &p2, const [std::vector< double >](#) &p3, int mode)
Based on three points on an arc, calculate the position of the midpoint of the other half of the fitted circle's arc
- [std::vector< double > forceTrans](#) (const [std::vector< double >](#) &pose_a_in_b, const [std::vector< double >](#) &force_in_a)

- `std::vector< double > getDeltaPoseBySensorDistance` (const `std::vector< double >` &distances, double position, double radius, double track_scale)
- `std::vector< double > deltaPoseTrans` (const `std::vector< double >` &pose_a_in_b, const `std::vector< double >` &ft_in_a)
- `std::vector< double > deltaPoseAdd` (const `std::vector< double >` &pose_a_in_b, const `std::vector< double >` &v_in_b)
- `std::vector< double > changePoseWithXYRef` (const `std::vector< double >` &pose_tar, const `std::vector< double >` &pose_ref)
- `std::vector< double > homMatrixToPose` (const `std::vector< double >` &homMatrix)
- `std::vector< double > poseToHomMatrix` (const `std::vector< double >` &pose)

Protected Attributes

- void * `d__`

11.10.1 Detailed Description

Definition at line 21 of file [math.h](#).

11.10.2 Constructor & Destructor Documentation

11.10.2.1 Math()

`arcs::common_interface::Math::Math ()`

11.10.2.2 ~Math()

`virtual arcs::common_interface::Math::~~Math ()` [virtual]

11.10.3 Member Data Documentation

11.10.3.1 d__

`void* arcs::common_interface::Math::d__` [protected]

Definition at line 1071 of file [math.h](#).

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

- `include/aubo/math.h`

11.11 arcs::common_interface::MotionControl Class Reference

```
#include <motion_control.h>
```

Public Member Functions

- [MotionControl](#) ()
- virtual [~MotionControl](#) ()
- double [getEqradius](#) ()
Get the equivalent radius, in meters.
- int [setEqradius](#) (double eqradius)
Set the equivalent radius, in meters.
- int [setSpeedFraction](#) (double fraction)
Dynamically adjust the robot's speed and acceleration ratio (0., 1.
- double [getSpeedFraction](#) ()
Get the speed and acceleration ratio, default is 1.
- int [speedFractionCritical](#) (bool enable)
Speed fraction critical section.
- bool [isSpeedFractionCritical](#) ()
Whether it is in the speed fraction critical section
- bool [isBlending](#) ()
Whether it is in the blending area
- int [pathOffsetLimits](#) (double v, double a)
Set the maximum speed and maximum acceleration for offset.
- int [pathOffsetCoordinate](#) (int ref_coord)
Set the reference coordinate system for offset.
- int [pathOffsetEnable](#) ()
Enable path offset
- int [pathOffsetSet](#) (const std::vector< double > &offset, int type=0)
Set path offset
- int [pathOffsetDisable](#) ()
Disable path offset
- int [pathOffsetSupv](#) (const std::vector< double > &min, const std::vector< double > &max, int strategy)
- int [jointOffsetEnable](#) ()
Enable joint offset
- int [jointOffsetSet](#) (const std::vector< double > &offset, int type=1)
Set joint offset
- int [jointOffsetDisable](#) ()
Disable joint offset
- int [getQueueSize](#) ()
Get the number of enqueued instruction segments (INST), including motion instructions such as move↵ Joint/moveLine/moveCircle and configuration instructions such as setPayload.
- int [getTrajectoryQueueSize](#) ()
Get the number of enqueued motion planning interpolation points
- int [getExecId](#) ()
Get the ID of the currently interpolating motion instruction segment.
- double [getDuration](#) (int id)
Get the expected execution duration of the motion segment with the specified ID.
- double [getMotionLeftTime](#) (int id)
Get the remaining execution time of the motion segment with the specified ID.
- int [stopMove](#) (bool quick, bool all_tasks)
StopMove is used to stop robot and external axes movements and any belonging process temporarily.
- int [startMove](#) ()
StartMove is used to resume robot, external axes movement and belonging process after the movement has been stopped

- int [storePath](#) (bool keep_sync)
 - storePath
- int [clearPath](#) ()
 - ClearPath clears the whole motion path on the current motion path level (base level or StorePath level).
- int [restoPath](#) ()
 - restoPath
- double [getProgress](#) ()
 - Get the execution progress of the current motion instruction segment.
- int [setWorkObjectHold](#) (const std::string &module_name, const std::vector< double > &mounting_pose)
 - Specify the name and mounting position when the workpiece is installed on the end of another robot or external axis.
- std::tuple< std::string, std::vector< double > > [getWorkObjectHold](#) ()
 - getWorkObjectHold
- std::vector< double > [getPauseJointPositions](#) ()
- int [setResumeStartPoint](#) (const std::vector< double > &q, int move_type, double blend_radius, const std::vector< double > &qdmax, const std::vector< double > &qddmax, const std::vector< double > &vmax, const std::vector< double > &amax)
 - Set resume motion parameters
- int [getResumeMode](#) ()
 - Get resume motion mode
- ARCS_DEPRECATED int [setServoMode](#) (bool enable)
 - Set servo mode Use setServoModeSelect instead
- ARCS_DEPRECATED bool [isServoModeEnabled](#) ()
 - Determine whether the servo mode is enabled.
- int [setServoModeSelect](#) (int mode)
 - Set servo motion mode
- int [getServoModeSelect](#) ()
 - Get the servo motion mode
- int [servoJoint](#) (const std::vector< double > &q, double a, double v, double t, double lookahead_time, double gain)
 - Joint space servo
- int [servoCartesian](#) (const std::vector< double > &pose, double a, double v, double t, double lookahead_time, double gain)
 - Cartesian space servo
- int [servoJointWithAxes](#) (const std::vector< double > &q, const std::vector< double > &extq, double a, double v, double t, double lookahead_time, double gain)
 - Servo motion (with external axes), used for executing offline trajectories, pass-through user planned trajectories, etc.
- int [servoJointWithAxisGroup](#) (const std::vector< double > &q, double a, double v, double t, double lookahead_time, double gain, const std::string &group_name, const std::vector< double > &extq)
- int [servoCartesianWithAxes](#) (const std::vector< double > &pose, const std::vector< double > &extq, double a, double v, double t, double lookahead_time, double gain)
 - Servo motion (with external axes), used for executing offline trajectories, pass-through user planned trajectories, etc.
- int [servoCartesianWithAxisGroup](#) (const std::vector< double > &pose, double a, double v, double t, double lookahead_time, double gain, const std::string &group_name, const std::vector< double > &extq)
- int [trackJoint](#) (const std::vector< double > &q, double t, double smooth_scale, double delay_sacle)
 - Tracking motion, used for executing offline trajectories or passing through user-planned trajectories, etc.
- int [trackCartesian](#) (const std::vector< double > &pose, double t, double smooth_scale, double delay_sacle)

- Tracking motion, used for executing offline trajectories or passing through user-planned trajectories, etc.
- int [followJoint](#) (const std::vector< double > &q)
 - Joint space following
 - int [followLine](#) (const std::vector< double > &pose)
 - Cartesian space following
 - int [speedJoint](#) (const std::vector< double > &qd, double a, double t)
 - Joint space velocity following
 - int [resumeSpeedJoint](#) (const std::vector< double > &qd, double a, double t)
 - Joint space velocity following (used to move to a safe position after a collision during process execution)
 - int [speedLine](#) (const std::vector< double > &xd, double a, double t)
 - Cartesian space velocity following
 - int [resumeSpeedLine](#) (const std::vector< double > &xd, double a, double t)
 - Cartesian space velocity following (used to move to a safe position after a collision during process execution)
 - int [moveSpline](#) (const std::vector< double > &q, double a, double v, double duration)
 - Perform spline interpolation in joint space
 - int [moveJoint](#) (const std::vector< double > &q, double a, double v, double blend_radius, double duration)
 - Add joint motion
 - int [moveJointWithAxisGroup](#) (const std::vector< double > &q, double a, double v, double blend_radius, double duration, const std::string &group_name, const std::vector< double > &extq)
 - Synchronous motion of robot and external axes
 - int [resumeMoveJoint](#) (const std::vector< double > &q, double a, double v, double duration)
 - Move to the pause point using joint motion.
 - int [moveLine](#) (const std::vector< double > &pose, double a, double v, double blend_radius, double duration)
 - Add linear motion
 - int [moveLineWithAxisGroup](#) (const std::vector< double > &pose, double a, double v, double blend_radius, double duration, const std::string &group_name, const std::vector< double > &extq)
 - Linear motion synchronized with external axes
 - int [moveProcess](#) (const std::vector< double > &pose, double a, double v, double blend_radius)
 - Add process motion
 - int [resumeMoveLine](#) (const std::vector< double > &pose, double a, double v, double duration)
 - Move to the pause point using linear motion.
 - int [moveCircle](#) (const std::vector< double > &via_pose, const std::vector< double > &end_pose, double a, double v, double blend_radius, double duration)
 - Add circular arc motion
 - int [moveCircleWithAxisGroup](#) (const std::vector< double > &via_pose, const std::vector< double > &end_pose, double a, double v, double blend_radius, double duration, const std::string &group_name, const std::vector< double > &extq)
 - moveCircle with external axes synchronized motion
 - int [setCirclePathMode](#) (int mode)
 - Set circle path mode
 - int [moveCircle2](#) (const [CircleParameters](#) ¶m)
 - Advanced arc or circular motion
 - int [pathBufferAlloc](#) (const std::string &name, int type, int size)
 - Create a new path point buffer
 - int [pathBufferAppend](#) (const std::string &name, const std::vector< std::vector< double > > &waypoints)
 - Add waypoints to the path buffer

- int [pathBufferEval](#) (const std::string &name, const std::vector< double > &a, const std::vector< double > &v, double t)
Perform computation and optimization (time-consuming operations).
- bool [pathBufferValid](#) (const std::string &name)
Whether the buffer with the specified name is valid
- int [pathBufferFree](#) (const std::string &name)
Release path buffer
- int [pathBufferFilter](#) (const std::string &name, int order, double fd, double fs)
Joint space path filter
- std::vector< std::string > [pathBufferList](#) ()
List all cached path names
- int [movePathBuffer](#) (const std::string &name)
Execute the cached path
- int [moveIntersection](#) (const std::vector< std::vector< double > > &poses, double a, double v, double main_pipe_radius, double sub_pipe_radius, double normal_distance, double normal_alpha)
Intersection interface
- int [stopJoint](#) (double acc)
Stop motion in joint space
- int [resumeStopJoint](#) (double acc)
Stop motion in joint space (used after moving to a safe position via resumeSpeedJoint following a collision during process execution)
- int [stopLine](#) (double acc, double acc_rot)
Stop motions in Cartesian space such as moveLine/moveCircle.
- int [resumeStopLine](#) (double acc, double acc_rot)
Stop motion in Cartesian space (used after moving to a safe position via resumeSpeedLine following a collision during process execution)
- int [weaveStart](#) (const std::string ¶ms)
Start weaving: between weaveStart and weaveEnd, moveLine/moveCircle/moveProcess follows params.
- int [weaveEnd](#) ()
End weaving
- int [setFuturePointSamplePeriod](#) (double sample_time)
Set the sampling interval for points on the future path
- std::vector< std::vector< double > > [getFuturePathPointsJoint](#) ()
Get trajectory points on the future path
- int [setConveyorTrackEncoder](#) (int encoder_id, int tick_per_meter)
Set conveyor encoder parameters
- int [conveyorTrackCircle](#) (int encoder_id, const std::vector< double > ¢er, bool rotate_tool)
Circular conveyor tracking
- int [conveyorTrackLine](#) (int encoder_id, const std::vector< double > &direction)
Linear conveyor tracking
- int [conveyorTrackStop](#) (int encoder_id, double a)
Stop conveyor tracking
- bool [conveyorTrackSwitch](#) (int encoder_id)
Switch conveyor tracking item.
- bool [hasItemOnConveyorToTrack](#) (int encoder_id)
Whether there is an item on the conveyor that can be tracked
- int [conveyorTrackCreatItem](#) (int encoder_id, int item_id, const std::vector< double > &offset)
Add an item to the conveyor queue
- int [setConveyorTrackCompensate](#) (int encoder_id, double comp)
Set the compensation value for conveyor tracking
- bool [isConveyorTrackSync](#) (int encoder_id)

- Determine whether the conveyor and the robot arm have reached relative rest
- int [setConveyorTrackLimit](#) (int encoder_id, double limit)
 - Set the maximum distance limit for conveyor tracking
- int [setConveyorTrackStartWindow](#) (int encoder_id, double window_min, double window_max)
 - Set the start window for conveyor tracking
- int [setConveyorTrackSensorOffset](#) (int encoder_id, double offset)
 - Set the distance from the conveyor teaching position to the sync switch
- int [setConveyorTrackSyncSeparation](#) (int encoder_id, double distance, double time)
 - Set conveyor sync separation, used to filter out unwanted signals from the sync switch.
- bool [isConveyorTrackExceed](#) (int encoder_id)
 - Whether the workpiece on the conveyor has moved beyond the maximum limit
- int [conveyorTrackClearItems](#) (int encoder_id)
 - Clear all items in the conveyor queue
- std::vector< int > [getConveyorTrackQueue](#) (int encoder_id)
 - Get encoder values of the conveyor queue
- int [getConveyorTrackNextItem](#) (int encoder_id)
 - Get the ID of the next item on the conveyor to be tracked
- int [moveSpiral](#) (const [SpiralParameters](#) ¶m, double blend_radius, double v, double a, double t)
 - Spiral motion
- int [getLookAheadSize](#) ()
 - Get look-ahead segment size
- int [setLookAheadSize](#) (int size)
 - Set look-ahead segment size
- int [weaveUpdateParameters](#) (const std::string ¶ms)
 - Update frequency and amplitude during weaving process
- int [enableJointSoftServo](#) (const std::vector< double > &stiffness)
 - Set joint current loop stiffness coefficient
- int [disableJointSoftServo](#) ()
 - Disable joint current loop stiffness coefficient
- bool [isJointSoftServoEnabled](#) ()
 - Determine whether the joint current loop stiffness coefficient is enabled.
- int [enableVibrationSuppress](#) (const std::vector< double > &omega, const std::vector< double > &zeta, int level)
 -
- int [disbaleVibrationSuppress](#) ()
 -
- int [setTimeOptimalEnable](#) (bool enable)
 -
- bool [isTimeOptimalEnabled](#) ()
 - Get the time optimal algorithm state: true - enable false - disable
- bool [isSupportedTimeOptimal](#) ()
 - Check whether the time optimal algorithm is supported true - supported false - not supported
- int [setTcpMaxLinearVelocity](#) (double v)
 - Set TCP maximum linear velocity
- double [getTcpMaxLinearVelocity](#) ()
 - Get TCP maximum linear velocity
- int [resetTcpMaxLinearVelocity](#) ()
 - Reset TCP maximum linear velocity
- int [setEndPath](#) ()
 - Set end path (terminate blending at current trajectory segment)

Protected Attributes

- void * [d_](#)

11.11.1 Detailed Description

Definition at line 69 of file [motion_control.h](#).

11.11.2 Constructor & Destructor Documentation

11.11.2.1 MotionControl()

arcs::common_interface::MotionControl::MotionControl ()

11.11.2.2 ~MotionControl()

virtual arcs::common_interface::MotionControl::~~MotionControl () [virtual]

11.11.3 Member Function Documentation

11.11.3.1 getConveyorTrackNextItem()

int arcs::common_interface::MotionControl::getConveyorTrackNextItem (
int encoder_id)

Get the ID of the next item on the conveyor to be tracked

Parameters

encoder_id	Reserved
------------	----------

Returns

Returns the item ID; returns -1 if there is no next item

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
getConveyorTrackNextItem(self: pyaubo_sdk.MotionControl, arg0: int) -> int
```

Lua function prototype

```
getConveyorTrackNextItem(encoder_id: number) -> int
```

JSON-RPC Request Example

```
{"jsonrpc": "2.0", "method": "rob1.MotionControl.getConveyorTrackNextItem", "params": [0, "id": 1]}
```

JSON-RPC Response Example

```
{"id": 1, "jsonrpc": "2.0", "result": {10}}
```

11.11.3.2 servoCartesianWithAxisGroup()

```
int arcs::common_interface::MotionControl::servoCartesianWithAxisGroup (
    const std::vector< double > & pose,
    double a,
    double v,
    double t,
    double lookahead_time,
    double gain,
    const std::string & group_name,
    const std::vector< double > & extq)
```

11.11.3.3 servoJointWithAxisGroup()

```
int arcs::common_interface::MotionControl::servoJointWithAxisGroup (
    const std::vector< double > & q,
    double a,
    double v,
    double t,
    double lookahead_time,
    double gain,
    const std::string & group_name,
    const std::vector< double > & extq)
```

11.11.4 Member Data Documentation

11.11.4.1 d__

void* arcs::common_interface::MotionControl::d__ [protected]

Definition at line 6562 of file [motion_control.h](#).

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

- include/aubo/robot/[motion_control.h](#)

11.12 arcs::common_interface::RegisterControl Class Reference

```
#include <register_control.h>
```

Public Member Functions

- [RegisterControl](#) ()
- virtual [~RegisterControl](#) ()
- bool [getBoolInput](#) (uint32_t address)
Reads the boolean from one of the input registers, which can also be accessed by a Field bus.
- int [setBoolInput](#) (uint32_t address, bool value)
- int [getInt32Input](#) (uint32_t address)
Reads the integer from one of the input registers, which can also be accessed by a FieldBus.
- int [setInt32Input](#) (uint32_t address, int value)
- float [getFloatInput](#) (uint32_t address)
Reads the float from one of the input registers, which can also be accessed by a Field bus.
- int [setFloatInput](#) (uint32_t address, float value)
- double [getDoubleInput](#) (uint32_t address)
Reads the double value from one of the input registers, which can also be accessed by a FieldBus.
- int [setDoubleInput](#) (uint32_t address, double value)
- bool [getBoolOutput](#) (uint32_t address)
Reads the boolean from one of the output registers, which can also be accessed by a Field bus.
- int [setBoolOutput](#) (uint32_t address, bool value)
- int [getInt32Output](#) (uint32_t address)
Reads the integer from one of the output registers, which can also be accessed by a FieldBus.
- int [setInt32Output](#) (uint32_t address, int value)
- float [getFloatOutput](#) (uint32_t address)
Reads the float from one of the output registers, which can also be accessed by a FieldBus.
- int [setFloatOutput](#) (uint32_t address, float value)
- double [getDoubleOutput](#) (uint32_t address)
Reads the double value from one of the output registers.
- int [setDoubleOutput](#) (uint32_t address, double value)
- int16_t [getInt16Register](#) (uint32_t address)
Used for Modbus Slave
- int [setInt16Register](#) (uint32_t address, int16_t value)
- bool [getInt16RegisterBit](#) (uint32_t address, uint8_t bit_offset)
Get the status of a specific bit in an Int16 register
- int [setInt16RegisterBit](#) (uint32_t address, uint8_t bit_offset, bool value)
Set the status of a specific bit in an Int16 register
- bool [hasNamedVariable](#) (const std::string &key)
Whether the named variable exists
- std::string [getNamedVariableType](#) (const std::string &key)
Get the type of a named variable
- bool [variableUpdated](#) (const std::string &key, uint64_t since)
Whether the named variable has been updated
- bool [getBool](#) (const std::string &key, bool default_value)
Get variable value
- int [setBool](#) (const std::string &key, bool value)
Set or update the variable value
- std::vector< char > [getVecChar](#) (const std::string &key, const std::vector< char > &default_value)
Get variable value
- int [setVecChar](#) (const std::string &key, const std::vector< char > &value)
Set or update the variable value
- int [getInt32](#) (const std::string &key, int default_value)
Get variable value
- int [setInt32](#) (const std::string &key, int value)

- Set or update the variable value
- `std::vector< int32_t > getVecInt32 (const std::string &key, const std::vector< int32_t > &default_↵_value)`
- Get variable value
- `int setVecInt32 (const std::string &key, const std::vector< int32_t > &value)`
- Set or update the variable value
- `float getFloat (const std::string &key, float default_value)`
- Get variable value
- `int setFloat (const std::string &key, float value)`
- Set or update the variable value
- `std::vector< float > getVecFloat (const std::string &key, const std::vector< float > &default_↵_value)`
- Get variable value
- `int setVecFloat (const std::string &key, const std::vector< float > &value)`
- Set or update the variable value
- `double getDouble (const std::string &key, double default_value)`
- Get variable value
- `int setDouble (const std::string &key, double value)`
- Set or update the variable value
- `std::vector< double > getVecDouble (const std::string &key, const std::vector< double > &default_↵_value)`
- Get variable value
- `int setVecDouble (const std::string &key, const std::vector< double > &value)`
- Set or update the variable value
- `std::string getString (const std::string &key, const std::string &default_value)`
- Get variable value
- `int setString (const std::string &key, const std::string &value)`
- Set or update the variable value
- `int clearNamedVariable (const std::string &key)`
- Clear variable
- `int setWatchDog (const std::string &key, double timeout, int action)`
- Set the watchdog
- `int getWatchDogAction (const std::string &key)`
- Get the watchdog action
- `int getWatchDogTimeout (const std::string &key)`
- Get the watchdog timeout value
- `int modbusAddSignal (const std::string &device_info, int slave_number, int signal_address, int signal_type, const std::string &signal_name, bool sequential_mode)`
- Adds a new modbus signal for the controller to supervise.
- `int modbusDeleteSignal (const std::string &signal_name)`
- Deletes the signal identified by the supplied signal name.
- `int modbusDeleteAllSignals ()`
- Delete all modbus signals
- `int modbusGetSignalStatus (const std::string &signal_name)`
- Reads the current value of a specific signal.
- `std::vector< std::string > modbusGetSignalNames ()`
- Get the collection of all signal names
- `std::vector< int > modbusGetSignalTypes ()`
- Get the collection of all signal types
- `std::vector< int > modbusGetSignalValues ()`
- Get the collection of all signal values
- `std::vector< int > modbusGetSignalErrors ()`

- Get the error status of all signal requests (0: no error, others: error) as a collection

 - int [modbusSendCustomCommand](#) (const std::string &device_info, int slave_number, int function_code, const std::vector< uint8_t > &data)

Sends a command specified by the user to the modbus unit located on the specified IP address.
- int [modbusSetDigitalInputAction](#) (const std::string &robot_name, const std::string &signal_name, [StandardInputAction](#) action)

Sets the selected digital input signal to either a "default" or "freedrive" action.
- int [modbusSetOutputRunstate](#) (const std::string &robot_name, const std::string &signal_name, [StandardOutputRunState](#) runstate)

Set Modbus signal output action
- int [modbusSetOutputSignal](#) (const std::string &signal_name, uint16_t value)

Sets the output register signal identified by the given name to the given value.
- int [modbusSetOutputSignal1](#) (const std::string &signal_name, const std::vector< uint16_t > &values)

Sets multiple consecutive output register signals starting from the given name to the given values.
- int [modbusSetOutputSignalWithTimeout](#) (const std::string &signal_name, uint16_t value, double timeout)

Sets the output register signal identified by the given name to the given, with timeout value.
- int [modbusSetOutputSignalPulse](#) (const std::string &signal_name, uint16_t value, double duration)

Set modbus signal output pulse (only supports coil output type)
- int [modbusSetSignalUpdateFrequency](#) (const std::string &signal_name, int update_frequency)

Sets the frequency with which the robot will send requests to the Modbus controller to either read or write the signal value.
- int [modbusGetSignalIndex](#) (const std::string &signal_name)

Get the index of the specified modbus signal, starting from 0.
- int [modbusGetSignalError](#) (const std::string &signal_name)

Get the error status of the specified modbus signal
- int [getModbusDeviceStatus](#) (const std::string &device_name)

Get the connection status of the specified modbus device
- int [addModbusEncoder](#) (int encoder_id, int range_id, const std::string &signal_name)

Use a modbus register signal as an encoder
- int [addInt32RegEncoder](#) (int encoder_id, int range_id, const std::string &key)

Add a virtual encoder for an Int32 register
- int [deleteVirtualEncoder](#) (int encoder_id)

Delete virtual encoder

Protected Attributes

- void * [d_](#)

11.12.1 Detailed Description

Definition at line 64 of file [register_control.h](#).

11.12.2 Constructor & Destructor Documentation

11.12.2.1 RegisterControl()

arcs::common_interface::RegisterControl::RegisterControl ()

11.12.2.2 ~RegisterControl()

virtual arcs::common_interface::RegisterControl::~~RegisterControl () [virtual]

11.12.3 Member Function Documentation

11.12.3.1 modbusSetOutputSignalWithTimeout()

```
int arcs::common_interface::RegisterControl::modbusSetOutputSignalWithTimeout (
    const std::string & signal_name,
    uint16_t value,
    double timeout)
```

Sets the output register signal identified by the given name to the given, with timeout value.

Parameters

signal_name	A string identifying an output register signal that in advance has been added.
value	An integer which must be a valid word (0-65535)
timeout	seconds

Returns

Python interface prototype

```
modbusSetOutputSignalWithTimeout(self: pyaubo_sdk.RegisterControl, arg0: str, arg1: int,
arg2: timeout) -> int
```

Lua interface prototype

```
modbusSetOutputSignalWithTimeout(signal_name: string, value: number, timeout: number) ->
nil
```

Lua example

```
modbusSetOutputSignalWithTimeout("Modbus_0",0,0.5)
```

JSON-RPC request example

```
{"jsonrpc":"2.0","method":"RegisterControl.modbusSetOutputSignalWithTimeout","params":["↵
Modbus_0",0,0.5],"id":1}
```

JSON-RPC response example

```
{"id":1,"jsonrpc":"2.0","result":0}
```

11.12.4 Member Data Documentation

11.12.4.1 d__

void* arcs::common_interface::RegisterControl::d__ [protected]

Definition at line 3369 of file [register_control.h](#).

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

- include/aubo/[register_control.h](#)

11.13 arcs::common_interface::RobotAlgorithm Class Reference

```
#include <robot_algorithm.h>
```

Public Member Functions

- [RobotAlgorithm](#) ()
- virtual [~RobotAlgorithm](#) ()
- [ForceSensorCalibResult](#) [calibrateTcpForceSensor](#) (const std::vector< std::vector< double > > &forces, const std::vector< std::vector< double > > &poses)
Force sensor calibration algorithm (three-point calibration method)
- [ForceSensorCalibResultWithError](#) [calibrateTcpForceSensor2](#) (const std::vector< std::vector< double > > &forces, const std::vector< std::vector< double > > &poses)
Force sensor calibration algorithm (three-point calibration method)
- [ResultWithErrno](#) [calibrateTcpForceSensor3](#) (const std::vector< std::vector< double > > &forces, const std::vector< std::vector< double > > &poses, const double &mass, const std::vector< double > &cog)
Force sensor offset calibration algorithm
- int [payloadIdentify](#) (const std::string &data_file_no_payload, const std::string &data_file_with_payload)
[Payload](#) identification algorithm interface based on current
- int [payloadIdentify1](#) (const std::string &file_name)
New version of payload identification algorithm interface based on current
- int [payloadCalculateFinished](#) ()
Whether payload identification calculation is finished
- [Payload](#) [getPayloadIdentifyResult](#) ()
Get the result of payload identification
- bool [frictionModelIdentify](#) (const std::vector< std::vector< double > > &q, const std::vector< std::vector< double > > &qd, const std::vector< std::vector< double > > &qdd, const std::vector< std::vector< double > > &temp)
Joint friction model identification algorithm interface
- [ResultWithErrno](#) [calibWorkpieceCoordinatePara](#) (const std::vector< std::vector< double > > &q, int type)
Workpiece coordinate calibration algorithm interface (requires correct TCP offset set before calling)
Input multiple sets of joint angles and calibration type, output workpiece coordinate pose (relative to robot base)
- [ResultWithErrno](#) [forwardDynamics](#) (const std::vector< double > &q, const std::vector< double > &torqs)

- Forward dynamics
- [ResultWithErrno forwardDynamics1](#) (const std::vector< double > &q, const std::vector< double > &torqs, const std::vector< double > &tcp_offset)
Forward dynamics based on the given TCP offset
 - [ResultWithErrno forwardKinematics](#) (const std::vector< double > &q)
Forward kinematics, based on the activated TCP offset (the most recently set via setTcpOffset) Input joint angles, output TCP pose
 - [ResultWithErrno forwardKinematics1](#) (const std::vector< double > &q, const std::vector< double > &tcp_offset)
Forward kinematics Input joint angles, output TCP pose
 - [ResultWithErrno forwardToolKinematics](#) (const std::vector< double > &q)
Forward kinematics (ignoring TCP offset)
 - [ResultWithErrno1 forwardKinematicsAll](#) (const std::vector< double > &q)
Forward kinematics, based on the activated TCP offset (the most recently set via setTcpOffset) Input joint angles, output links poses
 - [ResultWithErrno inverseKinematics](#) (const std::vector< double > &qnear, const std::vector< double > &pose)
Inverse kinematics Input TCP pose and reference joint angles, output joint angles
 - [ResultWithErrno inverseKinematics1](#) (const std::vector< double > &qnear, const std::vector< double > &pose, const std::vector< double > &tcp_offset)
Inverse kinematics Input TCP pose and reference joint angles, output joint angles
 - [ResultWithErrno1 inverseKinematicsAll](#) (const std::vector< double > &pose)
Solve all inverse kinematics solutions based on the activated TCP offset
 - [ResultWithErrno1 inverseKinematicsAll1](#) (const std::vector< double > &pose, const std::vector< double > &tcp_offset)
Solve all inverse kinematics solutions based on the provided TCP offset
 - [ResultWithErrno inverseToolKinematics](#) (const std::vector< double > &qnear, const std::vector< double > &pose)
Inverse kinematics (ignoring TCP offset)
 - [ResultWithErrno1 inverseToolKinematicsAll](#) (const std::vector< double > &pose)
Inverse kinematics (ignoring TCP offset)
 - [ResultWithErrno3 getRobotConfiguration](#) (const std::vector< double > &q)
Calculate and return the corresponding robot configuration based on input joint angles
 - std::vector< std::vector< double > > [pathMovej](#) (const std::vector< double > &q1, double r1, const std::vector< double > &q2, double r2, double d)
Solve the trajectory points between movej
 - [ResultWithErrno calcJacobian](#) (const std::vector< double > &q, bool base_or_end)
Calculate the Jacobian matrix at the robot end-effector
 - std::vector< std::vector< double > > [pathBlend3Points](#) (int type, const std::vector< double > &q←_start, const std::vector< double > &q_via, const std::vector< double > &q_to, double r, double d)
Solve the blended trajectory points
 - int [generatePayloadIdentifyTraj](#) (const std::string &name, const [TrajConfig](#) &traj_conf)
Generate excitation trajectory for payload identification This interface internally calls pathBufferAppend The offline trajectory is stored in the buffer, and can be executed later via movePathBuffer
 - int [payloadIdentifyTrajGenFinished](#) ()
Whether payload identification trajectory generation is finished
 - std::vector< std::vector< double > > [pathMoveS](#) (const std::vector< std::vector< double > > &qs, double d)
Solve the trajectory points for moveS
 - [ResultWithErrno1 calibVibrationParams](#) (const std::vector< std::vector< double > > &q, const std::vector< std::vector< double > > &qd, const std::vector< std::vector< double > > &target_q, const std::vector< std::vector< double > > &target_qd, const std::vector< std::vector< double > > &target_qdd, const std::vector< double > &tool_offset)

- [ResultWithErrno1](#) [calibVibrationParams1](#) (const std::string &record_cache_name, const std::vector< double > &tool_offset)
- int [needVibrationRecalib](#) (const [VibrationRecalibrationParameter](#) ¶m1, const [VibrationRecalibrationParameter](#) ¶m2, double threshold)
- int [validatePath](#) (int type, const std::vector< double > &start, double r1, const std::vector< double > &end, double r2, double d)

Validate the reachability of the robot's motion path from start point to end point

Protected Attributes

- void * [d__](#)

11.13.1 Detailed Description

Definition at line 30 of file [robot_algorithm.h](#).

11.13.2 Constructor & Destructor Documentation

11.13.2.1 RobotAlgorithm()

arcs::common_interface::RobotAlgorithm::RobotAlgorithm ()

11.13.2.2 ~RobotAlgorithm()

virtual arcs::common_interface::RobotAlgorithm::~~RobotAlgorithm () [virtual]

11.13.3 Member Function Documentation

11.13.3.1 pathMovej()

```
std::vector< std::vector< double > > arcs::common_interface::RobotAlgorithm::pathMovej (
    const std::vector< double > & q1,
    double r1,
    const std::vector< double > & q2,
    double r2,
    double d)
```

Solve the trajectory points between movej

Parameters

q1	Start point of movej
r1	Blend radius at q1
q2	End point of movej

r2	Blend radius at q2
d	Sampling distance

Returns

Discrete trajectory points (x, y, z, rx, ry, rz) between q1 and q2 in Cartesian space

Exceptions

arcs::common_interface::AuboException	
---	--

Python function prototype

```
pathMovej(self: pyaubo_sdk.RobotAlgorithm, arg0: List[float], arg1: float, arg2: List[float], arg3: float, arg4: float) -> List[List[float]]
```

Lua function prototype

```
pathMovej(q1: table, r1: number, q2: table, r2: number, d: number) -> table, number
```

Lua example

```
path , num = pathMovej({0.0,-0.2618,1.7453,0.4364,1.5711,0.0},0.25,{0.3234,-0.5405,1.5403,0.↵
5881,1.2962,0.7435},0.03,0.2)
```

11.13.4 Member Data Documentation

11.13.4.1 d_

void* arcs::common_interface::RobotAlgorithm::d_ [protected]

Definition at line 1691 of file [robot_algorithm.h](#).

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

- include/aubo/robot/[robot_algorithm.h](#)

11.14 arcs::common_interface::RobotConfig Class Reference

```
#include <robot_config.h>
```

Public Member Functions

- [RobotConfig](#) ()
- virtual [~RobotConfig](#) ()
- std::string [getName](#) ()
Get the robot's name.
- int [getDof](#) ()
Get the robot's degrees of freedom (from the hardware abstraction layer).
- double [getCycletime](#) ()
Get the robot's servo control cycle time (from the hardware abstraction layer).
- int [setSlowDownFraction](#) (int level, double fraction)
Set the speed reduction ratio for the preset slow-down mode.
- double [getSlowDownFraction](#) (int level)
Get the speed reduction ratio for the preset slow-down mode.
- double [getDefaultToolAcc](#) ()
Get the default tool acceleration, in m/s².
- double [getDefaultToolSpeed](#) ()
Get the default tool speed, in m/s.
- double [getDefaultJointAcc](#) ()
Get the default joint acceleration, in rad/s².
- double [getDefaultJointSpeed](#) ()
Get the default joint speed, in rad/s.
- std::string [getRobotType](#) ()
Get the robot type code.
- std::string [getRobotSubType](#) ()
Get the robot sub-type code.
- std::string [getControlBoxType](#) ()
Get the control box type code.
- int [setMountingPose](#) (const std::vector< double > &pose)
Set the mounting pose (robot base coordinate system relative to world coordinate system) world->base
- std::vector< double > [getMountingPose](#) ()
Get the mounting pose (robot base coordinate system relative to world coordinate system)
- int [attachRobotBaseTo](#) (const std::string &frame)
Attach the robot base to a coordinate frame.
- std::string [getRobotBaseParent](#) ()
- int [setWorkObjectData](#) (const [WObjectData](#) &wobj)
Set work object data.
- int [setCollisionLevel](#) (int level)
Set the collision sensitivity level.
- int [getCollisionLevel](#) ()
Get the collision sensitivity level.
- int [setCollisionStopType](#) (int type)
Set the collision stop type.
- int [getCollisionStopType](#) ()
Get the collision stop type.
- int [setHomePosition](#) (const std::vector< double > &positions)
Set the robot's home position.
- std::vector< double > [getHomePosition](#) ()
Get the robot's home position.
- int [setFreedriveDamp](#) (const std::vector< double > &damp)
Set freedrive damping.

- `std::vector< double > getFreedriveDamp ()`
Get freedrive damping.
- `int setHandguidDamp (const std::vector< double > &damp)`
Set handguiding damping.
- `std::vector< double > getHandguidDamp ()`
Get handguiding damping.
- `std::unordered_map< std::string, std::vector< double > > getKinematicsParam (bool real)`
Get the robot's DH parameters: alpha, a, d, theta, beta.
- `std::unordered_map< std::string, std::vector< double > > getKinematicsCompensate (double ref←_temperature)`
Get DH parameter compensation values (alpha, a, d, theta, beta) at the specified temperature.
- `int setKinematicsCompensate (const std::unordered_map< std::string, std::vector< double > > ¶m)`
Set standard DH compensation to the robot.
- `int setPersistentParameters (const std::string ¶m)`
Set parameters to be saved to the interface board base.
- `int setHardwareCustomParameters (const std::string ¶m)`
Set custom parameters for the hardware abstraction layer.
- `std::string getHardwareCustomParameters (const std::string ¶m)`
Get custom parameters from the hardware abstraction layer.
- `int setRobotZero ()`
Set the robot joint zero position.
- `std::vector< std::string > getTcpForceSensorNames ()`
Get the names of available TCP force sensors.
- `int selectTcpForceSensor (const std::string &name)`
Set the TCP force sensor.
- `int setTcpForceSensorPose (const std::vector< double > &sensor_pose)`
Set the sensor mounting pose.
- `std::vector< double > getTcpForceSensorPose ()`
Get the sensor mounting pose.
- `bool hasTcpForceSensor ()`
Whether a TCP force sensor is installed.
- `int setTcpForceOffset (const std::vector< double > &force_offset)`
Set the TCP force/torque offset.
- `std::vector< double > getTcpForceOffset ()`
Get the TCP force/torque offset.
- `std::vector< std::string > getBaseForceSensorNames ()`
Get the names of available base force sensors.
- `int selectBaseForceSensor (const std::string &name)`
Set the base force sensor.
- `bool hasBaseForceSensor ()`
Whether a base force sensor is installed.
- `int setBaseForceOffset (const std::vector< double > &force_offset)`
Set the base force/torque offset.
- `std::vector< double > getBaseForceOffset ()`
Get the base force/torque offset.
- `uint32_t getSafetyParametersChecksum ()`
Get the safety parameter checksum CRC32.
- `int confirmSafetyParameters (const RobotSafetyParameterRange ¶meters)`
Initiate a request to confirm safety configuration parameters: Write safety configuration parameters to the safety interface board flash or file.

- `uint32_t calcSafetyParametersChecksum` (const [RobotSafetyParameterRange](#) ¶meters)
Calculate the CRC32 checksum of safety parameters.
- `std::vector< double > getJointMaxPositions` ()
Get the joint maximum positions (physical limits).
- `std::vector< double > getJointMinPositions` ()
Get the joint minimum positions (physical limits).
- `std::vector< double > getJointMaxSpeeds` ()
Get the joint maximum speeds (physical limits).
- `std::vector< double > getJointMaxAccelerations` ()
Get the joint maximum accelerations (physical limits).
- `std::vector< double > getTcpMaxSpeeds` ()
Get the TCP maximum speed (physical limits).
- `std::vector< double > getTcpMaxAccelerations` ()
Get the TCP maximum acceleration (physical limits).
- `int setGravity` (const std::vector< double > &gravity)
Set the robot installation attitude (gravity vector).
- `std::vector< double > getGravity` ()
Get the robot's installation attitude (gravity vector).
- `int setTcpOffset` (const std::vector< double > &offset)
Set the current TCP offset.
- `std::vector< double > getTcpOffset` ()
Get the current TCP offset.
- `int setToolInertial` (double m, const std::vector< double > &com, const std::vector< double > &inertial)
Set tool mass, center of mass, and inertia.
- `int setPayload` (double m, const std::vector< double > &cog, const std::vector< double > &aom, const std::vector< double > &inertia)
Set the payload.
- `Payload getPayload` ()
Get the payload.
- `bool toolSpaceInRange` (const std::vector< double > &pose)
Whether the end-effector pose is within the safety range.
- `int firmwareUpdate` (const std::string &fw)
Initiate a firmware upgrade request.
- `std::tuple< std::string, double > getFirmwareUpdateProcess` ()
Get the current firmware upgrade process.
- `std::vector< double > getLimitJointMaxPositions` ()
Get the joint maximum positions (currently used limit values).
- `std::vector< double > getLimitJointMinPositions` ()
Get the joint minimum positions (currently used limit values).
- `std::vector< double > getLimitJointMaxSpeeds` ()
Get the joint maximum speeds (currently used limit values).
- `std::vector< double > getLimitJointMaxAccelerations` ()
Get the joint maximum accelerations (currently used limit values).
- `double getLimitTcpMaxSpeed` ()
Get the TCP maximum speed (currently used limit value).
- `SafeguardStopType getSafeguardStopType` ()
Get the current safeguard stop type.
- `int getSafeguardStopSource` ()
Get the complete safeguard stop source as a bitmask.
- `int getRobotEmergencyStopSource` ()

- Get the complete robot emergency stop source as a bitmask.
- `std::string` [getSelectedTcpForceSensorName](#) ()
 - Get the name of the selected TCP force sensor.
- `int` [attachWeldingGun](#) (double m, const `std::vector< double >` &cog, const `std::vector< double >` &inertia)
- `int` [setCollisionThreshold](#) (const `std::vector< double >` &threshold)
 - Enable optimized collision force.
- `std::vector< double >` [getCollisionThreshold](#) ()
 - Get collision threshold.
- `int` [enableAxisGroup](#) (const `std::string` &group_name)
- `int` [disableAxisGroup](#) ()
- `int` [enableEndCollisionCheck](#) (bool enable)
 - Set end collision threshold.
- `bool` [isEndCollisionCheckEnabled](#) ()
 - Get collision check is enabled.

Protected Attributes

- `void *` [d_](#)

11.14.1 Detailed Description

Definition at line 21 of file [robot_config.h](#).

11.14.2 Constructor & Destructor Documentation

11.14.2.1 RobotConfig()

`arcs::common_interface::RobotConfig::RobotConfig ()`

11.14.2.2 ~RobotConfig()

`virtual arcs::common_interface::RobotConfig::~~RobotConfig ()` [virtual]

11.14.3 Member Function Documentation

11.14.3.1 disableAxisGroup()

`int arcs::common_interface::RobotConfig::disableAxisGroup ()`

11.14.3.2 enableAxisGroup()

`int arcs::common_interface::RobotConfig::enableAxisGroup (`
 `const std::string & group_name)`

11.14.3.3 getRobotBaseParent()

```
std::string arcs::common_interface::RobotConfig::getRobotBaseParent ()
```

11.14.4 Member Data Documentation

11.14.4.1 d_

```
void* arcs::common_interface::RobotConfig::d_ [protected]
```

Definition at line 3806 of file [robot_config.h](#).

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

- [include/aubo/robot/robot_config.h](#)

11.15 arcs::common_interface::RobotInterface Class Reference

```
#include <robot_interface.h>
```

Public Member Functions

- [RobotInterface](#) ()
- virtual [~RobotInterface](#) ()
- [RobotConfigPtr](#) [getRobotConfig](#) ()
 [RobotConfig](#) () Get [RobotConfig](#) interface
- [MotionControlPtr](#) [getMotionControl](#) ()
 [MotionControl](#) () Get motion planning interface
- [ForceControlPtr](#) [getForceControl](#) ()
 [ForceControl](#)() Get force control interface
- [IoControlPtr](#) [getIoControl](#) ()
 [IoControl](#) (IO) Get IO control interface
- [SyncMovePtr](#) [getSyncMove](#) ()
 [SyncMove](#) () Get synchronized motion interface
- [RobotAlgorithmPtr](#) [getRobotAlgorithm](#) ()
 [RobotAlgorithm](#) () Get robot utility algorithm interface
- [RobotManagePtr](#) [getRobotManage](#) ()
 [RobotManage](#) () Get robot management interface (power on, start, stop, etc.)
- [RobotStatePtr](#) [getRobotState](#) ()
 [RobotState](#) () Get robot state interface
- [TracePtr](#) [getTrace](#) ()
 [Trace](#) () Get alarm information interface

Protected Attributes

- void * [d_](#)

11.15.1 Detailed Description

Definition at line 25 of file [robot_interface.h](#).

11.15.2 Constructor & Destructor Documentation

11.15.2.1 RobotInterface()

```
arcs::common_interface::RobotInterface::RobotInterface ()
```

11.15.2.2 ~RobotInterface()

```
virtual arcs::common_interface::RobotInterface::~~RobotInterface () [virtual]
```

11.15.3 Member Data Documentation

11.15.3.1 d__

```
void* arcs::common_interface::RobotInterface::d__ [protected]
```

Definition at line 389 of file [robot_interface.h](#).

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

- include/aubo/[robot_interface.h](#)

11.16 arcs::common_interface::RobotManage Class Reference

```
#include <robot_manage.h>
```

Public Member Functions

- [RobotManage](#) ()
- virtual [~RobotManage](#) ()
- int [poweron](#) ()
Initiate robot power-on request
- int [startup](#) ()
Initiate robot startup request
- int [releaseRobotBrake](#) ()
Initiate robot brake release request
- int [lockRobotBrake](#) ()
Initiate robot brake lock request
- int [poweroff](#) ()
Initiate robot power-off request
- int [backdrive](#) (bool enable)
Initiate robot backdrive request

- int [freedrive](#) (bool enable)
Initiate robot freedrive request This interface is deprecated after software version 0.31.x, use [handguideMode](#) instead: `handguideMode({1,1,1,1,1}, {0,0,0,0,0})`
- int [setHandguideParams](#) (const std::vector< int > &freeAxes, const std::vector< double > &feature)
Advanced hand-guiding mode
- std::vector< int > [getHandguideFreeAxes](#) ()
Get Axes that can be moved
- std::vector< double > [getHandguideFeature](#) ()
Get the drag reference coordinate system
- int [handguideMode](#) (const std::vector< int > &freeAxes, const std::vector< double > &feature)
Advanced hand-guiding mode
- int [exitHandguideMode](#) ()
Exit hand-guiding mode
- int [getHandguideStatus](#) ()
Get the status of the hand-guiding device (whether it is in a singular space)
- int [getHandguideTrigger](#) ()
Get the trigger source of the hand-guiding device
- bool [isHandguideEnabled](#) ()
Get the hand-guiding enable status
- int [setSim](#) (bool enable)
Initiate robot enter/exit simulation mode request
- int [setOperationalMode](#) ([OperationalModeType](#) mode)
Set the robot operational mode
- [OperationalModeType](#) [getOperationalMode](#) ()
Get the robot operational mode
- [RobotControlModeType](#) [getRobotControlMode](#) ()
Get the control mode
- bool [isFreedriveEnabled](#) ()
Whether the freedrive mode is enabled
- bool [isBackdriveEnabled](#) ()
Whether the backdrive mode is enabled
- bool [isSimulationEnabled](#) ()
Whether the simulation mode is enabled
- int [setUnlockProtectiveStop](#) ()
Clear protective stop, including collision stop
- int [restartInterfaceBoard](#) ()
Reset the safety interface board, usually called after the robot is powered off and needs to be reset, such as after emergency stop or fault.
- int [recordCacheFree](#) (const std::string &name)
Free and clear recorded data of the specified memory cache
- int [startRecordCache](#) (const std::string &name)
Start real-time trajectory recording to memory cache (no file output)
- int [stopRecordCache](#) ()
Stop current real-time trajectory recording to memory cache
- int [pauseRecordCache](#) (bool pause)
Pause or resume current real-time trajectory recording to memory cache
- int [getRecordCache](#) (const std::string &name, size_t frames=0)
Get recorded data from the specified memory cache
- int [startRecord](#) (const std::string &file_name)
Start real-time trajectory recording

- int [stopRecord](#) ()
Stop real-time recording
- int [pauseRecord](#) (bool pause)
Pause real-time recording
- int [setLinkModeEnable](#) (bool enable)
Initiate robot enter/exit link mode request.
- bool [isLinkModeEnabled](#) ()
Whether the link mode is enabled.
- int [generateDiagnoseFile](#) (const std::string &reason)
Manually trigger the generation of a diagnostic file

Protected Attributes

- void * [d__](#)

11.16.1 Detailed Description

Definition at line [21](#) of file [robot_manage.h](#).

11.16.2 Constructor & Destructor Documentation

11.16.2.1 RobotManage()

`arcs::common_interface::RobotManage::RobotManage ()`

11.16.2.2 ~RobotManage()

`virtual arcs::common_interface::RobotManage::~~RobotManage ()` [virtual]

11.16.3 Member Data Documentation

11.16.3.1 d__

`void* arcs::common_interface::RobotManage::d__` [protected]

Definition at line [1915](#) of file [robot_manage.h](#).

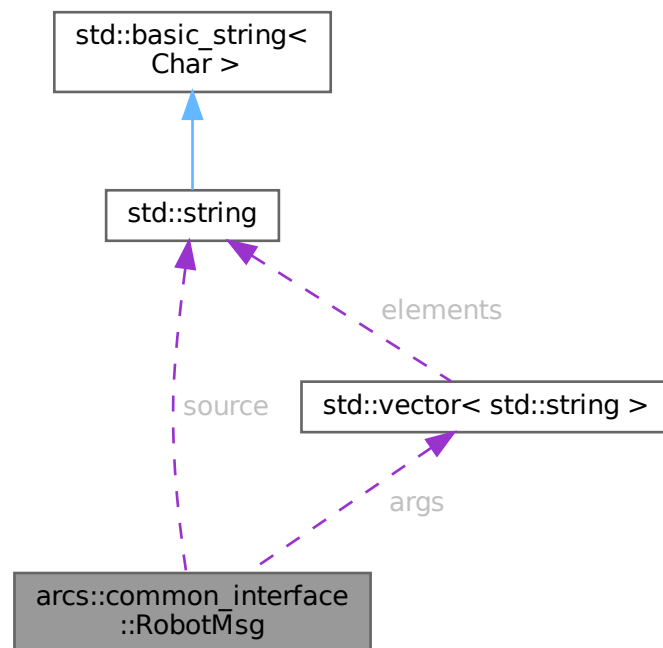
The documentation for this class was generated from the following file:

- `include/aubo/robot/robot_manage.h`

11.17 arcs::common_interface::RobotMsg Struct Reference

```
#include <type_def.h>
```

Collaboration diagram for arcs::common_interface::RobotMsg:



Public Attributes

- `uint64_t` [timestamp](#)
Timestamp, i.e., system time.
- `TraceLevel` [level](#)
Log level.
- `int` [code](#)
Error code.
- `std::string` [source](#)
Alias of the robot sending the message Can be found in `/root/arcs_ws/config/aubo_control.conf` The robot's alias can be found in the configuration file `/root/arcs_ws/config/aubo_control.conf`.
- `std::vector< std::string >` [args](#)
Robot parameters.

11.17.1 Detailed Description

Definition at line 828 of file [type_def.h](#).

11.17.2 Member Data Documentation

11.17.2.1 args

`std::vector<std::string> arcs::common_interface::RobotMsg::args`

Robot parameters.

Definition at line 843 of file [type_def.h](#).

11.17.2.2 code

`int arcs::common_interface::RobotMsg::code`

Error code.

Definition at line 833 of file [type_def.h](#).

11.17.2.3 level

[TraceLevel](#) `arcs::common_interface::RobotMsg::level`

Log level.

Definition at line 832 of file [type_def.h](#).

11.17.2.4 source

`std::string arcs::common_interface::RobotMsg::source`

Alias of the robot sending the message Can be found in `/root/arcs_ws/config/aubo_control.conf` The robot's alias can be found in the configuration file `/root/arcs_ws/config/aubo_control.conf`.

Definition at line 835 of file [type_def.h](#).

11.17.2.5 timestamp

`uint64_t arcs::common_interface::RobotMsg::timestamp`

Timestamp, i.e., system time.

Definition at line 830 of file [type_def.h](#).

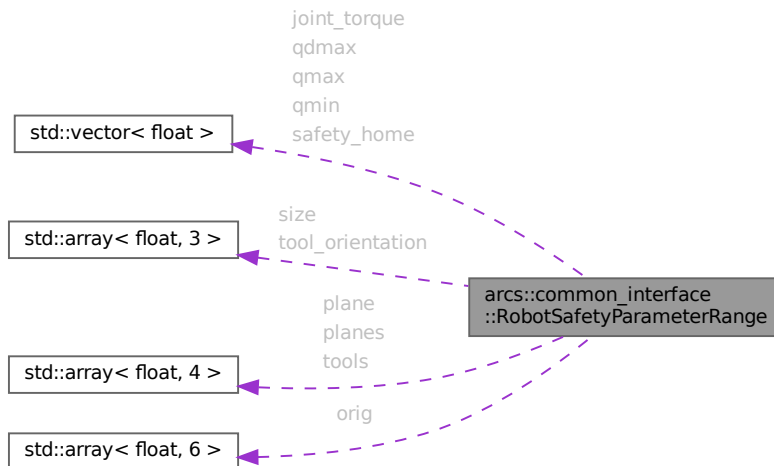
The documentation for this struct was generated from the following file:

- `include/aubo/type_def.h`

11.18 arcs::common_interface::RobotSafetyParameterRange Struct Reference

```
#include <type_def.h>
```

Collaboration diagram for arcs::common_interface::RobotSafetyParameterRange:



Public Member Functions

- [RobotSafetyParameterRange](#) ()

Public Attributes

- uint32_t [crc32](#) { 0 }
- struct {
 - float [power](#)
sum of joint torques times joint angular speeds
 - float [momentum](#)
robot momentum limit
 - float [stop_time](#)
stop time in milliseconds
 - float [stop_distance](#)
stop distance in meters
 - float [reduced_entry_time](#)
maximum time to enter reduced mode
 - float [reduced_entry_distance](#)
maximum distance to enter reduced mode (can be triggered by safety planes)
 - float [tcp_speed](#)
 - float [elbow_speed](#)
 - float [tcp_force](#)
 - float [elbow_force](#)
 - std::vector< float > [qmin](#)
 - std::vector< float > [qmax](#)

```

std::vector< float > qdmax
std::vector< float > joint_torque
Vector3f tool_orientation
float tool_deviation
Vector4f planes [SAFETY_PLANES_NUM]
    int restrict_elbow [SAFETY_PLANES_NUM]
        x,y,z,displacement
    } params [SAFETY_PARAM_SELECT_NUM]

```

At most 2 sets of parameters can be saved, default is the 0th set.

- struct {
 - Vector4f plane
 - int restrict_elbow
 - x,y,z,displacement
 } trigger_planes [SAFETY_PLANES_NUM]

 8 trigger planes
- struct {
 - Vector6f orig
 - origin of the cubic (x,y,z,rx,ry,rz)
 - Vector3f size
 - size of the cubic (x,y,z)
 - int restrict_elbow
 } cubic [SAFETY_CUBIC_NUM]

- 10 safety spaces
- Vector4f tools [TOOL_CONFIGURATION_NUM]
 - 3 tools
- float tool_inclination
 - x,y,z,radius
- float tool_azimuth { 0. }
 - azimuth angle
- std::vector< float > safety_home
- uint32_t safety_input_emergency_stop

Configurable IO input and output safety functions.

- uint32_t safety_input_safeguard_stop
- uint32_t safety_input_safeguard_reset
- uint32_t safety_input_auto_safeguard_stop
- uint32_t safety_input_auto_safeguard_reset
- uint32_t safety_input_three_position_switch
- uint32_t safety_input_operational_mode
- uint32_t safety_input_reduced_mode
- uint32_t safety_input_handguide
- uint32_t safety_output_emergency_stop
- uint32_t safety_output_not_emergency_stop
- uint32_t safety_output_robot_moving
- uint32_t safety_output_robot_steady
- uint32_t safety_output_reduced_mode
- uint32_t safety_output_not_reduced_mode
- uint32_t safety_output_safe_home
- uint32_t safety_output_robot_not_stopping
- uint32_t safety_output_safetyguard_stop
- int tp_3pe_for_handguide

Whether to use the three-position switch of the teach pendant as a hand guiding function switch.

- int allow_manual_high_speed

11.18.1 Detailed Description

Definition at line 37 of file [type_def.h](#).

11.18.2 Constructor & Destructor Documentation

11.18.2.1 RobotSafetyParameterRange()

arcs::common_interface::RobotSafetyParameterRange::RobotSafetyParameterRange () [inline]

Definition at line 39 of file [type_def.h](#).

References [allow_manual_high_speed](#), [cubic](#), [joint_torque](#), [params](#), [qddmax](#), [qmax](#), [qmin](#), [SAFETY_CUBIC_NUM](#), [safety_home](#), [safety_input_auto_safeguard_reset](#), [safety_input_auto_safeguard_stop](#), [safety_input_emergency_stop](#), [safety_input_handguide](#), [safety_input_operational_mode](#), [safety_input_reduced_mode](#), [safety_input_safeguard_reset](#), [safety_input_safeguard_stop](#), [safety_input_three_position_switch](#), [safety_output_emergency_stop](#), [safety_output_not_emergency_stop](#), [safety_output_not_reduced_mode](#), [safety_output_reduced_mode](#), [safety_output_robot_moving](#), [safety_output_robot_not_stopping](#), [safety_output_robot_steady](#), [safety_output_safe_home](#), [safety_output_safeguard_stop](#), [SAFETY_PARAM_SELECT_NUM](#), [SAFETY_PLANES_NUM](#), [tool_azimuth](#), [TOOL_CONFIGURATION_NUM](#), [tool_inclination](#), [tools](#), [tp_3pe_for_handguide](#), and [trigger_planes](#).

11.18.3 Member Data Documentation

11.18.3.1 allow_manual_high_speed

int arcs::common_interface::RobotSafetyParameterRange::allow_manual_high_speed

Definition at line 177 of file [type_def.h](#).

Referenced by [RobotSafetyParameterRange\(\)](#).

11.18.3.2 crc32

uint32_t arcs::common_interface::RobotSafetyParameterRange::crc32 { 0 }

Definition at line 105 of file [type_def.h](#).

11.18.3.3 [struct]

struct { ... } arcs::common_interface::RobotSafetyParameterRange::cubic[SAFETY_CUBIC_NUM]

10 safety spaces

Referenced by [RobotSafetyParameterRange\(\)](#).

11.18.3.4 elbow_force

float arcs::common_interface::RobotSafetyParameterRange::elbow_force

Definition at line 120 of file [type_def.h](#).

11.18.3.5 elbow_speed

float arcs::common_interface::RobotSafetyParameterRange::elbow_speed

Definition at line 118 of file [type_def.h](#).

11.18.3.6 joint_torque

std::vector<float> arcs::common_interface::RobotSafetyParameterRange::joint_torque

Definition at line 124 of file [type_def.h](#).

Referenced by [RobotSafetyParameterRange\(\)](#).

11.18.3.7 momentum

float arcs::common_interface::RobotSafetyParameterRange::momentum

robot momentum limit

Definition at line 112 of file [type_def.h](#).

11.18.3.8 orig

[Vector6f](#) arcs::common_interface::RobotSafetyParameterRange::orig

origin of the cubic (x,y,z,rx,ry,rz)

Definition at line 140 of file [type_def.h](#).

11.18.3.9 [struct]

struct { ... } arcs::common_interface::RobotSafetyParameterRange::params[[SAFETY_PARAM_SELECT_NUM](#)]

At most 2 sets of parameters can be saved, default is the 0th set.

Referenced by [RobotSafetyParameterRange\(\)](#).

11.18.3.10 plane

[Vector4f](#) arcs::common_interface::RobotSafetyParameterRange::plane

Definition at line 134 of file [type_def.h](#).

11.18.3.11 planes

[Vector4f](#) arcs::common_interface::RobotSafetyParameterRange::planes[[SAFETY_PLANES_NUM](#)]

Definition at line [127](#) of file [type_def.h](#).

11.18.3.12 power

float arcs::common_interface::RobotSafetyParameterRange::power

sum of joint torques times joint angular speeds

Definition at line [111](#) of file [type_def.h](#).

11.18.3.13 qdmax

std::vector<float> arcs::common_interface::RobotSafetyParameterRange::qdmax

Definition at line [123](#) of file [type_def.h](#).

Referenced by [RobotSafetyParameterRange\(\)](#).

11.18.3.14 qmax

std::vector<float> arcs::common_interface::RobotSafetyParameterRange::qmax

Definition at line [122](#) of file [type_def.h](#).

Referenced by [RobotSafetyParameterRange\(\)](#).

11.18.3.15 qmin

std::vector<float> arcs::common_interface::RobotSafetyParameterRange::qmin

Definition at line [121](#) of file [type_def.h](#).

Referenced by [RobotSafetyParameterRange\(\)](#).

11.18.3.16 reduced_entry_distance

float arcs::common_interface::RobotSafetyParameterRange::reduced_entry_distance

maximum distance to enter reduced mode (can be triggered by safety planes)

Definition at line [116](#) of file [type_def.h](#).

11.18.3.17 reduced_entry_time

float arcs::common_interface::RobotSafetyParameterRange::reduced_entry_time

maximum time to enter reduced mode

Definition at line 115 of file [type_def.h](#).

11.18.3.18 restrict_elbow

int arcs::common_interface::RobotSafetyParameterRange::restrict_elbow

x,y,z,displacement

Definition at line 128 of file [type_def.h](#).

11.18.3.19 safety_home

std::vector<float> arcs::common_interface::RobotSafetyParameterRange::safety_home

Definition at line 152 of file [type_def.h](#).

Referenced by [RobotSafetyParameterRange\(\)](#).

11.18.3.20 safety_input_auto_safeguard_reset

uint32_t arcs::common_interface::RobotSafetyParameterRange::safety_input_auto_safeguard_reset

Definition at line 160 of file [type_def.h](#).

Referenced by [RobotSafetyParameterRange\(\)](#).

11.18.3.21 safety_input_auto_safeguard_stop

uint32_t arcs::common_interface::RobotSafetyParameterRange::safety_input_auto_safeguard_stop

Definition at line 159 of file [type_def.h](#).

Referenced by [RobotSafetyParameterRange\(\)](#).

11.18.3.22 safety_input_emergency_stop

uint32_t arcs::common_interface::RobotSafetyParameterRange::safety_input_emergency_stop

Configurable IO input and output safety functions.

Definition at line 156 of file [type_def.h](#).

Referenced by [RobotSafetyParameterRange\(\)](#).

11.18.3.23 safety_input_handguide

uint32_t arcs::common_interface::RobotSafetyParameterRange::safety_input_handguide

Definition at line 164 of file [type_def.h](#).

Referenced by [RobotSafetyParameterRange\(\)](#).

11.18.3.24 safety_input_operational_mode

uint32_t arcs::common_interface::RobotSafetyParameterRange::safety_input_operational_mode

Definition at line 162 of file [type_def.h](#).

Referenced by [RobotSafetyParameterRange\(\)](#).

11.18.3.25 safety_input_reduced_mode

uint32_t arcs::common_interface::RobotSafetyParameterRange::safety_input_reduced_mode

Definition at line 163 of file [type_def.h](#).

Referenced by [RobotSafetyParameterRange\(\)](#).

11.18.3.26 safety_input_safeguard_reset

uint32_t arcs::common_interface::RobotSafetyParameterRange::safety_input_safeguard_reset

Definition at line 158 of file [type_def.h](#).

Referenced by [RobotSafetyParameterRange\(\)](#).

11.18.3.27 safety_input_safeguard_stop

uint32_t arcs::common_interface::RobotSafetyParameterRange::safety_input_safeguard_stop

Definition at line 157 of file [type_def.h](#).

Referenced by [RobotSafetyParameterRange\(\)](#).

11.18.3.28 safety_input_three_position_switch

uint32_t arcs::common_interface::RobotSafetyParameterRange::safety_input_three_position_switch

Definition at line 161 of file [type_def.h](#).

Referenced by [RobotSafetyParameterRange\(\)](#).

11.18.3.29 safety__output__emergency__stop

uint32_t arcs::common_interface::RobotSafetyParameterRange::safety__output__emergency__stop

Definition at line 166 of file [type_def.h](#).

Referenced by [RobotSafetyParameterRange\(\)](#).

11.18.3.30 safety__output__not__emergency__stop

uint32_t arcs::common_interface::RobotSafetyParameterRange::safety__output__not__emergency__stop

Definition at line 167 of file [type_def.h](#).

Referenced by [RobotSafetyParameterRange\(\)](#).

11.18.3.31 safety__output__not__reduced__mode

uint32_t arcs::common_interface::RobotSafetyParameterRange::safety__output__not__reduced__mode

Definition at line 171 of file [type_def.h](#).

Referenced by [RobotSafetyParameterRange\(\)](#).

11.18.3.32 safety__output__reduced__mode

uint32_t arcs::common_interface::RobotSafetyParameterRange::safety__output__reduced__mode

Definition at line 170 of file [type_def.h](#).

Referenced by [RobotSafetyParameterRange\(\)](#).

11.18.3.33 safety__output__robot__moving

uint32_t arcs::common_interface::RobotSafetyParameterRange::safety__output__robot__moving

Definition at line 168 of file [type_def.h](#).

Referenced by [RobotSafetyParameterRange\(\)](#).

11.18.3.34 safety__output__robot__not__stopping

uint32_t arcs::common_interface::RobotSafetyParameterRange::safety__output__robot__not__stopping

Definition at line 173 of file [type_def.h](#).

Referenced by [RobotSafetyParameterRange\(\)](#).

11.18.3.35 safety_output_robot_steady

uint32_t arcs::common_interface::RobotSafetyParameterRange::safety_output_robot_steady

Definition at line 169 of file [type_def.h](#).

Referenced by [RobotSafetyParameterRange\(\)](#).

11.18.3.36 safety_output_safe_home

uint32_t arcs::common_interface::RobotSafetyParameterRange::safety_output_safe_home

Definition at line 172 of file [type_def.h](#).

Referenced by [RobotSafetyParameterRange\(\)](#).

11.18.3.37 safety_output_safetyguard_stop

uint32_t arcs::common_interface::RobotSafetyParameterRange::safety_output_safetyguard_stop

Definition at line 174 of file [type_def.h](#).

Referenced by [RobotSafetyParameterRange\(\)](#).

11.18.3.38 size

[Vector3f](#) arcs::common_interface::RobotSafetyParameterRange::size

size of the cubic (x,y,z)

Definition at line 141 of file [type_def.h](#).

11.18.3.39 stop_distance

float arcs::common_interface::RobotSafetyParameterRange::stop_distance

stop distance in meters

Definition at line 114 of file [type_def.h](#).

11.18.3.40 stop_time

float arcs::common_interface::RobotSafetyParameterRange::stop_time

stop time in milliseconds

Definition at line 113 of file [type_def.h](#).

11.18.3.41 tcp_force

float arcs::common_interface::RobotSafetyParameterRange::tcp_force

Definition at line 119 of file [type_def.h](#).

11.18.3.42 tcp_speed

float arcs::common_interface::RobotSafetyParameterRange::tcp_speed

Definition at line 117 of file [type_def.h](#).

11.18.3.43 tool_azimuth

float arcs::common_interface::RobotSafetyParameterRange::tool_azimuth { 0. }

azimuth angle

Definition at line 151 of file [type_def.h](#).

Referenced by [RobotSafetyParameterRange\(\)](#).

11.18.3.44 tool_deviation

float arcs::common_interface::RobotSafetyParameterRange::tool_deviation

Definition at line 126 of file [type_def.h](#).

11.18.3.45 tool_inclination

float arcs::common_interface::RobotSafetyParameterRange::tool_inclination

Initial value:

```
{  
    0.  
}
```

x,y,z,radius

inclination angle

Definition at line 148 of file [type_def.h](#).

Referenced by [RobotSafetyParameterRange\(\)](#).

11.18.3.46 tool_orientation

[Vector3f](#) arcs::common_interface::RobotSafetyParameterRange::tool_orientation

Definition at line 125 of file [type_def.h](#).

11.18.3.47 tools

[Vector4f](#) arcs::common_interface::RobotSafetyParameterRange::tools[[TOOL_CONFIGURATION_NUM](#)]

3 tools

Definition at line [146](#) of file [type_def.h](#).

Referenced by [RobotSafetyParameterRange\(\)](#).

11.18.3.48 tp_3pe_for_handguide

int arcs::common_interface::RobotSafetyParameterRange::tp_3pe_for_handguide

Whether to use the three-position switch of the teach pendant as a hand guiding function switch.

Definition at line [176](#) of file [type_def.h](#).

Referenced by [RobotSafetyParameterRange\(\)](#).

11.18.3.49 [struct]

struct { ... } arcs::common_interface::RobotSafetyParameterRange::trigger_planes[[SAFETY_PLANES_NUM](#)]

8 trigger planes

Referenced by [RobotSafetyParameterRange\(\)](#).

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

- [include/aubo/type_def.h](#)

11.19 arcs::common_interface::RobotState Class Reference

```
#include <robot_state.h>
```

Public Member Functions

- [RobotState](#) ()
- virtual [~RobotState](#) ()
- [RobotModeType](#) [getRobotModeType](#) ()
Get the robot mode state
- [SafetyModeType](#) [getSafetyModeType](#) ()
Get the safety mode
- bool [isPowerOn](#) ()
Get the robot power-on state
- bool [isSteady](#) ()
Whether the robot has stopped
- bool [isCollisionOccurred](#) ()
Whether a collision has occurred
- bool [isWithinSafetyLimits](#) ()
Whether the robot is within safety limits
- std::vector< double > [getTcpPose](#) ()
Get the current TCP pose.
- std::vector< double > [getActualTcpOffset](#) ()
Get the current TCP offset, which is the TCP offset used by the pose returned from [getTcpPose](#)
- std::vector< double > [getTargetTcpPose](#) ()
Get the next target waypoint.
- std::vector< double > [getToolPose](#) ()
Get the tool pose (without TCP offset)
- std::vector< double > [getTcpSpeed](#) ()
Get the TCP speed
- std::vector< double > [getTcpForce](#) ()
Get the TCP force/torque
- std::vector< double > [getElbowPosistion](#) ()
Get the position of the elbow
- std::vector< double > [getElbowVelocity](#) ()
Get the elbow velocity
- std::vector< double > [getBaseForce](#) ()
Get the base force/torque
- std::vector< double > [getTcpTargetPose](#) ()
Get the last sent TCP target pose
- std::vector< double > [getTcpTargetSpeed](#) ()
Get the TCP target speed
- std::vector< double > [getTcpTargetForce](#) ()
Get the TCP target force/torque
- std::vector< [JointStateType](#) > [getJointState](#) ()
Get the joint state of the manipulator
- std::vector< [JointServoModeType](#) > [getJointServoMode](#) ()
Get the servo state of the joints
- std::vector< double > [getJointPositions](#) ()
Get the joint positions of the manipulator
- std::vector< double > [getJointPositionsHistory](#) (int steps)
Get the historical joint positions of the manipulator
- std::vector< double > [getJointSpeeds](#) ()
Get the joint speeds of the manipulator
- std::vector< double > [getJointAccelerations](#) ()

- Get the joint accelerations of the manipulator
- `std::vector< double > getJointTorqueSensors ()`
- Get the joint torques of the manipulator
- `std::vector< double > getJointContactTorques ()`
- Get the joint contact torques (external torques) of the manipulator
- `std::vector< double > getJointGravityTorques ()`
- Get the joint gravity torque of the manipulator
- `std::vector< double > getBaseForceSensor ()`
- Get the base force sensor readings
- `std::vector< double > getTcpForceSensors ()`
- Get the TCP force sensor readings
- `std::vector< double > getJointCurrents ()`
- Get the joint currents of the manipulator
- `std::vector< double > getJointVoltages ()`
- Get the joint voltages of the manipulator
- `std::vector< double > getJointTemperatures ()`
- Get the joint temperatures of the manipulator
- `std::vector< std::string > getJointUniqueIds ()`
- Get the globally unique IDs of the joints
- `std::vector< int > getJointFirmwareVersions ()`
- Get the joint firmware versions
- `std::vector< int > getJointHardwareVersions ()`
- Get the joint hardware versions
- `std::string getMasterBoardUniqueId ()`
- Get the globally unique ID of the MasterBoard
- `int getMasterBoardFirmwareVersion ()`
- Get the MasterBoard firmware version
- `int getMasterBoardHardwareVersion ()`
- Get the MasterBoard hardware version
- `std::string getSlaveBoardUniqueId ()`
- Get the globally unique ID of the SlaveBoard
- `int getSlaveBoardFirmwareVersion ()`
- Get the SlaveBoard firmware version
- `int getSlaveBoardHardwareVersion ()`
- Get the SlaveBoard hardware version
- `std::string getToolUniqueId ()`
- Get the globally unique ID of the tool
- `int getToolFirmwareVersion ()`
- Get the tool firmware version
- `int getToolHardwareVersion ()`
- Get the tool hardware version
- `int getToolCommMode ()`
- Get the tool communication mode
- `std::string getPedestalUniqueId ()`
- Get the globally unique ID of the pedestal
- `int getPedestalFirmwareVersion ()`
- Get the pedestal firmware version
- `int getPedestalHardwareVersion ()`
- Get the pedestal hardware version
- `std::vector< double > getJointTargetPositions ()`
- Get the target joint positions (angles) of the manipulator

- `std::vector< double > getJointTargetSpeeds ()`
Get the target joint speeds of the manipulator
- `std::vector< double > getJointTargetAccelerations ()`
Get the target joint accelerations of the manipulator
- `std::vector< double > getJointTargetTorques ()`
Get the target joint torques of the manipulator
- `std::vector< double > getJointTargetCurrents ()`
Get the target joint currents of the manipulator
- `bool isTeachPendantEnabled ()`
Get whether the teach pendant is enabled.
- `bool isToolFlangeEnabled ()`
Get whether the tool flange is enabled.
- `double getControlBoxTemperature ()`
Get the control box temperature
- `double getControlBoxHumidity ()`
Get the control box humidity
- `double getMainVoltage ()`
Get the main bus voltage
- `double getMainCurrent ()`
Get the main bus current
- `double getRobotVoltage ()`
Get the robot voltage
- `double getRobotCurrent ()`
Get the robot current
- `int getSlowDownLevel ()`
Get the robot slow down level
- `bool getTcpForceSensorStatus (const std::string &name)`
Get the communication status of the tool force sensor

Protected Attributes

- `void * d_`

11.19.1 Detailed Description

Definition at line 20 of file [robot_state.h](#).

11.19.2 Constructor & Destructor Documentation

11.19.2.1 RobotState()

`arcs::common_interface::RobotState::RobotState ()`

11.19.2.2 ~RobotState()

`virtual arcs::common_interface::RobotState::~~RobotState ()` [virtual]

11.19.3 Member Function Documentation

11.19.3.1 getBaseForceSensor()

std::vector< double > arcs::common_interface::RobotState::getBaseForceSensor ()

Get the base force sensor readings

Returns

Base force sensor readings

Exceptions

arcs::common_interface::AuboException	
---------------------------------------	--

Python function prototype

getBaseForceSensor(self: pyaubo_sdk.RobotState) -> List[float]

Lua function prototype

getBaseForceSensor() -> table

Lua example

BaseForceSensor = getBaseForceSensor()

JSON-RPC request example

```
{"jsonrpc": "2.0", "method": "rob1.RobotState.getBaseForceSensor", "params": [], "id": 1}
```

JSON-RPC response example

```
{"id": 1, "jsonrpc": "2.0", "result": [0.0, 0.0, 0.0, 0.0, 0.0, 0.0]}
```

11.19.4 Member Data Documentation

11.19.4.1 d__

void* arcs::common_interface::RobotState::d__ [protected]

Definition at line 3067 of file [robot_state.h](#).

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

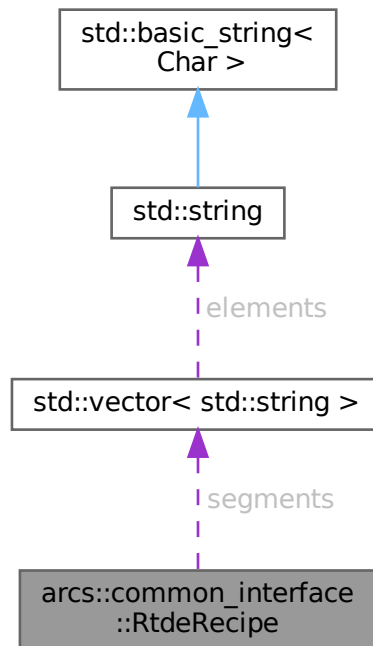
- include/aubo/robot/[robot_state.h](#)

11.20 arcs::common_interface::RtdeRecipe Struct Reference

RTDE menu.

```
#include <type_def.h>
```

Collaboration diagram for arcs::common_interface::RtdeRecipe:



Public Attributes

- bool `to_server`
Input/Output.
- int `chanel`
Channel.
- double `frequency`
Update frequency.
- int `trigger`
Trigger method (this feature is not yet implemented): 0 - Periodic; 1 - Change.
- `std::vector< std::string >` `segments`
segment list

11.20.1 Detailed Description

RTDE menu.

Definition at line 867 of file `type_def.h`.

11.20.2 Member Data Documentation

11.20.2.1 chanel

int arcs::common_interface::RtdeRecipe::chanel

Channel.

Definition at line 870 of file [type_def.h](#).

11.20.2.2 frequency

double arcs::common_interface::RtdeRecipe::frequency

Update frequency.

Definition at line 871 of file [type_def.h](#).

11.20.2.3 segments

std::vector<std::string> arcs::common_interface::RtdeRecipe::segments

segment list

Definition at line 875 of file [type_def.h](#).

11.20.2.4 to_server

bool arcs::common_interface::RtdeRecipe::to_server

Input/Output.

Definition at line 869 of file [type_def.h](#).

11.20.2.5 trigger

int arcs::common_interface::RtdeRecipe::trigger

Trigger method (this feature is not yet implemented): 0 - Periodic; 1 - Change.

Definition at line 872 of file [type_def.h](#).

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

- [include/aubo/type_def.h](#)

11.21 arcs::common_interface::RuntimeMachine Class Reference

```
#include <runtime_machine.h>
```

Public Member Functions

- [RuntimeMachine](#) ()
- virtual [~RuntimeMachine](#) ()
- int [newTask](#) (bool daemon=false)
Returns the task_id
- int [deleteTask](#) (int tid)
Delete a task, which will terminate any ongoing motion.
- int [detachTask](#) (int tid)
Wait for the task to finish naturally
- bool [isTaskAlive](#) (int tid)
Check if the task is alive
- int [getTaskQueueSize](#) (int tid)
Get the number of cached instructions in the task
- int [switchTask](#) (int tid)
Switch the current thread.
- int [setLabel](#) (int lineno, const std::string &comment)
Mark the line number and comment of the recorded instruction
- ARCS_DEPRECATED int [setPlanContext](#) (int tid, int lineno, const std::string &comment)
Add a comment to the aubo_control log Use setLabel instead
- int [nop](#) ()
No operation
- std::tuple< std::string, std::string > [getExecutionStatus](#) ()
Get the execution status of time-consuming interfaces (INST), such as setPersistentParameters
- std::tuple< std::string, std::string, int > [getExecutionStatus1](#) ()
- int [gotoLine](#) (int lineno)
Jump to the specified line number
- std::tuple< int, int, std::string > [getPlanContext](#) (int tid=-1)
Get the current runtime context
- std::tuple< int, int, std::string > [getAdvancePlanContext](#) (int tid=-1)
Get the context information of the advance planner
- int [getAdvancePtr](#) (int tid=-1)
Get the program pointer of AdvanceRun
- int [getMainPtr](#) (int tid=-1)
Get the program pointer of robot motion
- int [getInterpPtr](#) (int tid)
Get the pointer of the most recently interpreted instruction
- int [loadProgram](#) (const std::string &program)
Load a local project file.
- int [preloadProgram](#) (int index, const std::string &program)
Preload project file
- std::string [getPreloadProgram](#) (int index)
Get the name of the preloaded project file.
- int [clearPreloadPrograms](#) ()
Clear all preloaded project files.
- int [runProgram](#) ()
Run the already loaded project file
- int [start](#) ()
Start the runtime
- int [stop](#) ()
Stop the runtime, i.e., stop script execution.

- int [abort](#) ()
Abort robot operation.
- int [pause](#) ()
Pause the interpreter
- int [step](#) ()
Execute a single step
- int [resume](#) ()
Resume the interpreter
- int [arbitraryResume](#) ()
Resume the interpreter
- int [setResumeWait](#) (bool wait)
Wait for the previous sequence to complete before resuming the interpreter
- int [enterCritical](#) (double timeout)
The abort command is deferred during critical sections to avoid interrupting internal instructions.
- int [exitCritical](#) ()
Exit the critical section
- ARCS_DEPRECATED [RuntimeState getStatus](#) ()
Get the status of the planner
- [RuntimeState getRuntimeState](#) ()
- int [setBreakPoint](#) (int lineno)
Set a breakpoint
- int [removeBreakPoint](#) (int lineno)
Remove a breakpoint
- int [clearBreakPoints](#) ()
Clear all breakpoints
- int [timerStart](#) (const std::string &name)
Start the timer
- int [timerStop](#) (const std::string &name)
Stop the timer
- int [timerReset](#) (const std::string &name)
Reset the timer
- int [timerDelete](#) (const std::string &name)
Delete the timer
- double [getTimer](#) (const std::string &name)
Get the timer value
- int [triggBegin](#) (double distance, double delay)
Start configuring trigger
- int [triggEnd](#) ()
End configuring trigger
- int [triggInterrupt](#) (double distance, double delay)
Returns the automatically assigned interrupt number
- std::vector< int > [getTriggInterrupts](#) ()
Get the list of all interrupt numbers

Protected Attributes

- void * [d_](#)

11.21.1 Detailed Description

Definition at line 19 of file [runtime_machine.h](#).

11.21.2 Constructor & Destructor Documentation

11.21.2.1 RuntimeMachine()

`arcs::common_interface::RuntimeMachine::RuntimeMachine ()`

11.21.2.2 ~RuntimeMachine()

`virtual arcs::common_interface::RuntimeMachine::~~RuntimeMachine () [virtual]`

11.21.3 Member Function Documentation

11.21.3.1 getExecutionStatus1()

`std::tuple< std::string, std::string, int > arcs::common_interface::RuntimeMachine::getExecutionStatus1 ()`

11.21.3.2 getRuntimeState()

[RuntimeState](#) `arcs::common_interface::RuntimeMachine::getRuntimeState ()`

11.21.4 Member Data Documentation

11.21.4.1 d__

`void* arcs::common_interface::RuntimeMachine::d__ [protected]`

Definition at line [1617](#) of file [runtime_machine.h](#).

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

- `include/aubo/runtime_machine.h`

11.22 arcs::common_interface::Serial Class Reference

```
#include <serial.h>
```


Public Member Functions

- [Serial](#) ()
- virtual [~Serial](#) ()
- int [serialOpen](#) (const std::string &device, int baud, float stop_bits, int even, const std::string &serial_name="serial_0")
Open TCP/IP ethernet communication serial
- int [serialClose](#) (const std::string &serial_name="serial_0")
Close TCP/IP serial communication Close down the serial connection to the server.
- int [serialReadByte](#) (const std::string &variable, const std::string &serial_name="serial_0")
Reads a number of bytes from the serial.
- int [serialReadByteList](#) (int number, const std::string &variable, const std::string &serial_name="serial_0")
Reads a number of bytes from the serial.
- int [serialReadString](#) (const std::string &variable, const std::string &serial_name="serial_0", const std::string &prefix="", const std::string &suffix="", bool interpret_escape=false)
Reads all data from the serial and returns the data as a string.
- int [serialSendByte](#) (char value, const std::string &serial_name="serial_0")
Sends a byte to the server Sends the byte through the serial.
- int [serialSendInt](#) (int value, const std::string &serial_name="serial_0")
Sends an int (int32_t) to the server Sends the int through the serial.
- int [serialSendLine](#) (const std::string &str, const std::string &serial_name="serial_0")
Sends a string with a newline character to the server Sends the string <str> through the serial in ASCII coding, appending a newline at the end.
- int [serialSendString](#) (const std::string &str, const std::string &serial_name="serial_0")
Sends a string to the server Sends the string <str> through the serial in ASCII coding.
- int [serialSendAllString](#) (bool is_check, const std::vector< char > &str, const std::string &serial_name="serial_0")

Protected Attributes

- void * [d_](#)

11.22.1 Detailed Description

Definition at line 21 of file [serial.h](#).

11.22.2 Constructor & Destructor Documentation

11.22.2.1 Serial()

arcs::common_interface::Serial::Serial ()

11.22.2.2 ~Serial()

virtual arcs::common_interface::Serial::~Serial () [virtual]

11.22.3 Member Data Documentation

11.22.3.1 d_

void* arcs::common_interface::Serial::d_ [protected]

Definition at line 405 of file [serial.h](#).

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

- include/aubo/[serial.h](#)

11.23 arcs::common_interface::Socket Class Reference

```
#include <socket.h>
```

Public Member Functions

- [Socket](#) ()
- virtual [~Socket](#) ()
- int [socketOpen](#) (const std::string &address, int port, const std::string &socket_name="socket_0")
Open TCP/IP ethernet communication socket
- int [socketClose](#) (const std::string &socket_name="socket_0")
Closes TCP/IP socket communication Closes down the socket connection to the server.
- int [socketReadAsciiFloat](#) (int number, const std::string &variable, const std::string &socket_name="socket_0")
Reads a number of ascii formatted floats from the socket.
- int [socketReadBinaryInteger](#) (int number, const std::string &variable, const std::string &socket_name="socket_0")
Reads a number of 32 bit integers from the socket.
- int [socketReadByteList](#) (int number, const std::string &variable, const std::string &socket_name="socket_0")
Reads a number of bytes from the socket.
- int [socketReadString](#) (const std::string &variable, const std::string &socket_name="socket_0", const std::string &prefix="", const std::string &suffix="", bool interpret_escape=false)
Reads all data from the socket and returns the data as a string.
- int [socketReadAllString](#) (const std::string &variable, const std::string &socket_name="socket_0")
Reads all data from the socket and returns the data as a vector of chars.
- int [socketSendByte](#) (char value, const std::string &socket_name="socket_0")
Sends a byte to the server Sends the byte through the socket.
- int [socketSendInt](#) (int value, const std::string &socket_name="socket_0")
Sends an int (int32_t) to the server Sends the int through the socket.
- int [socketSendLine](#) (const std::string &str, const std::string &socket_name="socket_0")
Sends a string with a newline character to the server Sends the string <str> through the socket in ASCII coding.
- int [socketSendString](#) (const std::string &str, const std::string &socket_name="socket_0")
Sends a string to the server Sends the string <str> through the socket in ASCII coding.
- int [socketSendAllString](#) (bool is_check, const std::vector< char > &str, const std::string &socket_name="socket_0")
Sends all data in the given vector of chars to the server.
- bool [socketHasConnected](#) (const std::string &socket_name="socket_0")
Check if the socket is connected.

Protected Attributes

- void * [d__](#)

11.23.1 Detailed Description

Definition at line [22](#) of file [socket.h](#).

11.23.2 Constructor & Destructor Documentation

11.23.2.1 Socket()

arcs::common_interface::Socket::Socket ()

11.23.2.2 ~Socket()

virtual arcs::common_interface::Socket::~Socket () [virtual]

11.23.3 Member Data Documentation

11.23.3.1 d__

void* arcs::common_interface::Socket::d__ [protected]

Definition at line [640](#) of file [socket.h](#).

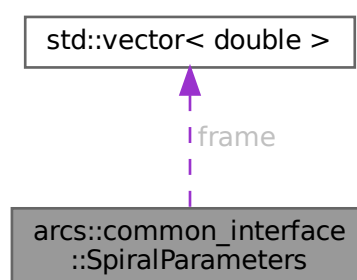
The documentation for this class was generated from the following file:

- include/aubo/[socket.h](#)

11.24 arcs::common_interface::SpiralParameters Struct Reference

```
#include <type_def.h>
```

Collaboration diagram for arcs::common_interface::SpiralParameters:



Public Attributes

- `std::vector< double > frame`
Reference point, the center point of the spiral and the reference coordinate system.
- `int plane`
Reference plane selection 0-XY 1-YZ 2-ZX.
- `double angle`
The angle of rotation, if positive, the robot rotates counterclockwise.
- `double spiral`
Positive outward.
- `double helix`
Positive upward.

11.24.1 Detailed Description

Definition at line [738](#) of file [type_def.h](#).

11.24.2 Member Data Documentation

11.24.2.1 `angle`

`double arcs::common_interface::SpiralParameters::angle`

The angle of rotation, if positive, the robot rotates counterclockwise.

Definition at line [742](#) of file [type_def.h](#).

11.24.2.2 `frame`

`std::vector<double> arcs::common_interface::SpiralParameters::frame`

Reference point, the center point of the spiral and the reference coordinate system.

Definition at line [740](#) of file [type_def.h](#).

11.24.2.3 `helix`

`double arcs::common_interface::SpiralParameters::helix`

Positive upward.

Definition at line [744](#) of file [type_def.h](#).

11.24.2.4 `plane`

`int arcs::common_interface::SpiralParameters::plane`

Reference plane selection 0-XY 1-YZ 2-ZX.

Definition at line [741](#) of file [type_def.h](#).

11.24.2.5 spiral

double arcs::common_interface::SpiralParameters::spiral

Positive outward.

Definition at line 743 of file [type_def.h](#).

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

- [include/aubo/type_def.h](#)

11.25 arcs::common_interface::SyncMove Class Reference

```
#include <sync_move.h>
```

Public Member Functions

- [SyncMove](#) ()
- virtual [~SyncMove](#) ()
- int [syncMoveOn](#) (const std::string &syncident, const [TaskSet](#) &taskset)
syncMoveOn is used to start synchronized movement mode.
- bool [syncMoveSegment](#) (int id)
Set the ID for the synchronized path segment.
- int [syncMoveOff](#) (const std::string &syncident)
syncMoveOff is used to end synchronized movement mode.
- int [syncMoveUndo](#) ()
syncMoveUndo is used to turn off synchronized movements, even if not all the other task programs execute the syncMoveUndo instruction.
- int [waitSyncTasks](#) (const std::string &syncident, const [TaskSet](#) &taskset)
waitSyncTasks is used to synchronize several task programs at a specific point in the program.
- bool [isSyncMoveOn](#) ()
isSyncMoveOn is used to tell if the mechanical unit group is in synchronized movement mode.
- int [syncMoveSuspend](#) ()
Suspend synchronized movement mode.
- int [syncMoveResume](#) ()
Resume synchronized movement mode.
- int [frameAdd](#) (const std::string &name, const std::vector< double > &pose, const std::string &ref←_name)
Add a frame with the name "name" initialized at the specified pose expressed in the ref_frame coordinate frame.
- int [frameAttach](#) (const std::string &child, const std::string &parent)
Attaches the child frame to the parent world model object.
- int [frameDeleteAll](#) ()
Delete all frames that have been added to the world model.
- int [frameDelete](#) (const std::string &name)
Delete the frame with name from the world model.
- int [frameMove](#) (const std::string &name, const std::vector< double > &pose, const std::string &ref←_name)

- Changes the placement of the coordinate frame named `name` to the new placement given by `pose` that is defined in the `ref_name` coordinate frame.
- `std::vector< double > frameGetPose` (`const std::string &name`, `const std::string &rel_frame`, `const std::string &ref_frame`)
Get the pose of the `name` frame relative to the `rel_frame` frame but expressed in the coordinates of the `ref_frame` frame.
 - `std::vector< double > frameConvertPose` (`const std::vector< double > &pose`, `const std::string &from_frame`, `const std::string &to_frame`)
Convert pose from `from_frame` to `to_frame`.
 - `bool frameExist` (`const std::string &name`)
Queries for the existence of a frame by the given name.
 - `std::string frameGetParent` (`const std::string &name`)
Get the parent of the frame named `name` in the world model.
 - `std::vector< std::string > frameGetChildren` (`const std::string &name`)
Returns a list of immediate child object frame names.
 - `int axisGroupAdd` (`const std::string &name`, `const std::vector< double > &pose`, `const std::string &ref_frame`)
Adds a new axis group with the given name to the world model.
 - `int axisGroupDelete` (`const std::string &name`)
Deletes the axis group with the given name from the world model.
 - `int axisGroupAddAxis` (`const std::string &group_name`, `const std::string &name`, `const std::string &parent`, `const std::vector< double > &pose`)
Adds an external axis with the given name to the axis group named `group_name`.
 - `int axisGroupUpdateAxis` (`const std::string &name`, `const std::vector< double > &pose`)
Updates the corresponding properties of axis with `name`.
 - `int axisGroupGetAxisIndex` (`const std::string &name`)
Returns the index of the axis with given `axis_name` in the RTDE target positions and actual positions arrays.
 - `std::string axisGroupGetAxisName` (`int index`)
Returns the name of the axis with the given `axis_index`.
 - `std::vector< double > axisGroupGetTargetPositions` (`const std::string &group_name`)
Returns the current target positions of the axis group with `group_name`.
 - `std::vector< double > axisGroupGetActualPositions` (`const std::string &group_name`)
Returns the current actual positions of the axis group with `group_name`.
 - `int axisGroupOffsetPositions` (`const std::string &group_name`, `const std::vector< double > &offset`)
Shifts the target and actual positions of the axis group `group_name` by the given `offset`.
 - `int axisGroupMoveJoint` (`const std::string &group_name`, `const std::vector< double > &q`, `double a`, `double v`)
Moves the axes of axis group named `group_name` to new positions `q`, using a trapezoidal velocity profile.
 - `int axisGroupSpeedJoint` (`const std::string &group_name`, `const std::vector< double > &qd`, `double a`, `double t`)
Accelerates the axes of axis group named `group_name` up to the target velocities `qd`.

Protected Attributes

- `void * d__`

11.25.1 Detailed Description

Definition at line 42 of file `sync_move.h`.

11.25.2 Constructor & Destructor Documentation

11.25.2.1 SyncMove()

arcs::common_interface::SyncMove::SyncMove ()

11.25.2.2 ~SyncMove()

virtual arcs::common_interface::SyncMove::~SyncMove () [virtual]

11.25.3 Member Data Documentation

11.25.3.1 d__

void* arcs::common_interface::SyncMove::d__ [protected]

Definition at line 1008 of file [sync_move.h](#).

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

- include/aubo/[sync_move.h](#)

11.26 arcs::common_interface::SystemInfo Class Reference

```
#include <system_info.h>
```

Public Member Functions

- [SystemInfo](#) ()
- virtual [~SystemInfo](#) ()
- int [getControlSoftwareVersionCode](#) ()
Get the controller software version code
- std::string [getControlSoftwareFullVersion](#) ()
Get the full controller software version
- int [getInterfaceVersionCode](#) ()
Get the interface version code
- std::string [getControlSoftwareBuildDate](#) ()
Get the controller software build date
- std::string [getControlSoftwareVersionHash](#) ()
Get the controller software git version
- uint64_t [getControlSystemTime](#) ()
Get the system time (software start time in nanoseconds)

Protected Attributes

- void * [d__](#)

11.26.1 Detailed Description

Definition at line 21 of file [system_info.h](#).

11.26.2 Constructor & Destructor Documentation

11.26.2.1 SystemInfo()

```
arcs::common_interface::SystemInfo::SystemInfo ()
```

11.26.2.2 ~SystemInfo()

```
virtual arcs::common_interface::SystemInfo::~~SystemInfo () [virtual]
```

11.26.3 Member Data Documentation

11.26.3.1 d__

```
void* arcs::common_interface::SystemInfo::d__ [protected]
```

Definition at line 346 of file [system_info.h](#).

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

- [include/aubo/system_info.h](#)

11.27 arcs::common_interface::Trace Class Reference

```
#include <trace.h>
```

Public Member Functions

- [Trace](#) ()
- virtual [~Trace](#) ()
- int [alarm](#) ([TraceLevel](#) level, int code, const std::vector< std::string > &args={})
Injects alarm information into the aubo_control log.
- int [textmsg](#) (const std::string &msg)
print message into log
- int [notify](#) (const std::string &msg)
Notify the system.
- int [popup](#) ([TraceLevel](#) level, const std::string &title, const std::string &msg, int mode)
Send a popup request to the connected RTDE client
- [RobotMsgVector](#) [peek](#) (size_t num, uint64_t last_time=0)
peek the latest AlarmInfo (after the last retrieval)

Protected Attributes

- void * [d_](#)

11.27.1 Detailed Description

Definition at line 22 of file [trace.h](#).

11.27.2 Constructor & Destructor Documentation

11.27.2.1 Trace()

arcs::common_interface::Trace::Trace ()

11.27.2.2 ~Trace()

virtual arcs::common_interface::Trace::~Trace () [virtual]

11.27.3 Member Function Documentation

11.27.3.1 notify()

```
int arcs::common_interface::Trace::notify (
    const std::string & msg)
```

Notify the system.

Parameters

msg	
-----	--

Returns

11.27.4 Member Data Documentation

11.27.4.1 d_

void* arcs::common_interface::Trace::d_ [protected]

Definition at line 229 of file [trace.h](#).

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

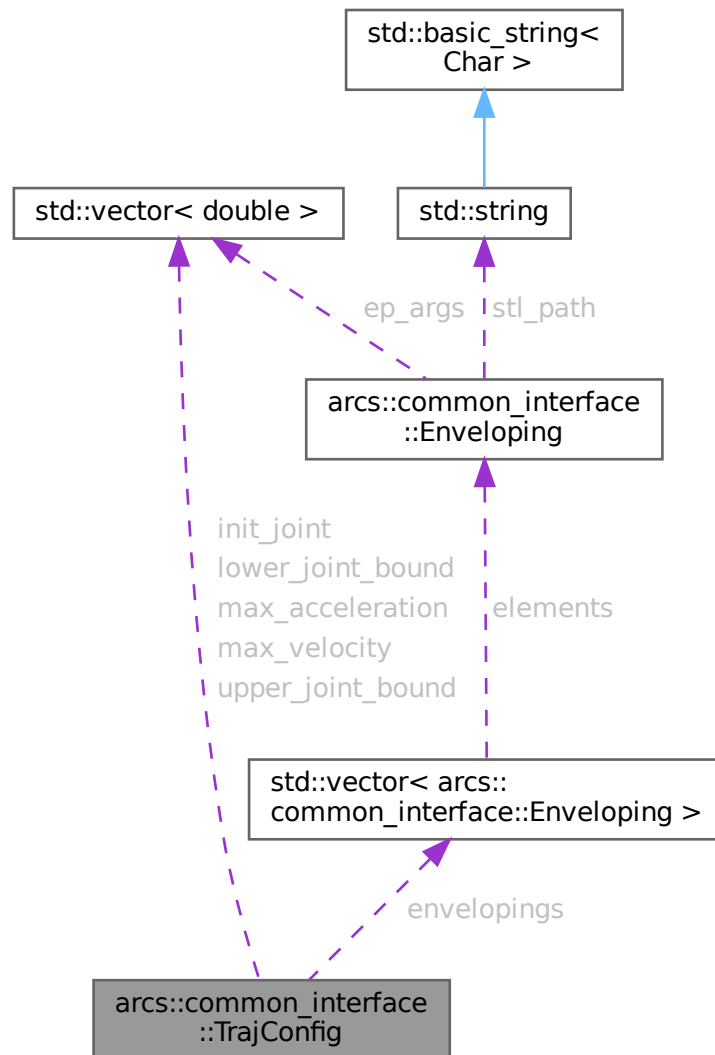
- include/aubo/[trace.h](#)

11.28 arcs::common_interface::TrajConfig Struct Reference

Trajectory configuration for payload identification.

```
#include <type_def.h>
```

Collaboration diagram for arcs::common_interface::TrajConfig:



Public Attributes

- `std::vector< Enveloping > envelopings`
- `PayloadIdentifyMoveAxis move_axis`
- `std::vector< double > init_joint`
- `std::vector< double > upper_joint_bound`
- `std::vector< double > lower_joint_bound`
- `std::vector< double > max_velocity`
- `std::vector< double > max_acceleration`

11.28.1 Detailed Description

Trajectory configuration for payload identification.

Definition at line 768 of file [type_def.h](#).

11.28.2 Member Data Documentation

11.28.2.1 envelopings

`std::vector<Enveloping> arcs::common_interface::TrajConfig::envelopings`

Definition at line 770 of file [type_def.h](#).

11.28.2.2 init_joint

`std::vector<double> arcs::common_interface::TrajConfig::init_joint`

Definition at line 772 of file [type_def.h](#).

11.28.2.3 lower_joint_bound

`std::vector<double> arcs::common_interface::TrajConfig::lower_joint_bound`

Definition at line 774 of file [type_def.h](#).

11.28.2.4 max_acceleration

`std::vector<double> arcs::common_interface::TrajConfig::max_acceleration`

Definition at line 776 of file [type_def.h](#).

11.28.2.5 max_velocity

`std::vector<double> arcs::common_interface::TrajConfig::max_velocity`

Definition at line 775 of file [type_def.h](#).

11.28.2.6 move_axis

`PayloadIdentifyMoveAxis arcs::common_interface::TrajConfig::move_axis`

Definition at line 771 of file [type_def.h](#).

11.28.2.7 upper_joint_bound

`std::vector<double> arcs::common_interface::TrajConfig::upper_joint_bound`

Definition at line 773 of file [type_def.h](#).

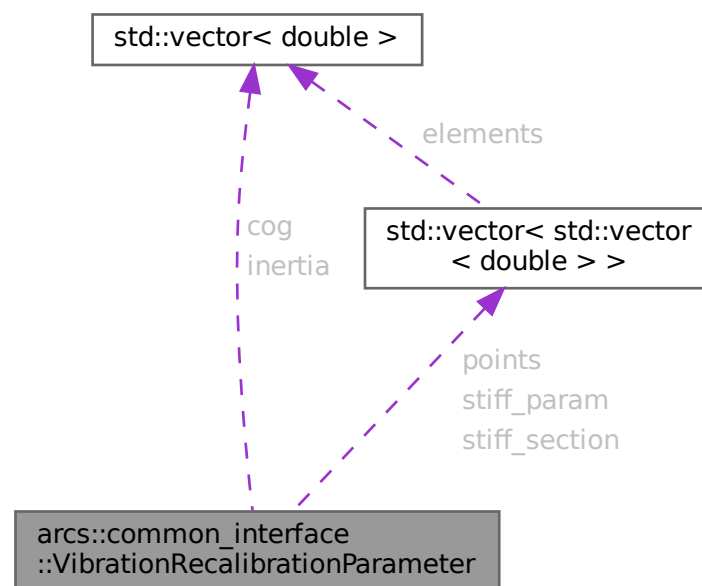
The documentation for this struct was generated from the following file:

- [include/aubo/type_def.h](#)

11.29 arcs::common_interface::VibrationRecalibrationParameter Struct Reference

```
#include <type_def.h>
```

Collaboration diagram for `arcs::common_interface::VibrationRecalibrationParameter`:



Public Attributes

- double [mass](#)
- `std::vector< double >` [cog](#)
- `std::vector< double >` [inertia](#)
- `std::vector< std::vector< double > >` [points](#)
- `std::vector< std::vector< double > >` [stiff_section](#)
- `std::vector< std::vector< double > >` [stiff_param](#)

11.29.1 Detailed Description

Definition at line 214 of file [type_def.h](#).

11.29.2 Member Data Documentation

11.29.2.1 cog

`std::vector<double> arcs::common_interface::VibrationRecalibrationParameter::cog`

Definition at line 216 of file [type_def.h](#).

11.29.2.2 inertia

`std::vector<double> arcs::common_interface::VibrationRecalibrationParameter::inertia`

Definition at line 217 of file [type_def.h](#).

11.29.2.3 mass

`double arcs::common_interface::VibrationRecalibrationParameter::mass`

Definition at line 215 of file [type_def.h](#).

11.29.2.4 points

`std::vector<std::vector<double> > arcs::common_interface::VibrationRecalibrationParameter::points`

Definition at line 218 of file [type_def.h](#).

11.29.2.5 stiff_param

`std::vector<std::vector<double> > arcs::common_interface::VibrationRecalibrationParameter::stiff_param`

Definition at line 220 of file [type_def.h](#).

11.29.2.6 stiff_section

`std::vector<std::vector<double> > arcs::common_interface::VibrationRecalibrationParameter::stiff_section`

Definition at line 219 of file [type_def.h](#).

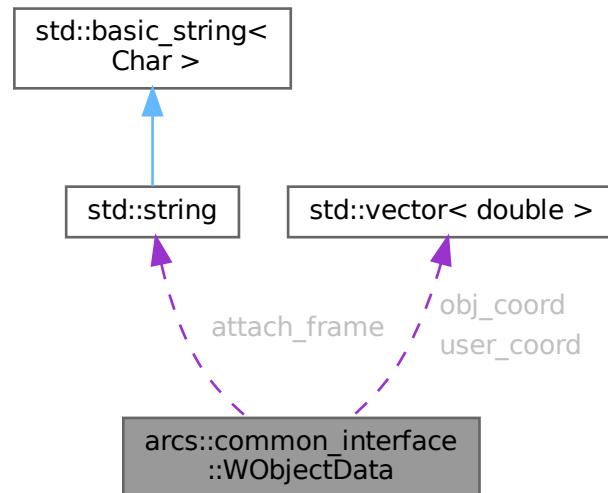
The documentation for this struct was generated from the following file:

- [include/aubo/type_def.h](#)

11.30 arcs::common_interface::WObjectData Struct Reference

```
#include <type_def.h>
```

Collaboration diagram for arcs::common_interface::WObjectData:



Public Attributes

- bool [remote_tool](#) { false }
Whether it is an external tool.
- std::string [attach_frame](#) { "" }
Coupled with the workpiece coordinate system.
- std::vector< double > [user_coord](#) { std::vector<double>(6, 0) }
User coordinate system.
- std::vector< double > [obj_coord](#) { std::vector<double>(6, 0) }
tool coordinate system, based on uframe

11.30.1 Detailed Description

Definition at line 187 of file [type_def.h](#).

11.30.2 Member Data Documentation

11.30.2.1 attach_frame

```
std::string arcs::common_interface::WObjectData::attach_frame { "" }
```

Coupled with the workpiece coordinate system.

Definition at line 194 of file [type_def.h](#).

11.30.2.2 obj_coord

```
std::vector<double> arcs::common_interface::WObjectData::obj_coord { std::vector<double>(6, 0) }
```

tool coordinate system, based on uframe

Definition at line 205 of file [type_def.h](#).

11.30.2.3 remote_tool

```
bool arcs::common_interface::WObjectData::remote_tool { false }
```

Whether it is an external tool.

Definition at line 190 of file [type_def.h](#).

11.30.2.4 user_coord

```
std::vector<double> arcs::common_interface::WObjectData::user_coord { std::vector<double>(6, 0) }
```

User coordinate system.

If robhold is false, the values of uframe are based on world. Otherwise, the values of uframe are based on flange.

Definition at line 201 of file [type_def.h](#).

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

- [include/aubo/type_def.h](#)

File Documentation

12.2 include/aubo/aubo_api.h File Reference

```
#include <aubo/system_info.h>
#include <aubo/runtime_machine.h>
#include <aubo/register_control.h>
#include <aubo/robot_interface.h>
#include <aubo/global_config.h>
#include <aubo/math.h>
#include <aubo/socket.h>
#include <aubo/serial.h>
#include <aubo/axis_interface.h>
#include <aubo/gripper_interface.h>
Include dependency graph for aubo_api.h:
```



- class arcs::common_interface::AuboApi

- namespace `arcs`
- namespace `arcs::common` interface

- using `arcs::common_interface::AuboApiPtr = std::shared_ptr<AuboApi>`

12.2.1 Detailed Description

API for controlling the robot and external axis.

Definition in file [aubo_api.h](#).

12.3 aubo_api.h

[Go to the documentation of this file.](#)

```

00001 /** @file aubo_api.h
00002  * @brief \~chinese      API
00003  * @brief \~english API for controlling the robot and external axis
00004  */
00005 #ifndef AUBO_SDK_AUBO_API_INTERFACE_H
00006 #define AUBO_SDK_AUBO_API_INTERFACE_H
00007
00008 #include <aubo/system_info.h>
00009 #include <aubo/runtime_machine.h>
00010 #include <aubo/register_control.h>
00011 #include <aubo/robot_interface.h>
00012 #include <aubo/global_config.h>
00013 #include <aubo/math.h>
00014 #include <aubo/socket.h>
00015 #include <aubo/serial.h>
00016 #include <aubo/axis_interface.h>
00017 #include <aubo/gripper_interface.h>
00018
00019 namespace arcs {
00020 namespace common_interface {
00021
00022 /**
00023  * @defgroup AuboApi AuboApi ( )
00024  * @ingroup AuboApi
00025  * AuboApi
00026  */
00027 class ARCS_ABI_EXPORT AuboApi
00028 {
00029 public:
00030     AuboApi();
00031     virtual ~AuboApi();
00032
00033     /**
00034      * @ingroup AuboApi
00035      * @ref Math
00036      * \~chinese
00037      *
00038      *
00039      * @return MathPtr
00040      *
00041      * @par Python
00042      * getMath(self: pyaubo_sdk.AuboApi) -> pyaubo_sdk.Math
00043      *
00044      * @par C++
00045      * @code
00046      * auto rpc_cli = std::make_shared<RpcClient>();
00047      * MathPtr ptr = rpc_cli->getMath();
00048      * @endcode
00049      * \~endchinese
00050      *
00051      * @english
00052      * Get pure mathematic related API
00053      *
00054      * @return Shared pointer to a Math object
00055      *
00056      * @par Python function prototype
00057      * getMath(self: pyaubo_sdk.AuboApi) -> pyaubo_sdk.Math
00058      *
00059      * @par C++ example
00060      * @code
00061      * auto rpc_cli = std::make_shared<RpcClient>();
00062      * MathPtr ptr = rpc_cli->getMath();
00063      * @endcode
00064      * \~endenglish
00065      */
00066     MathPtr getMath();
00067
00068     /**
00069      * @ingroup AuboApi

```

```

00070 * @ref SystemInfo
00071 * \chinese
00072 *
00073 *
00074 * @return SystemInfoPtr
00075 *
00076 * @par Python
00077 * getSystemInfo(self: pyaubo_sdk.AuboApi) -> pyaubo_sdk.SystemInfo
00078 *
00079 * @par C++
00080 * @code
00081 * auto rpc_cli = std::make_shared<RpcClient>();
00082 * SystemInfoPtr ptr = rpc_cli->getSystemInfo();
00083 * @endcode
00084 * \endchinese
00085 *
00086 * \english
00087 * Get system info
00088 *
00089 * @return Shared pointer to SystemInfo object
00090 *
00091 * @par Python function prototype
00092 * getSystemInfo(self: pyaubo_sdk.AuboApi) -> pyaubo_sdk.SystemInfo
00093 *
00094 * @par C++ example
00095 * @code
00096 * auto rpc_cli = std::make_shared<RpcClient>();
00097 * SystemInfoPtr ptr = rpc_cli->getSystemInfo();
00098 * @endcode
00099 * \endenglish
00100 */
00101 SystemInfoPtr getSystemInfo();
00102
00103 /**
00104 * @ingroup AuboApi
00105 * @ref RuntimeMachine
00106 * \chinese
00107 *
00108 *
00109 * @return RuntimeMachinePtr
00110 *
00111 * @par Python
00112 * getRuntimeMachine(self: pyaubo_sdk.AuboApi) -> pyaubo_sdk.RuntimeMachine
00113 *
00114 * @par C++
00115 * @code
00116 * auto rpc_cli = std::make_shared<RpcClient>();
00117 * RuntimeMachinePtr ptr = rpc_cli->getRuntimeMachine();
00118 * @endcode
00119 * \endchinese
00120 *
00121 * \english
00122 * Get runtime api
00123 *
00124 * @return Shared pointer to RuntimeMachine object
00125 * Python function prototype
00126 * getRuntimeMachine(self: pyaubo_sdk.AuboApi) -> pyaubo_sdk.RuntimeMachine
00127 *
00128 * @par C++ example
00129 * @code
00130 * auto rpc_cli = std::make_shared<RpcClient>();
00131 * RuntimeMachinePtr ptr = rpc_cli->getRuntimeMachine();
00132 * @endcode
00133 * \endenglish
00134 */
00135 RuntimeMachinePtr getRuntimeMachine();
00136
00137 /**
00138 * @ingroup AuboApi
00139 * @ref RegisterControl
00140 * \chinese
00141 *
00142 *
00143 * @return RegisterControlPtr
00144 *
00145 * @par Python
00146 * getRegisterControl(self: pyaubo_sdk.AuboApi) ->
00147 * pyaubo_sdk.RegisterControl
00148 *
00149 * @par C++
00150 * @code
00151 * auto rpc_cli = std::make_shared<RpcClient>();
00152 * RegisterControlPtr ptr = rpc_cli->getRegisterControl();
00153 * @endcode
00154 * \endchinese
00155 *
00156 * \english

```

```

00157 * External registers api
00158 *
00159 * @return Shared pointer to RegisterControl object
00160 *
00161 * @par Python function prototype
00162 * getRegisterControl(self: pyaubo_sdk.AuboApi) ->
00163 * pyaubo_sdk.RegisterControl
00164 *
00165 * @par C++ example
00166 * @code
00167 * auto rpc_cli = std::make_shared<RpcClient>();
00168 * RegisterControlPtr ptr = rpc_cli->getRegisterControl();
00169 * @endcode
00170 * \endenglish
00171 */
00172 RegisterControlPtr getRegisterControl();
00173
00174 /**
00175 * @ingroup AuboApi
00176 * \chinese
00177 *
00178 *
00179 * @return
00180 *
00181 * @par Python
00182 * getRobotNames(self: pyaubo_sdk.AuboApi) -> List[str]
00183 *
00184 * @par C++
00185 * @code
00186 * auto rpc_cli = std::make_shared<RpcClient>();
00187 * auto robot_name = rpc_cli->getRobotNames().front();
00188 * @endcode
00189 *
00190 * @par JSON-RPC
00191 * {"jsonrpc":"2.0","method":"getRobotNames","params":[],"id":1}
00192 *
00193 * @par JSON-RPC
00194 * {"id":1,"jsonrpc":"2.0","result":["rob1"]}
00195 * \endchinese
00196 *
00197 * \english
00198 * Get robot list
00199 *
00200 * @return robot list
00201 *
00202 * @par Python function prototype
00203 * getRobotNames(self: pyaubo_sdk.AuboApi) -> List[str]
00204 *
00205 * @par C++ example
00206 * @code
00207 * auto rpc_cli = std::make_shared<RpcClient>();
00208 * auto robot_name = rpc_cli->getRobotNames().front();
00209 * @endcode
00210 *
00211 * @par JSON-RPC request example
00212 * {"jsonrpc":"2.0","method":"getRobotNames","params":[],"id":1}
00213 *
00214 * @par JSON-RPC response example
00215 * {"id":1,"jsonrpc":"2.0","result":["rob1"]}
00216 * \endenglish
00217 */
00218 std::vector<std::string> getRobotNames();
00219
00220 /**
00221 * @ingroup AuboApi
00222 * @ref RobotInterface
00223 * \chinese
00224 * RobotInterfacePtr
00225 *
00226 * @param name
00227 * @return RobotInterfacePtr
00228 *
00229 * @par Python
00230 * getRobotInterface(self: pyaubo_sdk.AuboApi, arg0: str) ->
00231 * pyaubo_sdk.RobotInterface
00232 *
00233 * @par C++
00234 * @code
00235 * auto rpc_cli = std::make_shared<RpcClient>();
00236 * auto robot_name = rpc_cli->getRobotNames().front();
00237 * RobotInterfacePtr ptr = rpc_cli->getRobotInterface(robot_name);
00238 * @endcode
00239 * \endchinese
00240 *
00241 * \english
00242 * Get RobotInterfacePtr based on name
00243 *

```

```

00244 * @param name Robot name
00245 * @return Shared pointer to a RobotInterface object
00246 *
00247 * @par Python function prototype
00248 * getRobotInterface(self: pyaubo_sdk.AuboApi, arg0: str) ->
00249 * pyaubo_sdk.RobotInterface
00250 *
00251 * @par C++ example
00252 * @code
00253 * auto rpc_cli = std::make_shared<RpcClient>();
00254 * auto robot_name = rpc_cli->getRobotNames().front();
00255 * RobotInterfacePtr ptr = rpc_cli->getRobotInterface(robot_name);
00256 * @endcode
00257 * \endenglish
00258 */
00259 RobotInterfacePtr getRobotInterface(const std::string &name);
00260
00261 /**
00262 * @ingroup AuboApi
00263 * \~chinese \~english Get external axis list
00264 *
00265 * @return
00266 */
00267 std::vector<std::string> getAxisNames();
00268
00269 /**
00270 * @ingroup AuboApi
00271 * @ref AxisInterface
00272 * \~chinese
00273 *
00274 *
00275 * @param name
00276 * @return
00277 * \endchinese
00278 *
00279 * \english
00280 * Get external axis interface
00281 *
00282 * @param name
00283 * @return
00284 * \endenglish
00285 */
00286 AxisInterfacePtr getAxisInterface(const std::string &name);
00287
00288 /**
00289 * @ingroup AuboApi
00290 * @ref Socket
00291 * \~chinese
00292 * socket
00293 * @return SocketPtr
00294 *
00295 * @par Python
00296 * getSocket(self: pyaubo_sdk.AuboApi) -> arcs::common_interface::Socket
00297 * @endcode
00298 *
00299 * @par C++
00300 * @code
00301 * auto rpc_cli = std::make_shared<RpcClient>();
00302 * SocketPtr ptr = rpc_cli->getSocket();
00303 * @endcode
00304 * \endchinese
00305 *
00306 * \english
00307 * Get socket
00308 * @return Shared pointer to a socket object
00309 *
00310 * @par Python function prototype
00311 * getSocket(self: pyaubo_sdk.AuboApi) -> arcs::common_interface::Socket
00312 * @endcode
00313 *
00314 * @par C++ example
00315 * @code
00316 * auto rpc_cli = std::make_shared<RpcClient>();
00317 * SocketPtr ptr = rpc_cli->getSocket();
00318 * @endcode
00319 * \endenglish
00320 */
00321 SocketPtr getSocket();
00322
00323 /**
00324 * @ingroup AuboApi
00325 * @ref Serial
00326 * \~chinese
00327 * Serial
00328 * @return SerialPtr
00329 *
00330 * @par Python

```

```

00331     * getSerial(self: pyaubo_sdk.AuboApi) -> arcs::common_interface::Serial
00332     *
00333     * @par C++
00334     * @code
00335     * auto rpc_cli = std::make_shared<RpcClient>();
00336     * SerialPtr ptr = rpc_cli->getSerial();
00337     * @endcode
00338     * \endchinese
00339     *
00340     * \english
00341     * @return Shared pointer to Serial object
00342     *
00343     * @par Python function prototype
00344     * getSerial(self: pyaubo_sdk.AuboApi) -> arcs::common_interface::Serial
00345     *
00346     * @par C++ example
00347     * @code
00348     * auto rpc_cli = std::make_shared<RpcClient>();
00349     * SerialPtr ptr = rpc_cli->getSerial();
00350     * @endcode
00351     * \endenglish
00352     */
00353     SerialPtr getSerial();
00354
00355     /**
00356     * @ingroup AuboApi
00357     * @ref SyncMove
00358     * \-chinese \-english Get synchronous move interface
00359     * \-chinese @return SyncMovePtr
00360     * \-english @return Shared pointer to SyncMove object
00361     */
00362     SyncMovePtr getSyncMove(const std::string &name);
00363
00364     /**
00365     * @ingroup AuboApi
00366     * @ref Trace
00367     * \-chinese
00368     * \-english Get alert interface
00369     *
00370     * \-chinese @return TracePtr
00371     * \-english @return Shared pointer of trace object
00372     */
00373     TracePtr getTrace(const std::string &name);
00374
00375     /**
00376     * @ingroup AuboApi
00377     * @ref GripperInterface
00378     * \-chinese
00379     * \-english Get gripper interface
00380     *
00381     * \-chinese @return GripperInterfacePtr
00382     * \-english @return Shared pointer of gripper object
00383     */
00384     GripperInterfacePtr getGripperInterface();
00385
00386     protected:
00387     void *d_{ nullptr };
00388 };
00389 using AuboApiPtr = std::shared_ptr<AuboApi>;
00390
00391 // namespace common_interface
00392 // namespace arcs
00393 #endif // AUBO_SDK_AUBO_API_H

```

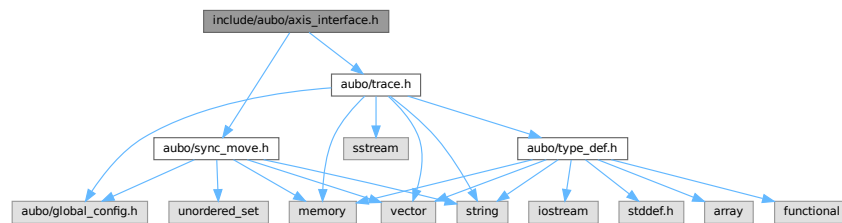
12.4 include/aubo/axis_interface.h File Reference

```

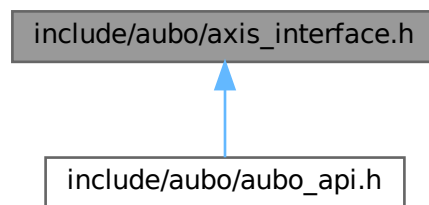
#include <aubo/sync_move.h>
#include <aubo/trace.h>

```

Include dependency graph for axis_interface.h:



This graph shows which files directly or indirectly include this file:



Classes

- class [arcs::common_interface::AxisInterface](#)

Namespaces

- namespace [arcs](#)
- namespace [arcs::common_interface](#)

Typedefs

- using [arcs::common_interface::AxisInterfacePtr](#) = `std::shared_ptr<AxisInterface>`

12.5 axis_interface.h

[Go to the documentation of this file.](#)

```

00001 /** @file axes.h
00002  * @brief
00003  */
00004 #ifndef AUBO_SDK_AXIS_INTERFACE_H
00005 #define AUBO_SDK_AXIS_INTERFACE_H
00006
00007 #include <aubo/sync_move.h>

```

```

00008 #include <aubo/trace.h>
00009
00010 namespace arcs {
00011 namespace common_interface {
00012
00013 /**
00014 * @defgroup AxisInterface AxisInterface ( )
00015 * \-chinese API
00016 * \-english External axis API interface
00017 */
00018 class ARCS_ABI_EXPORT AxisInterface
00019 {
00020 public:
00021     AxisInterface();
00022     virtual ~AxisInterface();
00023
00024 /**
00025 * @ingroup AxisInterface
00026 * \-chinese
00027 * \-english Power on
00028 * @return
00029 */
00030     int poweronExtAxis();
00031
00032 /**
00033 * @ingroup AxisInterface
00034 * \-chinese
00035 * \-english Power off
00036 * @return
00037 */
00038     int poweroffExtAxis();
00039
00040 /**
00041 * @ingroup AxisInterface
00042 * \-chinese
00043 * \-english Enable
00044 * @return
00045 */
00046     int enableExtAxis();
00047
00048 /**
00049 * @ingroup AxisInterface
00050 * \-chinese ( )
00051 * \-chinese @param pose
00052 * \-english Set mounting pose of external axis (wrt world frame)
00053 * \-english @param pose
00054
00055 * @return
00056 */
00057     int setExtAxisMountingPose(const std::vector<double> &pose);
00058
00059 /**
00060 * @ingroup AxisInterface
00061 * \-chinese
00062 *
00063 * @param pos
00064 * @param v
00065 * @param a
00066 * @param duration
00067 * @return
00068 * \endchinese
00069 *
00070 * \english move to pos, rotation or linear
00071 *
00072 * @param pos
00073 * @param v
00074 * @param a
00075 * @param duration
00076 * @return
00077 * \endenglish
00078 */
00079     int moveExtJoint(double pos, double v, double a, double duration);
00080
00081 /**
00082 * @ingroup AxisInterface
00083 * \-chinese
00084 *
00085 *
00086 * @param v
00087 * @param a
00088 * @param duration
00089 * @return
00090 * \endchinese
00091 *
00092 * \english
00093 * Set target speed, acceleration and duration
00094 * @param v

```



```

00095     * @param a
00096     * @param duration
00097     * @return
00098     * \endenglish
00099     */
00100 int speedExtJoint(double v, double a, double duration);
00101
00102 /**
00103  * @ingroup AxisInterface
00104  * \chinese
00105  *
00106  *
00107  * @param a
00108  * @return
00109  * \endchinese
00110  *
00111  * \english
00112  * stop ext joint
00113  * @param a
00114  * @return
00115  * \endenglish
00116  */
00117 int stopExtJoint(double a);
00118
00119 /**
00120  * @ingroup AxisInterface
00121  * \chinese 0 1
00122  * \english Get external axis type: 0 for rotation, 1 for linear
00123  * @return
00124  */
00125 int getExtAxisType();
00126
00127 /**
00128  * @ingroup AxisInterface
00129  * \chinese
00130  *
00131  *
00132  * @return
00133  * \endchinese
00134  *
00135  * \english
00136  * Get external axis status
00137  *
00138  * @return Current external axis status
00139  * \endenglish
00140  */
00141 AxisModeType getAxisModeType();
00142
00143 /**
00144  * @ingroup AxisInterface
00145  * \chinese
00146  *
00147  *
00148  * @return
00149  * \endchinese
00150  *
00151  * \english
00152  * Get external axis mounting pose
00153  *
00154  * @return External axis pose
00155  * \endenglish
00156  */
00157 std::vector<double> getExtAxisMountingPose();
00158
00159 /**
00160  * @ingroup AxisInterface
00161  * \chinese
00162  *
00163  *
00164  * @return
00165  * \endchinese
00166  *
00167  * \english
00168  * Get pose wrt mounting coordinate system, axis can be positioner or linear rail
00169  *
00170  * @return Pose wrt mounting coordinate system
00171  * \endenglish
00172  */
00173 std::vector<double> getExtAxisPose();
00174
00175 /**
00176  * @ingroup AxisInterface
00177  * \chinese
00178  *
00179  *
00180  * @return
00181  * \endchinese

```

```

00182      *
00183      * \english
00184      * Get external axis position
00185      *
00186      * @return External axis position
00187      * \endenglish
00188      */
00189      double getExtAxisPosition();
00190
00191      /**
00192      * @ingroup AxisInterface
00193      * \chinese
00194      *
00195      *
00196      * @return
00197      * \endchinese
00198      *
00199      * \english
00200      * Get external axis speed
00201      *
00202      * @return External axis speed
00203      * \endenglish
00204      */
00205      double getExtAxisVelocity();
00206
00207      /**
00208      * @ingroup AxisInterface
00209      * \chinese
00210      *
00211      *
00212      * @return
00213      * \endchinese
00214      *
00215      * \english
00216      * Get external axis acceleration
00217      *
00218      * @return External axis acceleration
00219      * \endenglish
00220      */
00221      double getExtAxisAcceleration();
00222
00223      /**
00224      * @ingroup AxisInterface
00225      * \chinese
00226      *
00227      *
00228      * @return
00229      * \endchinese
00230      *
00231      * \english
00232      * Get external axis current
00233      *
00234      * @return External axis current
00235      * \endenglish
00236      */
00237      double getExtAxisCurrent();
00238
00239      /**
00240      * @ingroup AxisInterface
00241      * \chinese
00242      *
00243      *
00244      * @return
00245      * \endchinese
00246      *
00247      * \english
00248      * Get external axis temperature
00249      *
00250      * @return External axis temperature
00251      * \endenglish
00252      */
00253      double getExtAxisTemperature();
00254
00255      /**
00256      * @ingroup AxisInterface
00257      * \chinese
00258      *
00259      *
00260      * @return
00261      * \endchinese
00262      *
00263      * \english
00264      * Get external axis voltage
00265      *
00266      * @return External axis voltage
00267      * \endenglish
00268      */

```

```

00269 double getExtAxisBusVoltage();
00270
00271 /**
00272  * @ingroup AxisInterface
00273  * \chinese
00274  *
00275  *
00276  * @return
00277  * \endchinese
00278  *
00279  * \english
00280  * Get external axis current
00281  *
00282  * @return external axis current
00283  * \endenglish
00284  */
00285 double getExtAxisBusCurrent();
00286
00287 /**
00288  * @ingroup AxisInterface
00289  * \chinese
00290  *
00291  *
00292  * @return
00293  * \endchinese
00294  *
00295  * \english
00296  * Get external axis max position
00297  *
00298  * @return External axis max position
00299  * \endenglish
00300  */
00301 double getExtAxisMaxPosition();
00302
00303 /**
00304  * @ingroup AxisInterface
00305  * \chinese
00306  *
00307  *
00308  * @return
00309  * \endchinese
00310  *
00311  * \english
00312  * Get external axis min position
00313  *
00314  * @return External axis min position
00315  * \endenglish
00316  */
00317 double getExtMinPosition();
00318
00319 /**
00320  * @ingroup AxisInterface
00321  * \chinese
00322  *
00323  *
00324  * @return
00325  * \endchinese
00326  *
00327  * \english
00328  * Get external axis max speed
00329  *
00330  * @return External axis max speed
00331  * \endenglish
00332  */
00333 double getExtAxisMaxVelocity();
00334
00335 /**
00336  * @ingroup AxisInterface
00337  * \chinese
00338  *
00339  *
00340  * @return
00341  * \endchinese
00342  *
00343  * \english
00344  * Get external axis max acceleration
00345  *
00346  * @return External axis max acceleration
00347  * \endenglish
00348  */
00349 double getExtAxisMaxAcceleration();
00350
00351 /**
00352  * @ingroup AxisInterface
00353  * \chinese
00354  * ( )
00355  *

```

```

00356     * @param target_name
00357     * @param phase
00358     * @param err
00359     * @return
00360     * \endchinese
00361     *
00362     * \english
00363     * Follow motion of another external axis not to be used during motion
00364     *
00365     * @param target_name name of target axis
00366     * @param phase phase difference
00367     * @param err max error when following motion
00368     * @return
00369     * \endenglish
00370     */
00371     int followAnotherAxis(const std::string &target_name, double phase,
00372                          double err);
00373
00374     /**
00375     * @ingroup AxisInterface
00376     * \~chinese @brief stopFollowAnotherAxis( )
00377     * \~english @brief stopFollowAnotherAxis(not to be used during motion)
00378     * @return
00379     */
00380     int stopFollowAnotherAxis();
00381
00382     /**
00383     * @ingroup AxisInterface
00384     * \chinese
00385     *
00386     *
00387     * @return
00388     * \endchinese
00389     *
00390     * \english
00391     * Get external axis error code
00392     *
00393     * @return External axis error code
00394     * \endenglish
00395     */
00396     int getErrorCode();
00397
00398     /**
00399     * @ingroup AxisInterface
00400     * \chinese
00401     *
00402     *
00403     * @return
00404     * \endchinese
00405     *
00406     * \english
00407     * Reset axis error
00408     *
00409     * @return
00410     * \endenglish
00411     */
00412     int clearAxisError();
00413
00414 protected:
00415     void *d_;
00416 };
00417 using AxisInterfacePtr = std::shared_ptr<AxisInterface>;
00418
00419 } // namespace common_interface
00420 } // namespace arcs
00421
00422 #endif // AUBO_SDK_AXIS_INTERFACE_H

```

12.6 include/aubo/error_stack/error_stack.h File Reference

```

#include <stdio.h>
#include <stdint.h>
#include <string.h>
#include <string>
#include <sstream>
#include <iomanip>

```

```
#include <aubo/global_config.h>
#include <aubo/error_stack/hal_error.h>
#include <aubo/error_stack/rtm_error.h>
#include <aubo/error_stack/system_error.h>
Include dependency graph for error_stack.h:
```



Namespaces

- namespace [arcs](#)
- namespace [arcs::error_stack](#)

Macros

- `#define` [_PH1](#) `"{}"`
- `#define` [_PH2](#) `"{}"`
- `#define` [_PH3](#) `"{}"`
- `#define` [_PH4](#) `"{}"`
- `#define` [AUBO_SDK_HAL_ERROR_H](#)
- `#define` [JOINT_ERRORS](#)
- `#define` [TOOL_ERRORS](#)
- `#define` [PEDSTRAL_ERRORS](#)
- `#define` [SAFETY_INTERFACE_BOARD_ERRORS](#)
- `#define` [HARDWARE_INTERFACE_ERRORS](#)
- `#define` [HAL_ERRORS](#)
- `#define` [AUBO_SDK_RTM_ERROR_H](#)
- `#define` [RTM_ERRORS](#)
- `#define` [AUBO_SDK_SYSTEM_ERROR_H](#)
- `#define` [SYSTEM_ERRORS](#)
- `#define` [ARCS_ERROR_CODES](#)
- `#define` [_D](#)(n, v, s, r)
- `#define` [_D](#)(n, v, s, r)
- `#define` [_D](#)(n, v, s, r)
- `#define` [_D](#)(n, v, s, r)
- `#define` [_D](#)(n, v, s, r)
- `#define` [_D](#)(n, v, s, r)

Enumerations

- enum [arcs::error_stack::ErrorCodes](#) { [arcs::error_stack::ARCS_ERROR_CODES](#) }

Functions

- constexpr int [ARCS_ABI_EXPORT arcs::error_stack::codeCompose](#) (int aa, int bb, int cccc)
- constexpr int [ARCS_ABI_EXPORT arcs::error_stack::mod](#) (int x)
- int [arcs::error_stack::str2ErrorCode](#) (const char *err_code_name)
- const char * [arcs::error_stack::errorCode2Str](#) (int err_code)
- std::ostream & [arcs::error_stack::dump](#) (std::ostream &os)

12.6.1 Detailed Description

Definition in file [error_stack.h](#).

12.6.2 Macro Definition Documentation

12.6.2.1 __D [1/6]

```
#define __D(
    n,
    v,
    s,
    r)
```

Value:

```
os « std::setw(20) « #n « "\t" « v « "\t" « s « "\t" « r \
« std::endl;
```

Definition at line 54 of file [error_stack.h](#).

12.6.2.2 __D [2/6]

```
#define __D(
    n,
    v,
    s,
    r)
```

Value:

```
if (err_code == v) \
    index = n##_INDEX;
```

Definition at line 54 of file [error_stack.h](#).

12.6.2.3 __D [3/6]

```
#define __D(
    n,
    v,
    s,
    r)
```

Value:

```
n##_INDEX,
```

Definition at line 54 of file [error_stack.h](#).

12.6.2.4 __D [4/6]

```
#define __D(  
    n,  
    v,  
    s,  
    r)
```

Value:

s,

Definition at line 54 of file [error_stack.h](#).

12.6.2.5 __D [5/6]

```
#define __D(  
    n,  
    v,  
    s,  
    r)
```

Value:

```
if (strcmp(#n, err_code_name) == 0) \  
    return v;
```

Definition at line 54 of file [error_stack.h](#).

12.6.2.6 __D [6/6]

```
#define __D(  
    n,  
    v,  
    s,  
    r)
```

Value:

n = (int)v,

Definition at line 54 of file [error_stack.h](#).

12.6.2.7 __PH1__

```
#define __PH1__ "{}"
```

Definition at line 18 of file [error_stack.h](#).

12.6.2.8 __PH2__

```
#define __PH2__ "{}"
```

Definition at line 19 of file [error_stack.h](#).

12.6.2.9 __PH3__

```
#define __PH3__ "{}"
```

Definition at line 20 of file [error_stack.h](#).

12.6.2.10 __PH4__

```
#define __PH4__ "{}"
```

Definition at line 21 of file [error_stack.h](#).

12.6.2.11 ARCS_ERROR_CODES

```
#define ARCS_ERROR_CODES
```

Value:

```
SYSTEM_ERRORS      \
JOINT_ERRORS        \
SAFETY_INTERFACE_BOARD_ERRORS \
RTM_ERRORS          \
TOOL_ERRORS         \
PEDSTRAL_ERRORS     \
HARDWARE_INTERFACE_ERRORS \
_D(ARCS_MAX_ERROR_CODE, -1, "Max error code", "suggest...")
```

Definition at line 41 of file [error_stack.h](#).

12.6.2.12 AUBO_SDK_HAL_ERROR_H

```
#define AUBO_SDK_HAL_ERROR_H
```

Definition at line 6 of file [error_stack.h](#).

12.6.2.13 AUBO_SDK_RTM_ERROR_H

```
#define AUBO_SDK_RTM_ERROR_H
```

Definition at line 6 of file [error_stack.h](#).

12.6.2.14 AUBO_SDK_SYSTEM_ERROR_H

```
#define AUBO_SDK_SYSTEM_ERROR_H
```

Definition at line 6 of file [error_stack.h](#).

12.6.2.15 HAL_ERRORS

```
#define HAL_ERRORS
```

Value:

```
JOINT_ERRORS          \
SAFETY_INTERFACE_BOARD_ERRORS \
TOOL_ERRORS           \
PEDSTRAL_ERRORS       \
HARDWARE_INTERFACE_ERRORS
```

12.6.2.16 HARDWARE_INTERFACE_ERRORS

```
#define HARDWARE_INTERFACE_ERRORS
```

12.6.2.17 JOINT_ERRORS

```
#define JOINT_ERRORS
```

12.6.2.18 PEDSTRAL_ERRORS

```
#define PEDSTRAL_ERRORS
```

Value:

```
_D(PKG_LOST, 50001, "Lost package from pedestal", "suggest...") \
_D(PEDSTRAL_OFFLINE, 50002, "pedestal error: pedestal may be offline", "(a) Check pedestal's hardware. (b) Check pedestal's id.") \
_D(PEDSTRAL_ERR_BOOTLOADER, 50003, "pedestal error: The pedestal is in bootloader mode. Retry firmware update.", "suggest...")
```

12.6.2.19 RTM_ERRORS

```
#define RTM_ERRORS
```

12.6.2.20 SAFETY_INTERFACE_BOARD_ERRORS

```
#define SAFETY_INTERFACE_BOARD_ERRORS
```

Value:

```
_D(IFB_ERR_ROBOTTYPE, 20001, "Robot error type!", "suggest...") \
_D(IFB_ERR_ADXL_SENS, 20002, "Base Acceleration sensor error!", "suggest...") \
_D(IFB_ERR_EN_LINE, 20003, "Encoder line error!", "suggest...") \
_D(IFB_ERR_ENTER_HDG_MODE, 20004, "Robot enter handguide mode!", "suggest...") \
_D(IFB_ERR_EXIT_HDG_MODE, 20005, "Robot exit handguide mode!", "suggest...") \
_D(IFB_ERR_MAC_DATA_BREAK, 20006, "MAC data break!", "suggest...") \
_D(IFB_ERR_DRV_FIRMWARE_VERSION, 20007, "Motor driver firmware version error!", "suggest...") \
_D(INIT_ERR_EN_DRV, 20008, "Motor driver enable failed!", "suggest...") \
_D(INIT_ERR_EN_AUTO_BACK, 20009, "Motor driver enable auto back failed!", "suggest...") \
_D(INIT_ERR_EN_CUR_LOOP, 20010, "Motor driver enable current loop failed!", "suggest...") \
_D(INIT_ERR_SET_TAG_CUR, 20011, "Motor driver set target current failed!", "suggest...") \
_D(INIT_ERR_RELEASE_BRAKE, 20012, "Motor driver release brake failed!", "suggest...") \
_D(INIT_ERR_EN_POS_LOOP, 20013, "Motor driver enable position loop failed!", "suggest...") \
_D(INIT_ERR_SET_MAX_ACC, 20014, "Motor set max accelerate failed!", "suggest...") \
_D(SAFETY_ERR_PROTECTION_STOP_TIMEOUT, 20015, "Protective stop timeout!", "suggest...") \
_D(SAFETY_ERR_REDUCED_MODE_TIMEOUT, 20016, "Reduced mode timeout!", "suggest...") \
_D(SYS_ERR_MCU_COM, 20017, "Robot system error: mcu communication error!", "suggest...") \
_D(SYS_ERR_RS485_COM, 20018, "Robot system error: RS485 communication error!", "suggest...") \
_D(IFB_ERR_DISCONNECTED, 20019, "Interface board may be disconnected. Please check connection between IPC and Interface board.", "suggest...") \
_D(IFB_ERR_PAYLOAD_ERROR, 20020, "Payload error.", "suggest...") \
_D(IFB_OFFLINE, 20021, "ifaceboard error: ifaceboard may be offline", "(a) Check ifaceboard's hardware. (b) Check ifaceboard's id.") \
_D(IFB_ERR_BOOTLOADER, 20022, "ifaceboard error: The ifaceboard is in bootloader mode. Retry firmware update.", "suggest...") \
_D(IFB_SLAVE_OFFLINE, 20023, "interface slave board error: interface slave board may be offline", "(a) Check interface slave board's hardware. (b) Check interface slave board's id.") \
_D(IFB_SLAVE_ERR_BOOTLOADER, 20024, "interface slave board error: The interface slave board is in bootloader mode. Retry firmware update.", "suggest...") \
_D(IFB_TOOL_ERR_ADXL_SENS, 20025, "Tool Acceleration sensor error!", "suggest...")
```

12.6.2.21 SYSTEM_ERRORS

```
#define SYSTEM_ERRORS
```

12.6.2.22 TOOL_ERRORS

```
#define TOOL_ERRORS
```

Value:

```
_D(TOOL_FLASH_VERIFY_FAILED, 40001, "Flash write verify failed", "suggest...") \
_D(TOOL_PROGRAM_CRC_FAILED, 40002, "Program flash checksum failed during bootloading", "suggest...") \
_D(TOOL_PROGRAM_CRC_FAILED2, 40003, "Program flash checksum failed at runtime", "suggest...") \
_D(TOOL_ID_UNDEFINED, 40004, "Tool ID is undefined", "suggest...") \
_D(TOOL_ILLEGAL_BL_CMD, 40005, "Illegal bootloader command", "suggest...") \
_D(TOOL_FW_WRONG, 40006, "Wrong firmware at the joint", "suggest...") \
_D(TOOL_HW_INVALID, 40007, "Invalid hardware revision", "suggest...") \
_D(TOOL_SHORT_CURCUIT_H, 40011, "Short circuit detected on Digital Output: " _PH1_ " high side", "suggest...") \
_D(TOOL_SHORT_CURCUIT_L, 40012, "Short circuit detected on Digital Output: " _PH1_ " low side", "suggest...") \
_D(TOOL_AVERAGE_CURR_HIGH, 40013, "10 second Average tool IO Current of " _PH1_ " A is outside of the allowed range.", "suggest...") \
_D(TOOL_POWER_PIN_OVER_CURR, 40014, "Current of " _PH1_ " A on the POWER pin is outside of the allowed range.", "suggest...") \
_D(TOOL_DOUT_PIN_OVER_CURR, 40015, "Current of " _PH1_ " A on the Digital Output pins is outside of the allowed range.", "suggest...") \
_D(TOOL_GROUND_PIN_OVER_CURR, 40016, "Current of " _PH1_ " A on the ground pin is outside of the allowed range.", "suggest...") \
_D(TOOL_RX_FRAMING, 40021, "RX framing error", "suggest...") \
_D(TOOL_RX_PARITY, 40022, "RX Parity error", "suggest...") \
_D(TOOL_48V_LOW, 40031, "48V input is too low", "suggest...") \
_D(TOOL_48V_HIGH, 40032, "48V input is too high", "suggest...") \
_D(TOOL_ERR_OFFLINE, 40033, "tool error: tool may be offline", "(a) Check tool's hardware. (b) Check joint's id.") \
_D(TOOL_ERR_BOOTLOADER, 40034, "tool error: The tool is in bootloader mode. Retry firmware update. ", "suggest...")
```

12.7 error_stack.h

[Go to the documentation of this file.](#)

```
00001 /** @file error_stack.h
00002  * @brief
00003  */
00004 #ifndef AUBO_SDK_ERROR_STACK_H
00005 #define AUBO_SDK_ERROR_STACK_H
00006
00007 #include <stdio.h>
00008 #include <stdint.h>
00009 #include <string.h>
00010 #include <string>
00011 #include <sstream>
00012 #include <iomanip>
00013
00014 #include <aubo/global_config.h>
00015
00016 //      fmt
00017 #ifndef _PH1_
00018 #define _PH1_ "{}"
00019 #define _PH2_ "{}"
00020 #define _PH3_ "{}"
00021 #define _PH4_ "{}"
00022 #endif
00023
00024 namespace arcs {
00025 namespace error_stack {
00026
00027 constexpr int ARCS_ABI_EXPORT codeCompose(int aa, int bb, int cccc)
00028 {
00029     return (int)((aa * 1000000) + (bb * 10000) + cccc);
00030 }
00031
00032 constexpr int ARCS_ABI_EXPORT mod(int x)
00033 {
00034     return (x % 1000000);
00035 }
00036
00037 #include <aubo/error_stack/hal_error.h>
```

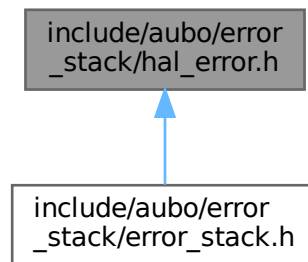
```

00038 #include <aubo/error_stack/rtm_error.h>
00039 #include <aubo/error_stack/system_error.h>
00040
00041 #define ARCS_ERROR_CODES \
00042     SYSTEM_ERRORS \
00043     JOINT_ERRORS \
00044     SAFETY_INTERFACE_BOARD_ERRORS \
00045     RTM_ERRORS \
00046     TOOL_ERRORS \
00047     PEDSTRAL_ERRORS \
00048     HARDWARE_INTERFACE_ERRORS \
00049     _D(ARCS_MAX_ERROR_CODE, -1, "Max error code", "suggest...")
00050
00051 //
00052 enum ErrorCodes
00053 {
00054     #define _D(n, v, s, r) n = (int)v,
00055     ARCS_ERROR_CODES
00056 #undef _D
00057 };
00058
00059 inline int str2ErrorCode(const char *err_code_name)
00060 {
00061     #define _D(n, v, s, r) \
00062         if (strcmp(#n, err_code_name) == 0) \
00063             return v;
00064     ARCS_ERROR_CODES
00065 #undef _D
00066     return ARCS_MAX_ERROR_CODE;
00067 }
00068
00069 inline const char *errorCode2Str(int err_code)
00070 {
00071     static const char *errcode_str[] = {
00072         #define _D(n, v, s, r) s,
00073         ARCS_ERROR_CODES
00074     #undef _D
00075     };
00076
00077     enum arcs_index
00078     {
00079         #define _D(n, v, s, r) n##_INDEX,
00080         ARCS_ERROR_CODES
00081     #undef _D
00082     };
00083
00084     int index = -1;
00085
00086     #define _D(n, v, s, r) \
00087         if (err_code == v) \
00088             index = n##_INDEX;
00089     ARCS_ERROR_CODES
00090 #undef _D
00091
00092     if (index == -1) {
00093         index = ARCS_MAX_ERROR_CODE_INDEX;
00094     }
00095
00096     return errcode_str[(unsigned)index];
00097 }
00098
00099 inline std::ostream &dump(std::ostream &os)
00100 {
00101     #define _D(n, v, s, r) \
00102         os « std::setw(20) « #n « "t" « v « "t" « s « "t" « r \
00103         « std::endl;
00104     ARCS_ERROR_CODES
00105 #undef _D
00106
00107     return os;
00108 }
00109 }
00110
00111 } // namespace error_stack
00112 } // namespace arcs
00113
00114 #endif // AUBO_SDK_ERROR_STACK_H

```

12.8 include/aubo/error_stack/hal_error.h File Reference

This graph shows which files directly or indirectly include this file:



Macros

- `#define` [JOINT_ERRORS](#)
- `#define` [TOOL_ERRORS](#)
- `#define` [PEDSTRAL_ERRORS](#)
- `#define` [SAFETY_INTERFACE_BOARD_ERRORS](#)
- `#define` [HARDWARE_INTERFACE_ERRORS](#)
- `#define` [HAL_ERRORS](#)

12.8.1 Detailed Description

Definition in file [hal_error.h](#).

12.8.2 Macro Definition Documentation

12.8.2.1 HAL_ERRORS

```
#define HAL_ERRORS
```

Value:

```
JOINT_ERRORS \
SAFETY_INTERFACE_BOARD_ERRORS \
TOOL_ERRORS \
PEDSTRAL_ERRORS \
HARDWARE_INTERFACE_ERRORS
```

Definition at line 160 of file [error_stack.h](#).

12.8.2.2 HARDWARE_INTERFACE_ERRORS

```
#define HARDWARE_INTERFACE_ERRORS
```

Definition at line 122 of file [error_stack.h](#).

12.8.2.3 JOINT_ERRORS

```
#define JOINT_ERRORS
```

Definition at line 20 of file [error_stack.h](#).

12.8.2.4 PEDSTRAL_ERRORS

```
#define PEDSTRAL_ERRORS
```

Value:

```
_D(PKG_LOST, 50001, "Lost package from pedestal", "suggest...") \
_D(PEDSTRAL_OFFLINE, 50002, "pedestal error: pedestal may be offline", "(a) Check pedestal's hardware. (b) Check pedestal's id.") \
_D(PEDSTRAL_ERR_BOOTLOADER, 50003, "pedestal error: The pedestal is in bootloader mode. Retry firmware update. ", "suggest...")
```

Definition at line 88 of file [error_stack.h](#).

12.8.2.5 SAFETY_INTERFACE_BOARD_ERRORS

```
#define SAFETY_INTERFACE_BOARD_ERRORS
```

Value:

```
_D(IFB_ERR_ROBOTTYPE, 20001, "Robot error type!", "suggest...") \
_D(IFB_ERR_ADXL_SENS, 20002, "Base Acceleration sensor error!", "suggest...") \
_D(IFB_ERR_EN_LINE, 20003, "Encoder line error!", "suggest...") \
_D(IFB_ERR_ENTER_HDG_MODE, 20004, "Robot enter handguide mode!", "suggest...") \
_D(IFB_ERR_EXIT_HDG_MODE, 20005, "Robot exit handguide mode!", "suggest...") \
_D(IFB_ERR_MAC_DATA_BREAK, 20006, "MAC data break!", "suggest...") \
_D(IFB_ERR_DRV_FIRMWARE_VERSION, 20007, "Motor driver firmware version error!", "suggest...") \
_D(INIT_ERR_EN_DRV, 20008, "Motor driver enable failed!", "suggest...") \
_D(INIT_ERR_EN_AUTO_BACK, 20009, "Motor driver enable auto back failed!", "suggest...") \
_D(INIT_ERR_EN_CUR_LOOP, 20010, "Motor driver enable current loop failed!", "suggest...") \
_D(INIT_ERR_SET_TAG_CUR, 20011, "Motor driver set target current failed!", "suggest...") \
_D(INIT_ERR_RELEASE_BRAKE, 20012, "Motor driver release brake failed!", "suggest...") \
_D(INIT_ERR_EN_POS_LOOP, 20013, "Motor driver enable position loop failed!", "suggest...") \
_D(INIT_ERR_SET_MAX_ACC, 20014, "Motor set max accelerate failed!", "suggest...") \
_D(SAFETY_ERR_PROTECTION_STOP_TIMEOUT, 20015, "Protective stop timeout!", "suggest...") \
_D(SAFETY_ERR_REDUCED_MODE_TIMEOUT, 20016, "Reduced mode timeout!", "suggest...") \
_D(SYS_ERR_MCU_COM, 20017, "Robot system error: mcu communication error!", "suggest...") \
_D(SYS_ERR_RS485_COM, 20018, "Robot system error: RS485 communication error!", "suggest...") \
_D(IFB_ERR_DISCONNECTED, 20019, "Interface board may be disconnected. Please check connection between IPC and Interface board.", "suggest...") \
_D(IFB_ERR_PAYLOAD_ERROR, 20020, "Payload error.", "suggest...") \
_D(IFB_OFFLINE, 20021, "ifaceboard error: ifaceboard may be offline", "(a) Check ifaceboard's hardware. (b) Check ifaceboard's id.") \
_D(IFB_ERR_BOOTLOADER, 20022, "ifaceboard error: The ifaceboard is in bootloader mode. Retry firmware update. ", "suggest...") \
_D(IFB_SLAVE_OFFLINE, 20023, "interface slave board error: interface slave board may be offline", "(a) Check interface slave board's hardware. (b) Check interface slave board's id.") \
_D(IFB_SLAVE_ERR_BOOTLOADER, 20024, "interface slave board error: The interface slave board is in bootloader mode. Retry firmware update. ", "suggest...") \
_D(IFB_TOOL_ERR_ADXL_SENS, 20025, "Tool Acceleration sensor error!", "suggest...")
```

Definition at line 94 of file [error_stack.h](#).

12.8.2.6 TOOL_ERRORS

```
#define TOOL_ERRORS
```

Value:

```
_D(TOOL_FLASH_VERIFY_FAILED, 40001, "Flash write verify failed", "suggest...") \
_D(TOOL_PROGRAM_CRC_FAILED, 40002, "Program flash checksum failed during bootloading", "suggest...") \
_D(TOOL_PROGRAM_CRC_FAILED2, 40003, "Program flash checksum failed at runtime", "suggest...") \
_D(TOOL_ID_UNDEFINED, 40004, "Tool ID is undefined", "suggest...") \
_D(TOOL_ILLEGAL_BL_CMD, 40005, "Illegal bootloader command", "suggest...") \
_D(TOOL_FW_WRONG, 40006, "Wrong firmware at the joint", "suggest...") \
_D(TOOL_HW_INVALID, 40007, "Invalid hardware revision", "suggest...") \
_D(TOOL_SHORT_CURCUI_T_H, 40011, "Short circuit detected on Digital Output: " _PH1_ " high side", "suggest...") \
_D(TOOL_SHORT_CURCUI_T_L, 40012, "Short circuit detected on Digital Output: " _PH1_ " low side", "suggest...") \
_D(TOOL_AVERAGE_CURR_HIGH, 40013, "10 second Average tool IO Current of " _PH1_ " A is outside of the allowed range.", "suggest...") \
_D(TOOL_POWER_PIN_OVER_CURR, 40014, "Current of " _PH1_ " A on the POWER pin is outside of the allowed range.", "suggest...") \
_D(TOOL_DOUT_PIN_OVER_CURR, 40015, "Current of " _PH1_ " A on the Digital Output pins is outside of the allowed range.", "suggest...") \
_D(TOOL_GROUND_PIN_OVER_CURR, 40016, "Current of " _PH1_ " A on the ground pin is outside of the allowed range.", "suggest...") \
_D(TOOL_RX_FRAMING, 40021, "RX framing error", "suggest...") \
_D(TOOL_RX_PARITY, 40022, "RX Parity error", "suggest...") \
_D(TOOL_48V_LOW, 40031, "48V input is too low", "suggest...") \
_D(TOOL_48V_HIGH, 40032, "48V input is too high", "suggest...") \
_D(TOOL_ERR_OFFLINE, 40033, "tool error: tool may be offline", "(a) Check tool's hardware. (b) Check joint's id.") \
_D(TOOL_ERR_BOOTLOADER, 40034, "tool error: The tool is in bootloader mode. Retry firmware update. ", "suggest...")
```

Definition at line 66 of file [error_stack.h](#).

12.9 hal_error.h

[Go to the documentation of this file.](#)

```
00001 /** @file hal_error.h
00002  * @brief
00003  */
00004 #ifndef AUBO_SDK_HAL_ERROR_H
00005 #define AUBO_SDK_HAL_ERROR_H
00006
00007 //
00008 // JNT: joint
00009 // PDL: pedstral
00010 // TP: teach pendant
00011 // COMM: communication
00012 // ENC: encoder
00013 // CURR: current
00014 // POS: position
00015 // PKG: package
00016 // PROG: program
00017
00018 // clang-format off
00019 #define JOINT_ERRORS \
00020 _D(JOINT_ERR_OVER_CURENET, 10001, "joint" _PH1_ " error: over current", "(a) Check for short circuit. (b) Do a Complete rebooting sequence. (c) If this happens more than two times in a row, replace joint") \
00021 _D(JOINT_ERR_OVER_VOLTAGE, 10002, "joint" _PH1_ " error: over voltage", "(a) Do a Complete rebooting sequence. (b) Check 48 V Power supply, current distributor, energy eater and Control Board for issues") \
00022 _D(JOINT_ERR_LOW_VOLTAGE, 10003, "joint" _PH1_ " error: low voltage", "(a) Do a Complete rebooting sequence. (b) Check for short circuit in robot arm. (c) Check 48 V Power supply, current distributor, energy eater and Control Board for issues") \
00023 _D(JOINT_ERR_OVER_TEMP, 10004, "joint" _PH1_ " error: over temperature", "(a) Check robot's environment and make sure the robot is operating within recommended limits. (b) Do a Complete rebooting sequence") \
00024 _D(JOINT_ERR_HALL, 10005, "joint" _PH1_ " error: hall", "suggest...") \
00025 _D(JOINT_ERR_ENCODER, 10006, "joint" _PH1_ " error: encoder", "Check encoder connections") \
00026 _D(JOINT_ERR_ABS_ENCODER, 10007, "joint" _PH1_ " error: abs encoder", "suggest...") \
00027 _D(JOINT_ERR_Q_CURRENT, 10008, "joint" _PH1_ " error: detect current", "suggest...") \
00028 _D(JOINT_ERR_ENC_POLL, 10009, "joint" _PH1_ " error: encoder pollustion", "suggest...") \
00029 _D(JOINT_ERR_ENC_Z_SIGNAL, 10010, "joint" _PH1_ " error: enocder z signal", "suggest...") \
00030 _D(JOINT_ERR_ENC_CAL, 10011, "joint" _PH1_ " error: encoder calibrate", "suggest...") \
00031 _D(JOINT_ERR_IMU_SENS, 10012, "joint" _PH1_ " error: IMU sensor", "suggest...") \
00032 _D(JOINT_ERR_TEMP_SENS, 10013, "joint" _PH1_ " error: TEMP sensor", "suggest...") \
00033 _D(JOINT_ERR_CAN_BUS, 10014, "joint" _PH1_ " error: CAN bus error", "suggest...") \
00034 _D(JOINT_ERR_SYS_CUR, 10015, "joint" _PH1_ " error: system current error", "suggest...") \
00035 _D(JOINT_ERR_SYS_POS, 10016, "joint" _PH1_ " error: system position error", "suggest...") \
00036 _D(JOINT_ERR_OVER_SP, 10017, "joint" _PH1_ " error: over speed", "suggest...") \
00037 _D(JOINT_ERR_OVER_ACC, 10018, "joint" _PH1_ " error: over accelerate", "suggest...") \
00038 _D(JOINT_ERR_TRACE, 10019, "joint" _PH1_ " error: trace accuracy", "suggest...") \
```

```

00039 _D(JOINT_ERR_TAG_POS_OVER, 10020, "joint" _PH1_ " error: target position out of range", "suggest...") \
00040 _D(JOINT_ERR_TAG_SP_OVER, 10021, "joint" _PH1_ " error: target speed out of range", "suggest...") \
00041 _D(JOINT_ERR_COLLISION, 10022, "joint" _PH1_ " error: collision", "suggest...") \
00042 _D(JOINT_ERR_COMMON, 10023, "joint" _PH1_ " error: unkown error. Check communication with joint.",
"suggest...") \
00043 _D(JOINT_ERR_SWITCH_SERVO_MODE, 10024, "joint" _PH1_ " error: switch servo mode timeout.", "suggest...") \
00044 _D(JOINT_ERR_MOTOR_STUCK, 10025, "joint" _PH1_ " error: motor stucked.", "suggest...") \
00045 _D(JOINT_ERR_REDUCER_OVER_TEMP, 10026, "joint" _PH1_ " error: reducer over temperature", "(a) Check
robot's environment and make sure the robot is operating within recommended limits. (b) Do a Complete rebooting
sequence") \
00046 _D(JOINT_ERR_REDUCER_NTC, 10027, "joint" _PH1_ " error: reducer TEMP sensor failure", "(a) Check robot's
environment and make sure the robot is operating within recommended limits. (b) Do a Complete rebooting sequence") \
00047 _D(JOINT_ERR_ABS_MULTITURN, 10028, "joint" _PH1_ " error: absolute encoder multiturn error", "(a) Check
robot's environment and make sure the robot is operating within recommended limits. (b) Do a Complete rebooting
sequence") \
00048 _D(JOINT_ERR_ADC_ZERO_OFFSET, 10029, "joint" _PH1_ " error: ADC zero offset failure", "(a) Check robot's
environment and make sure the robot is operating within recommended limits. (b) Do a Complete rebooting sequence") \
00049 _D(JOINT_ERR_SHORT_CIRCUIT, 10030, "joint" _PH1_ " error: short circuit", "(a) Check robot's environment
and make sure the robot is operating within recommended limits. (b) Do a Complete rebooting sequence") \
00050 _D(JOINT_ERR_PHASE_LOST, 10031, "joint" _PH1_ " error: motor phase lost", "(a) Check robot's environment
and make sure the robot is operating within recommended limits. (b) Do a Complete rebooting sequence") \
00051 _D(JOINT_ERR_BRAKE, 10032, "joint" _PH1_ " error: brake failure", "(a) Check robot's environment and make
sure the robot is operating within recommended limits. (b) Do a Complete rebooting sequence") \
00052 _D(JOINT_ERR_FIRMWARE_UPDATE, 10033, "joint" _PH1_ " error: firmware update failure", "(a) Check robot's
environment and make sure the robot is operating within recommended limits. (b) Do a Complete rebooting sequence") \
00053 _D(JOINT_ERR_BATTERY_LOW, 10034, "joint" _PH1_ " error: battery low", "(a) Check robot's environment and
make sure the robot is operating within recommended limits. (b) Do a Complete rebooting sequence") \
00054 _D(JOINT_ERR_PHASE_ALIGN, 10035, "joint" _PH1_ " error: phase align", "(a) Check robot's environment and
make sure the robot is operating within recommended limits. (b) Do a Complete rebooting sequence") \
00055 _D(JOINT_ERR_CAN_HW_FAULT, 10036, "joint" _PH1_ " error: CAN bus hw fault", "(a) Check robot's
environment and make sure the robot is operating within recommended limits. (b) Do a Complete rebooting sequence") \
00056 _D(JOINT_ERR_POS_DISCONTINUOUS, 10037, "joint" _PH1_ " error: target position discontinuous", "(a) Check
robot's environment and make sure the robot is operating within recommended limits. (b) Do a Complete rebooting
sequence") \
00057 _D(JOINT_ERR_POS_INIT, 10038, "joint" _PH1_ " error: position initiallization failure", "(a) Check robot's
environment and make sure the robot is operating within recommended limits. (b) Do a Complete rebooting sequence") \
00058 _D(JOINT_ERR_TORQUE_SENSOR, 10039, "joint" _PH1_ " error: torqure sensor failure", "(a) Check robot's
environment and make sure the robot is operating within recommended limits. (b) Do a Complete rebooting sequence") \
00059 _D(JOINT_ERR_OFFLINE, 10040, "joint" _PH1_ " error: joint may be offline", "(a) Check joint's hardware. (b)
Check joint's id.") \
00060 _D(JOINT_ERR_BOOTLOADER, 10041, "joint" _PH1_ " error: The joint is in bootloader mode. Retry firmware
update. ", "suggest...") \
00061 _D(JOINT_ERR_SLAVE_OFFLINE, 10042, "slave joint" _PH1_ " error: slave joint may be offline", "(a) Check
slave joint's hardware. (b) Check slave joint's id.") \
00062 _D(JOINT_ERR_SLAVE_BOOTLOADER, 10043, "slave joint" _PH1_ " error: The slave joint is in bootloader
mode. Retry firmware update. ", "suggest...") \
00063 _D(JOINT_ERR_ETHERCAT_BUS, 10044, "joint" _PH1_ " error: ETHERCAT bus error", "suggest...")
00064
00065 #define TOOL_ERRORS \
00066 _D(TOOL_FLASH_VERIFY_FAILED, 40001, "Flash write verify failed", "suggest...") \
00067 _D(TOOL_PROGRAM_CRC_FAILED, 40002, "Program flash checksum failed during bootloading", "suggest...") \
00068 _D(TOOL_PROGRAM_CRC_FAILED2, 40003, "Program flash checksum failed at runtime", "suggest...") \
00069 _D(TOOL_ID_UNDEFINED, 40004, "Tool ID is undefined", "suggest...") \
00070 _D(TOOL_ILLEGAL_BL_CMD, 40005, "Illegal bootloader command", "suggest...") \
00071 _D(TOOL_FW_WRONG, 40006, "Wrong firmware at the joint", "suggest...") \
00072 _D(TOOL_HW_INVALID, 40007, "Invalid hardware revision", "suggest...") \
00073 _D(TOOL_SHORT_CURCUIT_H, 40011, "Short circuit detected on Digital Output: " _PH1_ " high side",
"suggest...") \
00074 _D(TOOL_SHORT_CURCUIT_L, 40012, "Short circuit detected on Digital Output: " _PH1_ " low side",
"suggest...") \
00075 _D(TOOL_AVERAGE_CURR_HIGH, 40013, "10 second Average tool IO Current of " _PH1_ " A is outside of the
allowed range.", "suggest...") \
00076 _D(TOOL_POWER_PIN_OVER_CURR, 40014, "Current of " _PH1_ " A on the POWER pin is outside of the
allowed range.", "suggest...") \
00077 _D(TOOL_DOUT_PIN_OVER_CURR, 40015, "Current of " _PH1_ " A on the Digital Output pins is outside of the
allowed range.", "suggest...") \
00078 _D(TOOL_GROUND_PIN_OVER_CURR, 40016, "Current of " _PH1_ " A on the ground pin is outside of the
allowed range.", "suggest...") \
00079 _D(TOOL_RX_FRAMING, 40021, "RX framing error", "suggest...") \
00080 _D(TOOL_RX_PARITY, 40022, "RX Parity error", "suggest...") \
00081 _D(TOOL_48V_LOW, 40031, "48V input is too low", "suggest...") \
00082 _D(TOOL_48V_HIGH, 40032, "48V input is too high", "suggest...") \
00083 _D(TOOL_ERR_OFFLINE, 40033, "tool error: tool may be offline", "(a) Check tool's hardware. (b) Check joint's
id.") \
00084 _D(TOOL_ERR_BOOTLOADER, 40034, "tool error: The tool is in bootloader mode. Retry firmware update. ",
"suggest...")
00085
00086
00087 #define PEDSTRAL_ERRORS \
00088 _D(PKG_LOST, 50001, "Lost package from pedestal", "suggest...") \
00089 _D(PEDSTRAL_OFFLINE, 50002, "pedestal error: pedestal may be offline", "(a) Check pedestal's hardware. (b)
Check pedestal's id.") \
00090 _D(PEDESTAL_ERR_BOOTLOADER, 50003, "pedestal error: The pedestal is in bootloader mode. Retry firmware
update. ", "suggest...")
00091
00092

```



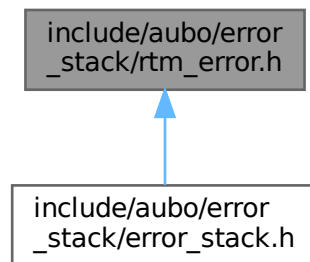
```

00093 #define SAFETY_INTERFACE_BOARD_ERRORS \
00094     _D(IFB_ERR_ROBOTTYPE, 20001, "Robot error type!", "suggest...") \
00095     _D(IFB_ERR_ADXL_SENS, 20002, "Base Acceleration sensor error!", "suggest...") \
00096     _D(IFB_ERR_EN_LINE, 20003, "Encoder line error!", "suggest...") \
00097     _D(IFB_ERR_ENTER_HDG_MODE, 20004, "Robot enter handguide mode!", "suggest...") \
00098     _D(IFB_ERR_EXIT_HDG_MODE, 20005, "Robot exit handguide mode!", "suggest...") \
00099     _D(IFB_ERR_MAC_DATA_BREAK, 20006, "MAC data break!", "suggest...") \
00100     _D(IFB_ERR_DRV_FIRMWARE_VERSION, 20007, "Motor driver firmware version error!", "suggest...") \
00101     _D(INIT_ERR_EN_DRV, 20008, "Motor driver enable failed!", "suggest...") \
00102     _D(INIT_ERR_EN_AUTO_BACK, 20009, "Motor driver enable auto back failed!", "suggest...") \
00103     _D(INIT_ERR_EN_CUR_LOOP, 20010, "Motor driver enable current loop failed!", "suggest...") \
00104     _D(INIT_ERR_SET_TAG_CUR, 20011, "Motor driver set target current failed!", "suggest...") \
00105     _D(INIT_ERR_RELEASE_BRAKE, 20012, "Motor driver release brake failed!", "suggest...") \
00106     _D(INIT_ERR_EN_POS_LOOP, 20013, "Motor driver enable position loop failed!", "suggest...") \
00107     _D(INIT_ERR_SET_MAX_ACC, 20014, "Motor set max accelerate failed!", "suggest...") \
00108     _D(SAFETY_ERR_PROTECTION_STOP_TIMEOUT, 20015, "Protective stop timeout!", "suggest...") \
00109     _D(SAFETY_ERR_REDUCED_MODE_TIMEOUT, 20016, "Reduced mode timeout!", "suggest...") \
00110     _D(SYS_ERR_MCU_COM, 20017, "Robot system error: mcu communication error!", "suggest...") \
00111     _D(SYS_ERR_RS485_COM, 20018, "Robot system error: RS485 communication error!", "suggest...") \
00112     _D(IFB_ERR_DISCONNECTED, 20019, "Interface board may be disconnected. Please check connection between IPC
and Interface board.", "suggest...") \
00113     _D(IFB_ERR_PAYLOAD_ERROR, 20020, "Payload error.", "suggest...") \
00114     _D(IFB_ERR_OFFLINE, 20021, "ifaceboard error: ifaceboard may be offline", "(a) Check ifaceboard's hardware. (b) Check
ifaceboard's id.") \
00115     _D(IFB_ERR_BOOTLOADER, 20022, "ifaceboard error: The ifaceboard is in bootloader mode. Retry firmware
update.", "suggest...") \
00116     _D(IFB_SLAVE_OFFLINE, 20023, "interface slave board error: interface slave board may be offline", "(a) Check
interface slave board's hardware. (b) Check interface slave board's id.") \
00117     _D(IFB_SLAVE_ERR_BOOTLOADER, 20024, "interface slave board error: The interface slave board is in
bootloader mode. Retry firmware update.", "suggest...") \
00118     _D(IFB_TOOL_ERR_ADXL_SENS, 20025, "Tool Acceleration sensor error!", "suggest...")
00119
00120 #define HARDWARE_INTERFACE_ERRORS \
00121     _D(HW_SCB_SETUP_FAILED, 60001, "Setup of Interface Board failed", "suggest...") \
00122     _D(HW_PKG_CNT_DISAGREE, 60002, "Packet counter disagreements", "suggest...") \
00123     _D(HW_SCB_DISCONNECT, 60003, "Connection to Interface Board lost", "suggest...") \
00124     _D(HW_SCB_PKG_LOST, 60004, "Package lost from Interface Board", "suggest...") \
00125     _D(HW_SCB_CONN_INIT_FAILED, 60005, "Ethernet connection initialization with Interface Board failed",
"suggest...") \
00126     _D(HW_LOST_JOINT_PKG, 60006, "Lost package from joint " _PH1_ "", "suggest...") \
00127     _D(HW_LOST_TOOL_PKG, 60007, "Lost package from tool", "suggest...") \
00128     _D(HW_JOINT_PKG_CNT_DISAGREE, 60008, "Packet counter disagreement in packet from joint " _PH1_ "",
"suggest...") \
00129     _D(HW_TOOL_PKG_CNT_DISAGREE, 60009, "Packet counter disagreement in packet from tool", "suggest...") \
00130     _D(HW_JOINTS_FAULT, 60011, "" _PH1_ " joint entered the Fault State", "suggest...") \
00131     _D(HW_JOINTS_VIOLATION, 60012, "" _PH1_ " joint entered the Violation State", "suggest...") \
00132     _D(HW_TP_FAULT, 60013, "Teach Pendant entered the Fault State", "suggest...") \
00133     _D(HW_TP_VIOLATION, 60014, "Teach Pendant entered the Violation State", "suggest...") \
00134     _D(HW_JOINT_MV_TOO_FAR, 60021, "" _PH1_ " joint moved too far before robot entered RUNNING State",
"suggest...") \
00135     _D(HW_JOINT_STOP_NOT_FAST, 60022, "Joint Not stopping fast enough", "suggest...") \
00136     _D(HW_JOINT_MV_LIMIT, 60023, "Joint moved more than allowable limit", "suggest...") \
00137     _D(HW_FT_SENSOR_DATA_INVALID, 60024, "Force-Torque Sensor data invalid", "suggest...") \
00138     _D(HW_NO_FT_SENSOR, 60025, "Force-Torque sensor is expected, but it cannot be detected", "suggest...") \
00139     _D(HW_FT_SENSOR_NOT_CALIB, 60026, "Force-Torque sensor is detected but not calibrated", "suggest...") \
00140     _D(HW_RELEASE_BRAKE_FAILED, 60030, "Robot was not able to brake release, see log for details", "suggest...") \
00141     _D(HW_OVERCURREN_SHUTDOWN, 60040, "Overcurrent shutdown", "suggest...") \
00142     _D(HW_ENERGY_SURPLUS, 60050, "Energy surplus shutdown", "suggest...") \
00143     _D(HW_IDLE_POWER_HIGH, 60060, "Idle power consumption to high", "suggest...") \
00144     _D(HW_ENTER_COLLISION_TIMEOUT, 60071, "Enter collision stop procedure timeout", "suggest...") \
00145     _D(HW_POWERON_TIMEOUT, 60072, "Poweron robot timeout", "suggest...") \
00146     _D(HW_NO_NIC_FOUND, 60073, "No network cards found.", "suggest...") \
00147     _D(HW_IFB_NOT_FOUND, 60074, "No Interface Board found.", "suggest...") \
00148     _D(HW_IFB_BOOTLOAD, 60075, "The Interface Board is in bootloader mode. Update firmware firstly.", "suggest...")
00149
00150     _D(HW_TOOL_NOT_FOUND, 60076, "No Tool Board found.", "suggest...") \
00151     _D(HW_BASE_NOT_FOUND, 60077, "No Base Board found.", "suggest...") \
00152     _D(HW_BRINGUP_TIMEOUT, 60078, "Poweron robot timeout", "suggest...") \
00153     _D(HW_COLLISION_RECOVERY_FAILED, 60079, "Collision recovery failed", "suggest...") \
00154     _D(HW_TP_ENABLED, 60080, "Teach pendant enabled status changed to " _PH1_ "", "suggest...")
00155
00156 // clang-format on
00157
00158 //
00159 #define HAL_ERRORS \
00160     JOINT_ERRORS \
00161     SAFETY_INTERFACE_BOARD_ERRORS \
00162     TOOL_ERRORS \
00163     PEDSTRAL_ERRORS \
00164     HARDWARE_INTERFACE_ERRORS
00165
00166 #endif // AUBO_SDK_JOINT_ERROR_H

```


12.10 include/aubo/error_stack/rtm_error.h File Reference

This graph shows which files directly or indirectly include this file:



Macros

- `#define` [RTM_ERRORS](#)

12.10.1 Detailed Description

Definition in file [rtm_error.h](#).

12.10.2 Macro Definition Documentation

12.10.2.1 RTM_ERRORS

```
#define RTM_ERRORS
```

Definition at line [10](#) of file [error_stack.h](#).

12.11 rtm_error.h

[Go to the documentation of this file.](#)

```

00001 /** @file rtm_error.h
00002  * @brief
00003  */
00004 #ifndef AUBO_SDK_RTM_ERROR_H
00005 #define AUBO_SDK_RTM_ERROR_H
00006
00007 // clang-format off
00008
00009 #define RTM_ERRORS \
00010     _D(ROBOT_BE_PULLING, 30001, "Something is pulling the robot.", "Please check TCP configuration, payload and
00011     mounting settings") \
00012     _D(PSTOP_ELBOW_POS, 30002, "Protective Stop: Elbow position close to safety plane limits.", "Please move robot
00013     Elbow joint away from the safety plane") \
00014     _D(PSTOP_STOP_TIME, 30003, "Protective Stop: Exceeding user safety settings for stopping time.", "(a) Check
00015     speeds and accelerations in the program (b) Check usage of TCP, payload and CoG correctly (c) Check external
00016     equipment activation if correctly set") \
00017     _D(PSTOP_STOP_DISTANCE, 30004, "Protective Stop: Exceeding user safety settings for stopping distance.", "(a)
00018     Check speeds and accelerations in the program (b) Check usage of TCP, payload and CoG correctly (c) Check external
00019     equipment activation if correctly set") \
00020     _D(PSTOP_CLAMP, 30005, "Protective Stop: Danger of clamping between the Robot's lower arm and tool.", "(a)
00021     Check speeds and accelerations in the program (b) Check usage of TCP, payload and CoG correctly (c) Check external
00022     equipment activation if correctly set") \
00023     _D(PSTOP_POS_LIMIT, 30006, "Protective Stop: Position close to joint limits", "suggest...") \
00024     _D(PSTOP_ORI_LIMIT, 30007, "Protective Stop: Tool orientation close to limits", "suggest...") \
00025     _D(PSTOP_PLANE_LIMIT, 30008, "Protective Stop: Position close to safety plane limits", "suggest...") \
00026     _D(PSTOP_POS_DEVIATE, 30009, "Protective Stop: Position deviates from path", "Check payload, center of gravity
00027     and acceleration settings.") \
00028     _D(JOINT_CHK_PAYLOAD, 30010, "Joint " _PH1_ ": Check payload, center of gravity and acceleration settings.
00029     Log screen may contain additional information.", "suggest...") \
00030     _D(PSTOP_SINGULARITY, 30011, "Protective Stop: Position in singularity.", "Please use MoveJ or change the
00031     motion") \
00032     _D(PSTOP_CANNOT_MAINTAIN, 30012, "Protective Stop: Robot cannot maintain its position, check if payload is
00033     correct", "suggest...") \
00034     _D(PSTOP_WRONG_PAYLOAD, 30013, "Protective Stop: Wrong payload or mounting detected, or something is
00035     pushing the robot when entering Freedrive mode", "Verify that the TCP configuration and mounting in the used
00036     installation is correct") \
00037     _D(PSTOP_JOINT_COLLISION, 30014, "Protective Stop: Collision detected by joint " _PH1_, "Make sure no
00038     objects are in the path of the robot and resume the program") \
00039     _D(PSTOP_POS_DISAGREE, 30015, "Protective stop: The robot was powered off last time due to a joint position
00040     disagreement.", "(a) Verify that the robot position in the 3D graphics matches the real robot, to ensure that the encoders
00041     function before releasing the brakes. Stand back and monitor the robot performing its first program cycle as expected. (b)
00042     If the position is not correct, the robot must be repaired. In this case, click Power Off Robot. (c) If the position is correct,
00043     please tick the check box below the 3D graphics and click Robot Position Verified") \
00044     _D(TARGET_JOINT_SPEED_EXCEED, 30016, "Target joint speed exceed limits", "suggest...") \
00045     _D(TARGET_POS_SUDDEN_CHG, 30017, "Sudden change in target position", "suggest...") \
00046     _D(SUDDEN_STOP, 30018, "Sudden stop.", "To abort a motion, use \"stopj\" or \"stopl\" script commands to
00047     generate a smooth deceleration before using \"wait\". Avoid aborting motions between waypoints with blend") \
00048     _D(ROBOT_STOP_ABNORMAL, 30019, "Robot has not stopped in the allowed reaction and braking time",
00049     "suggest...") \
00050     _D(PROG_INVALID_SETP, 30020, "Robot program resulted in invalid setpoint.", "Please review waypoints in the
00051     program") \
00052     _D(BLEND_INVALID_SETP, 30021, "Blending failed and resulted in an invalid setpoint.", "Try changing the blend
00053     radius or contact technical support") \
00054     _D(APPROACH_SINGULARITY, 30022, "Robot approaching singularity - Acceleration threshold failed.", "Review
00055     waypoints in the program, try using MoveJ instead of MoveL in the position close to singularity") \
00056     _D(TSPEED_UNMATCH_POS, 30023, "Target speed does not match target position", "suggest...") \
00057     _D(INCONSIS_TPOS_SPD, 30024, "Inconsistency between target position and speed", "suggest...") \
00058     _D(JOINT_TSPD_UNMATCH_POS, 30025, "Target joint speed does not match target joint position change - Joint "
00059     _PH1_ " ", "suggest...") \
00060     _D(FIELDBUS_INPUT_DISCONN, 30026, "Fieldbus input disconnected.", "Please check fieldbus connections (RTDE,
00061     ModBus, EtherNet/IP and Profinet) or disable the fieldbus in the installation. Check RTDE watchdog feature. Check if a
00062     URcap is using this feature.") \
00063     _D(OPMODE_CHANGED, 30027, "Operational mode changed: " _PH1_ " ", "suggest...") \
00064     _D(NO_KIN_CALIB, 30028, "No Kinematic Calibration found (calibration.conf file is either corrupt or missing).", "A
00065     new kinematics calibration may be needed if the robot needs to improve its kinematics, otherwise, ignore this message") \
00066     _D(KIN_CALIB_UNMATCH_JOINT, 30029, "Kinematic Calibration for the robot does not match the joint(s).", "If
00067     moving a program from a different robot to this one, rekinematic calibrate the second robot to improve kinematics,
00068     otherwise ignore this message.") \
00069     _D(KIN_CALIB_UNMATCH_ROBOT, 30030, "Kinematic Calibration does not match the robot.", "Please check if the
00070     serial number of the robot arm matches the Control Box") \
00071     _D(JOINT_OFFSET_CHANGED, 30031, "Large movement of the robot detected while it was powered off. The joints
00072     were moved while it was powered off, or the encoders do not function", "suggest...") \
00073     _D(OFFSET_CHANGE_HIGH, 30032, "Change in offset is too high", "suggest...") \
00074     _D(JOINT_SPEED_LIMIT, 30033, "Close to joint speed safety limit.", "Review program speed and acceleration") \
00075     _D(TOOL_SPEED_LIMIT, 30034, "Close to tool speed safety limit.", "Review program speed and acceleration") \
00076     _D(MOMENTUM_LIMIT, 30035, "Close to momentum safety limit.", "Review program speed and acceleration") \
00077     _D(ROBOT_MV_STOP, 30036, "Robot is moving when in Stop Mode", "suggest...") \
00078     _D(HAND_PROTECTION, 30037, "Hand protection: Tool is too close to the lower arm: " _PH1_ " meter.", "(a)
00079     Check wrist position. (b) Verify mounting (c) Do a Complete rebooting sequence (d) Update software (e) Contact your
00080     local AUBO Robots service provider for assistance") \
00081     _D(WRONG_SAFETYMODE, 30038, "Wrong safety mode: " _PH1_, "suggest...") \
00082     _D(SAFETYMODE_CHANGED, 30039, "Safety mode changed: " _PH1_, "suggest...") \

```

```

00049 _D(JOINT_ACC_LIMIT, 30040, "Close to joint acceleration safety limit", "suggest...") \
00050 _D(TOOL_ACC_LIMIT, 30041, "Close to tool acceleration safety limit", "suggest...") \
00051 _D(JOINT_TEMPERATURE_LIMIT, 30042, "Joint " _PH1_ " temperature too high(> " _PH2_ "°C)", "suggest...") \
00052 _D(CONTROL_BOX_TEMPERATURE_LIMIT, 30043, "Control box temperature too high(> " _PH1_ "°C)",
"suggest...") \
00053 _D(ROBOT_EMERGENCY_STOP, 30044, "Robot emergency stop", "suggest...") \
00054 _D(ROBOTMODE_CHANGED, 30045, "Robot mode changed: " _PH1_, "suggest...") \
00055 _D(ROBOTMODE_ERROR, 30046, "Wrong robot mode: " _PH1_, "suggest...") \
00056 _D(POSE_OUT_OF_REACH, 30047, "Target pose [ " _PH1_ " ] out of reach", "suggest...") \
00057 _D(TP_PLAN_FAILED, 30048, "Trajectory plan FAILED.", "suggest...") \
00058 _D(START_FORCE_FAILED, 30049, "Start force control failed, because force sensor does not exist.", "suggest...") \
00059 _D(OVER_SAFE_PLANE_LIMIT, 30050, _PH1_ " axis exceeds the safety plane limit (Move_type: " _PH2_ " id: "
_PH3_ ").", "Please move the robot to the safety plane range.") \
00060 _D(POWERON_FAIL_VIOLATION, 30051, "Failed to power on because the robot safety mode is in violation",
"suggest...") \
00061 _D(POWERON_FAIL_SYSTEMEMERGENCYSTOP, 30052, "Failed to power on because the robot safety mode is in
system emergency stop", "suggest...") \
00062 _D(POWERON_FAIL_ROBOTEMERGENCYSTOP, 30053, "Failed to power on because the robot safety mode is in
robot emergency stop", "Pop up the red emergency stop button on the teach pendant when the robot is in a safe range of
motion") \
00063 _D(POWERON_FAIL_FAULT, 30054, "Failed to power on because the robot safety mode is in fault", "suggest...") \
00064 _D(STARTUP_FAIL_VIOLATION, 30055, "Failed to startup because the robot safety mode is in violation",
"suggest...") \
00065 _D(STARTUP_FAIL_SYSTEMEMERGENCYSTOP, 30056, "Failed to startup because the robot safety mode is in
system emergency stop", "suggest...") \
00066 _D(STARTUP_FAIL_ROBOTEMERGENCYSTOP, 30057, "Failed to startup because the robot safety mode is in
robot emergency stop", "Pop up the red emergency stop button on the teach pendant when the robot is in a safe range of
motion") \
00067 _D(STARTUP_FAIL_FAULT, 30058, "Failed to startup because the robot safety mode is in fault", "suggest...") \
00068 _D(BACKDRIVE_FAIL_VIOLATION, 30059, "Failed to backdrive because the robot safety mode is in violation",
"suggest...") \
00069 _D(BACKDRIVE_FAIL_SYSTEMEMERGENCYSTOP, 30060, "Failed to backdrive because the robot safety mode is
in system emergency stop", "suggest...") \
00070 _D(BACKDRIVE_FAIL_ROBOTEMERGENCYSTOP, 30061, "Failed to backdrive because the robot safety mode is in
robot emergency stop", "Pop up the red emergency stop button on the teach pendant when the robot is in a safe range of
motion") \
00071 _D(BACKDRIVE_FAIL_FAULT, 30062, "Failed to backdrive because the robot safety mode is in fault", "suggest...") \
00072 _D(SETSIM_FAIL_VIOLATION, 30063, "Switch sim mode failed because the robot safety mode is in violation",
"suggest...") \
00073 _D(SETSIM_FAIL_SYSTEMEMERGENCYSTOP, 30064, "Switch sim mode failed because the robot safety mode is in
system emergency stop", "suggest...") \
00074 _D(SETSIM_FAIL_ROBOTEMERGENCYSTOP, 30065, "Switch sim mode failed because the robot safety mode is in
robot emergency stop", "Pop up the red emergency stop button on the teach pendant when the robot is in a safe range of
motion") \
00075 _D(SETSIM_FAIL_FAULT, 30066, "Switch sim mode failed because the robot safety mode is in fault", "suggest...") \
00076 _D(FREEDRIVE_FAIL_VIOLATION, 30067, "Enable handguide mode failed because the robot safety mode is in
violation", "suggest...") \
00077 _D(FREEDRIVE_FAIL_SYSTEMEMERGENCYSTOP, 30068, "Enable handguide mode failed because the robot
safety mode is in system emergency stop", "suggest...") \
00078 _D(FREEDRIVE_FAIL_ROBOTEMERGENCYSTOP, 30069, "Enable handguide mode failed because the robot safety
mode is in robot emergency stop", "Pop up the red emergency stop button on the teach pendant when the robot is in a
safe range of motion") \
00079 _D(FREEDRIVE_FAIL_FAULT, 30070, "Enable handguide mode failed because the robot safety mode is in fault",
"suggest...") \
00080 _D(UPFIRMWARE_FAIL_VIOLATION, 30071, "Firmware update failed because the robot safety mode is in
violation", "suggest...") \
00081 _D(UPFIRMWARE_FAIL_SYSTEMEMERGENCYSTOP, 30072, "Firmware update failed because the robot safety
mode is in system emergency stop", "suggest...") \
00082 _D(UPFIRMWARE_FAIL_ROBOTEMERGENCYSTOP, 30073, "Firmware update failed because the robot safety
mode is in robot emergency stop", "Pop up the red emergency stop button on the teach pendant when the robot is in a
safe range of motion") \
00083 _D(UPFIRMWARE_FAIL_FAULT, 30074, "Firmware update failed because the robot safety mode is in fault",
"suggest...") \
00084 _D(SETPERSOSTENT_FAIL_VIOLATION, 30075, "Set persistent parameter failed because the robot safety mode is
in violation", "suggest...") \
00085 _D(SETPERSOSTENT_FAIL_SYSTEMEMERGENCYSTOP, 30076, "Set persistent parameter failed because the
robot safety mode is in system emergency stop", "suggest...") \
00086 _D(SETPERSOSTENT_FAIL_ROBOTEMERGENCYSTOP, 30077, "Set persistent parameter failed because the robot
safety mode is in robot emergency stop", "Pop up the red emergency stop button on the teach pendant when the robot is
in a safe range of motion") \
00087 _D(SETPERSOSTENT_FAIL_FAULT, 30078, "Set persistent parameter failed because the robot safety mode is in
fault", "suggest...") \
00088 _D(SETPERSOSTENT_FAIL_PARAM_ERR, 30079, "Set persistent parameter failed", "(a) Check the parameter
format, whether all are floating point numbers") \
00089 _D(ROBOT_CABLE_DISCONN, 30080, "Robot cable not connected", "(a) Make sure the cable between Control Box
and Robot Arm is correctly connected and it has no damage. (b) Check for loose connections (c) Do a Complete rebooting
sequence (d) Update software (e) Contact your local AUBO Robots service provider for assistance Contact your local
AUBO Robots service provider for assistance.") \
00090 _D(TP_TOO_SHORT, 30081, "The generated trajectory is ignored because it is too short", "(a) Please check if the
added waypoints are coincident (b) If it is an arc movement, please check whether the three points are collinear") \
00091 _D(INV_KIN_FAIL, 30082, "Inverse kinematics solution failed. The target pose may be in a singular position or exceed
the joint limits", "(a) Change the target pose and try moving again") \
00092 _D(FREEDRIVE_ENABLED, 30083, "Freedrive status changed to " _PH1_ "'", "suggest...") \
00093 _D(TP_INV_FAIL_REFERENCE_JOINT_OUT_OF_LIMIT, 30084, "Inverse kinematics solution failed. Reference
angle [ " _PH1_ " ] exceeds joint limit [ " _PH2_ " ].", "suggest...") \
00094 _D(TP_INV_FAIL_NO_SOLUTION, 30085, "Inverse kinematics solution failed. The reference angle [ " _PH1_ " ] and
the target angle [ " _PH2_ " ] are used as parameters. there is no solution in the calculation of the inverse solution

```

```

process.", "suggest...") \
00095   _D(SERVO_FAIL_VIOLATION, 30086, "Switch servo mode failed because the robot safety mode is in violation",
"suggest...") \
00096   _D(SERVO_FAIL_SYSTEMEMERGENCYSTOP, 30087, "Switch servo mode failed because the robot safety mode is
in system emergency stop", "suggest...") \
00097   _D(SERVO_FAIL_ROBOTEMERGENCYSTOP, 30088, "Switch servo mode failed because the robot safety mode is in
robot emergency stop", "Pop up the red emergency stop button on the teach pendant when the robot is in a safe range of
motion") \
00098   _D(SERVO_FAIL_FAULT, 30089, "Switch servo mode failed because the robot safety mode is in fault", "suggest...") \
00099   _D(FREEDRIVE_FAIL_NO_RUNNING, 30090, "Enable handguide mode failed because the robot mode type is "
_PH1_ " (not running)", "suggest...") \
00100   _D(RUNTIME_MACHINE_ERROR, 30091, "The state of the running machine is " _PH1_ ", not " _PH2_ ". "
_PH3_ " function execution failed because the state is wrong.", "suggest...") \
00101   _D(RESUME_FAR_PAUSE_PT, 30092, "Cannot resume from joint position [" _PH1_ "].\nToo far away from
paused point [" _PH2_ "]", "suggest...") \
00102   _D(PAYLOAD_LIGHTER_ERROR, 30093, "The payload setting is too small!", "suggest...") \
00103   _D(PAYLOAD_OVERLOAD_ERROR, 30094, "The payload setting is too large!", "suggest...") \
00104   _D(PAUSE_FAIL_NOT_POSITION_PLAN_MODE, 30095, "This motion does not support the pause function. The
motion is stopping.", "suggest...") \
00105   _D(TP_PLAN_FAILED_CIRCULAR_WAYPOINTS_COINCIDE, 30096, "The planning failed because the three
waypoints of the arc were determined to coincide.", "Check the circular waypoints to make sure they are different.") \
00106   _D(SERVO_WRONG_SAFETYMODE, 30097, "Switch servo mode failed because the robot safety mode is in "
_PH1_ " ", "Check the circular waypoints to make sure they are different.") \
00107   _D(SET_PERSTPARAM_WRONG_SAFETYMODE, 30098, "Set persistent parameter failed because the robot safety
mode is in " _PH1_ ", "suggest...") \
00108   _D(SET_KINPARAM_WRONG_SAFETYMODE, 30099, "Set Kinematics Compensate parameters failed because the
robot safety mode is in " _PH1_ ", "suggest...") \
00109   _D(SET_ROBOT_ZERO_WRONG_SAFETYMODE, 30100, "Set current joint angles to zero failed because the robot
safety mode is in " _PH1_ ", "suggest...") \
00110   _D(UPFIRMWARE_WRONG_SAFETYMODE, 30101, "Firmware update failed because the robot safety mode is in "
_PH1_ ", "suggest...") \
00111   _D(POWERON_WRONG_SAFETYMODE, 30102, "Failed to power on because the robot safety mode is in " _PH1_ ",
"suggest...") \
00112   _D(STARTUP_WRONG_SAFETYMODE, 30103, "Failed to startup because the robot safety mode is in " _PH1_ ",
"suggest...") \
00113   _D(BACKDRIVE_WRONG_SAFETYMODE, 30104, "Failed to backdrive because the robot safety mode is in system
emergency stop", "suggest...") \
00114   _D(SETSIM_WRONG_SAFETYMODE, 30105, "Switch sim mode failed because the robot safety mode is in
violation", "suggest...") \
00115   _D(FREEDRIVE_WRONG_SAFETYMODE, 30106, "Enable handguide mode failed because the robot safety mode is
in wrong safety mode: " _PH1_ ", "suggest...") \
00116   _D(TP_PLAN_FAILED_JOINT_JUMP_BIGGER, 30107, "Inverse kinematics solution failed. The target point and
the current point are in different robot configuration spaces.", "Add a few more points between the target point and the
current point.") \
00117   _D(RUN_PROGRAM_FAILED, 30108, "Run program " _PH1_ " failed.", "suggset...") \
00118   _D(FREEDRIVE_FAIL_WRONG_RTMSTATE, 30109, "Unable to enter the HandGuide mode as the robot is not
currently in a stopped or paused state.", "suggset...") \
00119   _D(SAFEGUARDSTOP_CONFIGURABLE_INPUT, 30110, "Configurable safety input is triggered.", "suggset...") \
00120   _D(SAFEGUARDSTOP_3PE, 30111, "3PE is triggered.", "suggset...") \
00121   _D(SAFEGUARDSTOP_SI, 30112, "SIO/SII is triggered.", "suggset...") \
00122   _D(ROBOT_TYPE_CHANGED, 30200, "Robot type changed to " _PH1_ " ', and robot subtype changed to "
_PH2_ " ", "suggest...") \
00123   _D(LINKMODE_CHANGED, 30201, "Link mode changed to " _PH1_ " ", "suggest...") \
00124   _D(ROBOT_SELF_COLLISION, 30301, "Detect risk of robot self collision", "suggest...") \
00125   _D(CONSTANT_INVALID, 30302, "Joint torque constants are invalid. HandGuide will be disabled, and the collision
protection may be triggered by mistake.", "suggest...") \
00126   _D(GRAVITY_INVALID, 30303, "Abnormal value of gravity acceleration sensor. HandGuide will be disabled, and the
collision protection may be triggered by mistake.", "suggest...") \
00127   _D(DYNAMICS_INVALID, 30304, "Robot dynamics parameters are invalid. HandGuide will be disabled, and the
collision protection may be triggered by mistake.", "suggest...") \
00128   _D(FRICTION_INVALID, 30305, "Joint friction parameters are invalid. HandGuide will be disabled, and the collision
protection may be triggered by mistake.", "suggest...") \
00129   _D(HANDGUIDE_UNDER_DEVELOP, 30306, "Robot type of " _PH1_ " function under development. HandGuide
will be disabled, and the collision protection may be triggered by mistake.", "suggest...") \
00130   _D(SLOW_DOWN_INFO, 30307, "Slow down level changed to " _PH1_ " (" _PH2_ "%)", "suggest...") \
00131   _D(WRONG_JOINT_DESIGNED_LIMIT, 30308, "Joint designed ranges exceeds ranges read from hardware
interface.", "suggest...") \
00132   _D(FREEDRIVE_IN_SIMULATION, 30309, "Enable handguide mode failed because the robot is in simulation mode.",
"suggest...") \
00133   _D(ROBOT_STOPPING_TIMEOUT, 30310, "Robot stopping timeout.", "suggest...") \
00134   _D(PSTOP_INCORRECT_FORCE_OFFSET, 30311, "Protective Stop: Sudden change in force control target
position. Force sensor offset may be incorrect or force sensor fault.", "suggest...") \
00135   _D(WRONG_JOINT_SAFETY_LIMIT, 30312, "Joint safety ranges exceeds designed ranges.", "suggest...") \
00136   _D(PSTOP_TCP_PLANE_VIOLATION, 30401, "Protective Stop: TCP position close to safety plane limits.",
"suggest...") \
00137   _D(PSTOP_ELBOW_PLANE_VIOLATION, 30402, "Protective Stop: elbow position close to safety plane limits.",
"suggest...") \
00138   _D(PSTOP_JOINT_TORQUE_VIOLATION, 30403, "Protective Stop: joint" _PH1_ " exceeds torque limit.",
"suggest...") \
00139   _D(PSTOP_JOINT_POSITION_VIOLATION, 30404, "Protective Stop: joint" _PH1_ " exceeds position limit.",
"suggest...") \
00140   _D(PSTOP_JOINT_SPEED_VIOLATION, 30405, "Protective Stop: joint" _PH1_ " exceeds speed limit.",
"suggest...") \
00141   _D(PSTOP_TCP_SPEED_VIOLATION, 30406, "Protective Stop: TCP speed close to safety limits.", "suggest...") \
00142   _D(PSTOP_ELBOW_SPEED_VIOLATION, 30407, "Protective Stop: elbow speed close to safety limits.", "suggest...") \
00143   _D(PSTOP_TCP_FORCE_VIOLATION, 30408, "Protective Stop: TCP force close to safety limits.", "suggest...") \

```

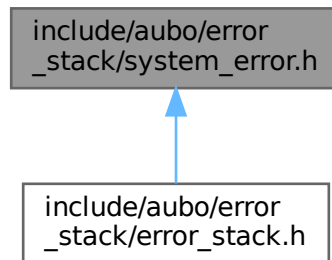
```

00144     _D(PSTOP_ELBOW_TORQUE_VIOLATION, 30409, "Protective Stop: elbow torque close to safety limits.",
"suggest...") \
00145     _D(PSTOP_POWER_VIOLATION, 30410, "Protective Stop: robot power close to safety limits.", "suggest...") \
00146     _D(PSTOP_MOMENTUM_VIOLATION, 30411, "Protective Stop: robot momentum close to safety limits.",
"suggest...") \
00147     _D(PSTOP_TCP_CUBE_VIOLATION, 30412, "Protective Stop: TCP position close to safety cube.", "suggest...") \
00148     _D(PSTOP_ELBOW_CUBE_VIOLATION, 30413, "Protective Stop: TCP position close to safety cube.", "suggest...") \
00149     _D(REDUCE_ELBOW_PLANE_TRIGGER, 30414, "Reduce mode: elbow close to safety plane triggers reduction
mode.", "suggest...") \
00150     _D(REDUCE_TCP_PLANE_TRIGGER, 30415, "Reduce mode: TCP close to safety plane triggers reduction mode.",
"suggest...") \
00151     _D(PSTOP_MOVE_OUT_RANGE, 30416, "Joint " _PH1_ " has exceeded the limit, please do not continue to move
out of the range", "suggest...") \
00152     _D(RESUME_PAUSE_FAILED, 30417, "Resume Failed: Safety mode type is " _PH1_ "", "suggest...") \
00153     _D(FIRMWARE_UPDATE_FAIL_EMERGENCYSTOP, 30418, "Failed to firmware update because the robot safety
mode is in " _PH1_ , "Release emergency stop when the robot is in a safe range of motion") \
00154     _D(TOOL_SENSOR_CHANGED, 30419, "Tool sensor type changed to " _PH1_ "", "suggest...") \
00155     _D(TOOL_SENSOR_REMOVED, 30420, "Tool sensor is removed.", "suggest...") \
00156     _D(CAL_TARGET_CURRENT_ERR, 30421, "The calculation of the target current failed. Please try again later.",
"suggest...") \
00157     _D(CONVEYOR_MODE_CHANGED, 30422, "Conveyor" _PH1_ ": track mode changed to " _PH2_ , track item id
is " _PH3_ , "suggest...") \
00158     _D(CONVEYOR_ENQUEUE, 30423, "Conveyor" _PH1_ ": the queue has been changed, item" _PH2_ " is enqueue",
"suggest...") \
00159     _D(CONVEYOR_DEQUEUE_FINISH, 30424, "Conveyor" _PH1_ ": the queue has been changed, item" _PH2_ "
dequeue due to track finished", "suggest...") \
00160     _D(CONVEYOR_DEQUEUE_STARTWINDOW, 30425, "Conveyor" _PH1_ ": the queue has been changed, item"
_PH2_ " dequeue due to exceeds startwindow", "suggest...") \
00161     _D(CONVEYOR_DEQUEUE_LIMIT, 30426, "Conveyor" _PH1_ ": the queue has been changed, item" _PH2_ "
dequeue due to exceed limit area", "suggest...") \
00162     _D(CONVEYOR_DEQUEUE_CLEAR, 30427, "Conveyor" _PH1_ ": item queue is cleared", "suggest...") \
00163     _D(CONVEYOR_NEXT_TRACK, 30428, "Conveyor" _PH1_ ": item" _PH2_ " inside the start window that can be
tracked ", "suggest...") \
00164     _D(CONVEYOR_EXCEED_LIMIT, 30429, "Conveyor" _PH1_ ": item" _PH2_ " exceeds the limit area during
tracking", "suggest...") \
00165     _D(WRONG_POWER_SAFETY_LIMIT, 30430, "Robot power safety value exceeds designed value.", "suggest...") \
00166     _D(WRONG_POWER_DESIGNED_LIMIT, 30431, "Power designed value exceeds value read from hardware
interface.", "suggest...") \
00167     _D(TOOL_SENSOR_STATUS_CHANGED, 30432, "Tool sensor status changed to " _PH1_ , "suggest...") \
00168     _D(COLLISION_THRESHOLD_INVALID, 30433, "Robot collision threshold parameters are invalid.Please reidentify
the threshold or modify the configuration to ensure that it does not cause accidental collisions.", "suggest...") \
00169     _D(GRIPPER_DISCONNECT, 30434, "The gripper " _PH1_ " is disconnected.", "suggest...") \
00170     _D(GRIPPER_UNKNOWN_FAULT, 30435, "There is an unknown fault with the gripper " _PH1_ , "suggest...") \
00171     _D(GRIPPER_CURRENT_ANOMALY_FAULT, 30436, "There is an abnormal current fault with the gripper "
_PH1_ , "suggest...") \
00172     _D(GRIPPER_VOLTAGE_ANOMALY_FAULT, 30437, "There is an abnormal voltage fault with the gripper "
_PH1_ , "suggest...") \
00173     _D(GRIPPER_OVER_TEMPERATURE_FAULT, 30438, "There is an over-temperature fault with the gripper "
_PH1_ , "suggest...") \
00174     _D(GRIPPER_INTERNAL_FAULT, 30439, "There is an internal fault with the gripper " _PH1_ , "suggest...") \
00175     _D(GRIPPER_COMMUNICATION_FAULT, 30440, "There is an communication fault with the gripper " _PH1_ ,
"suggest...") \
00176     _D(GRIPPER_CONTROL_COMMAND_FAULT, 30441, "There is an control command fault with the gripper "
_PH1_ , "suggest...") \
00177     _D(GRIPPER_ENABLE_FAULT, 30442, "There is an enable fault with the gripper " _PH1_ , "suggest...")
00178
00179 // clang-format on
00180
00181 #endif // AUBO_SDK_RTM_ERROR_H

```

12.12 include/aubo/error_stack/system_error.h File Reference

This graph shows which files directly or indirectly include this file:



Macros

- `#define` [SYSTEM_ERRORS](#)

12.12.1 Detailed Description

Definition in file [system_error.h](#).

12.12.2 Macro Definition Documentation

12.12.2.1 SYSTEM_ERRORS

```
#define SYSTEM_ERRORS
```

Definition at line 8 of file [error_stack.h](#).

12.13 system_error.h

[Go to the documentation of this file.](#)

```

00001 /** @file system_error.h
00002  * @brief
00003  */
00004 #ifndef AUBO_SDK_SYSTEM_ERROR_H
00005 #define AUBO_SDK_SYSTEM_ERROR_H
00006
00007 #define SYSTEM_ERRORS
00008     _D(DEBUG, 0, "Debug message " _PH1_, "suggest...")
00009     _D(POPUP, 1, "Popup title: " _PH1_, "msg: " _PH2_, "mode: " _PH3_,
00010        "suggest...")
00011     _D(POPUP_DISMISS, 2, _PH1_, "suggest...")
00012     _D(SYSTEM_HALT, 3, _PH1_, "suggest...")
00013     _D(INV_ARGUMENTS, 4, "Invalid arguments.", "suggest...")
00014     _D(USER_NOTIFY, 5, _PH1_, "suggest...")
00015     _D(POPUP_DISMISS_BY_ID, 6, _PH1_, "suggest...")
00016     _D(MODBUS_SIGNAL_CREATED, 10, "Modbus signal " _PH1_ " created.",
  
```



```

00017     "suggest...")
00018     _D(MODBUS_SIGNAL_REMOVED, 11, "Modbus signal " _PH1_ " removed.", \
00019     "suggest...")
00020     _D(MODBUS_SIGNAL_VALUE_CHANGED, 12, \
00021     "Modbus signal " _PH1_ " value changed to " _PH2_, "suggest...") \
00022     _D(RUNTIME_CONTEXT, 13, \
00023     "tid: " _PH1_ " lineno: " _PH2_ " index: " _PH3_ " comment: " _PH4_, \
00024     "suggest...")
00025     _D(INTERP_CONTEXT, 14, \
00026     "tid: " _PH1_ " lineno: " _PH2_ " index: " _PH3_ " comment: " _PH4_, \
00027     "suggest...")
00028     _D(PROGRAM_LOADED, 15, "program loaded: " _PH1_, "suggest...") \
00029     _D(TASK_DELETED, 16, "tid: " _PH1_, " was deleted") \
00030     _D(MODBUS_SLAVE_BIT, 20, "Modbus slave address: " _PH1_ " value " _PH2_, \
00031     "suggest...")
00032     _D(MODBUS_SLAVE_REG, 21, "Modbus slave address: " _PH1_ " value " _PH2_, \
00033     "suggest...")
00034     _D(PNIO_SLAVE_SLOT_VALUE, 30, \
00035     "PNIO slot: " _PH1_ " subslot " _PH2_ " index " _PH3_ " value " _PH4_, \
00036     "suggest...")
00037     _D(PNIO_CONNECT_STATUS, 31, "PNIO connection status changed to " _PH1_, \
00038     "suggest...")
00039     _D(PNIO_DEVICE_NAME, 32, "PNIO device name changed to " _PH1_, \
00040     "suggest...")
00041     _D(PNIO_IP, 33, "PNIO ip " _PH1_ " mask " _PH2_ " gateway " _PH3_, \
00042     "suggest...")
00043     _D(ICM_SERVER_STATUS, 40, "ICM server status changed to " _PH1_, \
00044     "suggest...")
00045     _D(EIP_SLAVE_VALUE, 50, \
00046     "EIP slave: trans_type " _PH1_ " index " _PH2_ " value " _PH3_, \
00047     "suggest...")
00048     _D(EIP_SLAVE_CONNECT_STATUS, 51, \
00049     "EIP slave connection status changed to " _PH1_, "suggest...") \
00050     _D(LOG_PROGRAM_SUCCESS, 100, \
00051     "[" _PH1_ "] Load program " _PH2_ " successful", "suggest...") \
00052     _D(LOG_PROGRAM_FAILED, 101, \
00053     "[" _PH1_ "] Load program " _PH2_ " failed, file not found", \
00054     "suggest...") \
00055     _D(LOG_PROGRAM_FAILED2, 102, \
00056     "[" _PH1_ "] Load program " _PH2_ \
00057     " failed, configuration file (.ins) does not match", \
00058     "suggest...")
00059
00060 #endif // AUBO_SDK_SYSTEM_ERROR_H

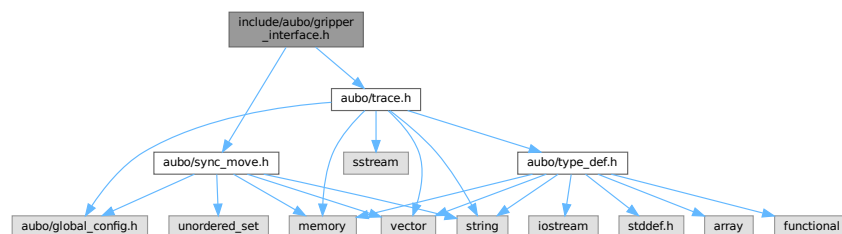
```

12.14 include/aubo/gripper_interface.h File Reference

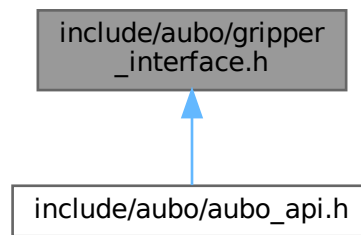
```
#include <aubo/sync_move.h>
```

```
#include <aubo/trace.h>
```

Include dependency graph for gripper_interface.h:



This graph shows which files directly or indirectly include this file:



Classes

- class [arcs::common_interface::GripperInterface](#)

Namespaces

- namespace [arcs](#)
- namespace [arcs::common_interface](#)

Typedefs

- using [arcs::common_interface::GripperInterfacePtr](#) = [std::shared_ptr<GripperInterface>](#)

12.14.1 Detailed Description

Definition in file [gripper_interface.h](#).

12.15 gripper_interface.h

[Go to the documentation of this file.](#)

```

00001 /** @file gripper_interface.h
00002  * @brief
00003  */
00004 #ifndef AUBO_SDK_GRIPPER_INTERFACE_H
00005 #define AUBO_SDK_GRIPPER_INTERFACE_H
00006
00007 #include <aubo/sync_move.h>
00008 #include <aubo/trace.h>
00009
00010 namespace arcs {
00011 namespace common_interface {
00012
00013 /**
00014  * @defgroup GripperInterface GripperInterface ( )
00015  * \-chinese API
00016  */
  
```



```

00017 class ARCS_ABI_EXPORT GripperInterface
00018 {
00019 public:
00020     GripperInterface();
00021     virtual ~GripperInterface();
00022
00023     /**
00024      * @ingroup GripperInterface
00025      * \chinese
00026      *
00027      *
00028      * @return
00029      * \endchinese
00030      */
00031     std::vector<std::string> gripperGetSupportedModels();
00032
00033     /**
00034      * @ingroup GripperInterface
00035      * \chinese
00036      *
00037      *
00038      * @param model
00039      * @param device_name
00040      * @return 0
00041      * \endchinese
00042      */
00043     ResultWithErrno2 gripperScanDevices(const std::string &model,
00044                                         const std::string &device_name);
00045
00046     /**
00047      * @ingroup GripperInterface
00048      * \chinese
00049      *
00050      * @return
00051      */
00052     std::vector<std::string> gripperGetNames();
00053
00054     /**
00055      * @ingroup GripperInterface
00056      * \chinese
00057      *
00058      *
00059      * @param name
00060      * @param model
00061      * @return 0
00062      * \endchinese
00063      */
00064     int gripperAdd(const std::string &name, const std::string &model);
00065
00066     /**
00067      * @ingroup GripperInterface
00068      * \chinese
00069      *
00070      *
00071      * @param name
00072      * @return 0
00073      * \endchinese
00074      */
00075     int gripperDelete(const std::string &name);
00076
00077     /**
00078      * @ingroup GripperInterface
00079      * \chinese
00080      *
00081      *
00082      * @param name
00083      * @param new_name
00084      * @return 0
00085      * \endchinese
00086      */
00087     int gripperRename(const std::string &name, const std::string &new_name);
00088
00089     /**
00090      * @ingroup GripperInterface
00091      * \chinese
00092      *
00093      *
00094      * @param name
00095      * @param device_name
00096      * @return 0
00097      * \endchinese
00098      */
00099     int gripperConnect(const std::string &name, const std::string &device_name);
00100
00101     /**
00102      * @ingroup GripperInterface
00103      * \chinese

```

```

00104  *
00105  *
00106  * @param name
00107  * @return 0
00108  * \endchinese
00109  */
00110  int gripperDisconnect(const std::string &name);
00111
00112  /**
00113  * @ingroup GripperInterface
00114  * \chinese
00115  *
00116  *
00117  * @param name
00118  * @return true false
00119  * \endchinese
00120  */
00121  bool gripperIsConnected(const std::string &name);
00122
00123  /**
00124  * @ingroup GripperInterface
00125  * \chinese
00126  *
00127  *
00128  * @param name
00129  * @param work_mode
00130  * @return 0
00131  * \endchinese
00132  */
00133  int gripperSetWorkMode(const std::string &name, int work_mode);
00134
00135  /**
00136  * @ingroup GripperInterface
00137  * \chinese
00138  *
00139  *
00140  * @param name
00141  * @return
00142  * \endchinese
00143  */
00144  int gripperGetWorkMode(const std::string &name);
00145
00146  /**
00147  * @ingroup GripperInterface
00148  * \chinese
00149  *
00150  *
00151  * @param name
00152  * @param isVisible
00153  * @param pose
00154  * @return true false
00155  * \endchinese
00156  */
00157  int gripperSetMountPose(const std::string &name,
00158                          const std::vector<double> &pose,
00159                          bool enable_collision);
00160
00161  /**
00162  * @brief
00163  * @param name
00164  * @return
00165  */
00166  std::vector<double> gripperGetMountPose(const std::string &name);
00167
00168  /**
00169  * @ingroup GripperInterface
00170  * \chinese
00171  *
00172  *
00173  * @param name
00174  * @return
00175  * \endchinese
00176  */
00177  int gripperEnable(const std::string &name, bool enable);
00178
00179  /**
00180  * @ingroup GripperInterface
00181  * \chinese
00182  *
00183  *
00184  * @param name
00185  * @return
00186  * \endchinese
00187  */
00188  bool gripperIsEnabled(const std::string &name);
00189
00190  /**

```

```

00191  * @ingroup GripperInterface
00192  * \chinese
00193  *
00194  *
00195  * @param name
00196  * @param position    m/Pa
00197  * @param velocity_percent    %    [0, 1]
00198  * @param force        N
00199  * @param angle        rad
00200  * @param r_velocity_percent    %    [0, 1]
00201  * @param torque_percent    %    [0, 1]
00202  *
00203  * @return
00204  * \endchinese
00205  */
00206  int gripperSetPosition(const std::string &name, const double position);
00207  int gripperSetVelocity(const std::string &name,
00208                        const double velocity_percent);
00209  int gripperSetForce(const std::string &name, const double force);
00210  int gripperSetAngle(const std::string &name, const double angle);
00211  int gripperSetRVelocity(const std::string &name,
00212                        const double r_velocity_percent);
00213  int gripperSetTorque(const std::string &name, const double torque_percent);
00214
00215  /**
00216  * @ingroup GripperInterface
00217  * \chinese
00218  *
00219  *
00220  * @param name
00221  * @return
00222  * \endchinese
00223  */
00224  int gripperMove(const std::string &name);
00225
00226  /**
00227  * @ingroup GripperInterface
00228  * \chinese
00229  *
00230  *
00231  * @param name
00232  * @return
00233  * \endchinese
00234  */
00235  int gripperStop(const std::string &name);
00236
00237  /**
00238  * @ingroup GripperInterface
00239  * \chinese
00240  *
00241  *
00242  * @param name
00243  * @return
00244  *
00245  * @note          Set
00246  *
00247  * \endchinese
00248  */
00249  std::string gripperGetHardwareVersion(const std::string &name);
00250  std::string gripperGetSoftwareVersion(const std::string &name);
00251  double gripperGetPosition(const std::string &name);
00252  double gripperGetVelocity(const std::string &name);
00253  double gripperGetForce(const std::string &name);
00254  double gripperGetAngle(const std::string &name);
00255  double gripperGetRVelocity(const std::string &name);
00256  double gripperGetTorque(const std::string &name);
00257  bool gripperGetObjectDetection(const std::string &name);
00258  bool gripperGetMotionState(const std::string &name);
00259  double gripperGetVoltage(const std::string &name);
00260  double gripperGetTemperature(const std::string &name);
00261
00262  /**
00263  * @ingroup GripperInterface
00264  * \chinese
00265  * modbus ID
00266  *
00267  * @param name
00268  * @param slave_id Id
00269  * @return
00270  * \endchinese
00271  */
00272  int gripperResetSlaveId(const std::string &name, const int slave_id);
00273
00274  /**
00275  * @ingroup GripperInterface
00276  * \chinese
00277  *

```

```

00278  * @param name
00279  * @return
00280  * \endchinese
00281  */
00282  int gripperGetStatusCode(const std::string &name);
00283
00284 protected:
00285  void *d_;
00286 };
00287 using GripperInterfacePtr = std::shared_ptr<GripperInterface>;
00288
00289 } // namespace common_interface
00290 } // namespace arcs
00291
00292 #endif // AUBO_SDK_GRIPPER_INTERFACE_H

```

12.16 include/aubo/math.h File Reference

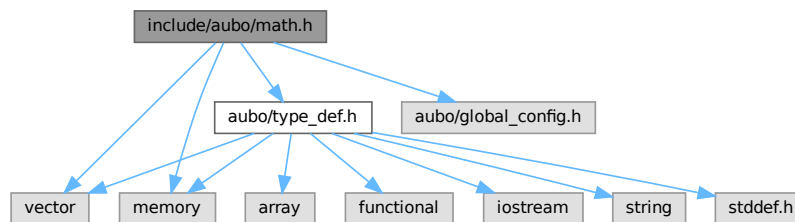
Mathematic operation interface, such as euler to quaternion conversion, addition/subtraction of poses.

```

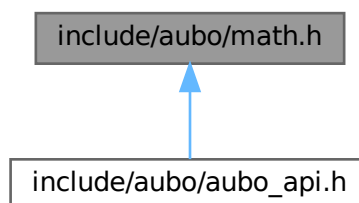
#include <vector>
#include <memory>
#include <aubo/type_def.h>
#include <aubo/global_config.h>

```

Include dependency graph for math.h:



This graph shows which files directly or indirectly include this file:



Classes

- class [arcs::common_interface::Math](#)

Namespaces

- namespace [arcs](#)
- namespace [arcs::common_interface](#)

Typedefs

- using [arcs::common_interface::MathPtr](#) = std::shared_ptr<[Math](#)>

12.16.1 Detailed Description

Mathematic operation interface, such as euler to quaternion conversion, addition/subtraction of poses.

Definition in file [math.h](#).

12.17 math.h

[Go to the documentation of this file.](#)

```

00001 /** @file math.h
00002 *  \~chinese @brief
00003 *  \~english @brief Mathematic operation interface, such as euler to quaternion conversion, addition/subtraction of poses
00004 */
00005 #ifndef AUBO_SDK_MATH_INTERFACE_H
00006 #define AUBO_SDK_MATH_INTERFACE_H
00007
00008 #include <vector>
00009 #include <memory>
00010
00011 #include <aubo/type_def.h>
00012 #include <aubo/global_config.h>
00013
00014 namespace arcs {
00015 namespace common_interface {
00016
00017 /**
00018 * @defgroup Math Math ( )
00019 * \~chinese API
00020 */
00021 class ARCS_ABI_EXPORT Math
00022 {
00023 public:
00024     Math();
00025     virtual ~Math();
00026
00027 /**
00028 * @ingroup Math
00029 * \~english
00030 * Pose addition
00031 *
00032 * Both arguments contain three position parameters (x, y, z) jointly called
00033 * P, and three rotation parameters (R_x, R_y, R_z) jointly called R. This
00034 * function calculates the result x_3 as the addition of the given poses as
00035 * follows:
00036 *
00037 * p_3.P = p_1.P + p_2.P
00038 *
00039 * p_3.R = p_1.R * p_2.R
00040 *
00041 * @param p1 Tool pose 1
00042 * @param p2 Tool pose 2
00043 * @return sum of position parts and product of rotation parts (pose)
00044 *
00045 * @par Python interface prototype
00046 * poseAdd(self: pyaubo_sdk.Math, arg0: List[float], arg1: List[float]) ->
00047 * List[float]
00048 *
00049 * @par Lua interface prototype
00050 * poseAdd(p1: table, p2: table) -> table
00051 *
00052 * @par Lua example

```

```

00053 * pose_add = poseAdd({0.2, 0.5, 0.1, 1.57, 0, 0},{0.2, 0.5, 0.6, 1.57, 0, 0})
00054 *
00055 * @par JSON-RPC request example
00056 * {"jsonrpc":"2.0","method":"Math.poseAdd","params":[[0.2, 0.5, 0.1, 1.57,
00057 * 0, 0],[0.2, 0.5, 0.6, 1.57, 0, 0]],"id":1}
00058 *
00059 * @par JSON-RPC response example
00060 * {"id":1,"jsonrpc":"2.0","result":[0.4,1.0,0.7,3.14,-0.0,0.0]}
00061 * \endenglish
00062 *
00063 * \chinese
00064 *
00065 *      x y z   P
00066 *      R_x R_y R_z   R
00067 *      p_3
00068 * p_3.P = p_1.P + p_2.P,
00069 * p_3.R = p_1.R * p_2.R
00070 *
00071 * @param p1   1 pose
00072 * @param p2   2 pose
00073 * @return      pose
00074 *
00075 * @par Python
00076 * poseAdd(self: pyaubo_sdk.Math, arg0: List[float], arg1: List[float]) ->
00077 * List[float]
00078 *
00079 * @par Lua
00080 * poseAdd(p1: table, p2: table) -> table
00081 *
00082 * @par Lua
00083 * pose_add = poseAdd({0.2, 0.5, 0.1, 1.57, 0, 0},{0.2, 0.5, 0.6, 1.57, 0, 0})
00084 *
00085 * @par JSON-RPC
00086 * {"jsonrpc":"2.0","method":"Math.poseAdd","params":[[0.2, 0.5, 0.1, 1.57,
00087 * 0, 0],[0.2, 0.5, 0.6, 1.57, 0, 0]],"id":1}
00088 *
00089 * @par JSON-RPC
00090 * {"id":1,"jsonrpc":"2.0","result":[0.4,1.0,0.7,3.14,-0.0,0.0]}
00091 * \endchinese
00092 */
00093 std::vector<double> poseAdd(const std::vector<double> &p1,
00094                             const std::vector<double> &p2);
00095
00096 /**
00097 * @ingroup Math
00098 * \chinese
00099 *
00100 *
00101 *      x y z   P
00102 *      R_x R_y R_z   R
00103 *      p_3
00104 * p_3.P = p_1.P - p_2.P,
00105 * p_3.R = p_1.R * p_2.R.inverse
00106 *
00107 * @param p1   1
00108 * @param p2   2
00109 * @return
00110 *
00111 * @par Python
00112 * poseSub(self: pyaubo_sdk.Math, arg0: List[float], arg1: List[float]) ->
00113 * List[float]
00114 *
00115 * @par Lua
00116 * poseSub(p1: table, p2: table) -> table
00117 *
00118 * @par Lua
00119 * pose_sub = poseSub({0.2, 0.5, 0.1, 1.57, 0, 0},{0.2, 0.5, 0.6, 1.57, 0, 0})
00120 *
00121 * @par JSON-RPC
00122 * {"jsonrpc":"2.0","method":"Math.poseSub","params":[[0.2, 0.5, 0.1, 1.57,
00123 * 0, 0],[0.2, 0.5, 0.6, 1.57, 0, 0]],"id":1}
00124 *
00125 * @par JSON-RPC
00126 * {"id":1,"jsonrpc":"2.0","result":[0.0,0.0,-0.5,0.0,-0.0,0.0]}
00127 * \endchinese
00128 *
00129 * \english
00130 * Pose subtraction
00131 *
00132 * Both arguments contain three position parameters (x, y, z) jointly called
00133 * P, and three rotation parameters (R_x, R_y, R_z) jointly called R. This
00134 * function calculates the result x_3 as the addition of the given poses as
00135 * follows:
00136 *
00137 * p_3.P = p_1.P - p_2.P,
00138 *
00139 * p_3.R = p_1.R * p_2.R.inverse

```

```

00140 *
00141 * @param p1 tool pose 1
00142 * @param p2 tool pose 2
00143 * @return difference between two poses
00144 *
00145 * @par Python interface prototype
00146 * poseSub(self: pyaubo_sdk.Math, arg0: List[float], arg1: List[float]) ->
00147 * List[float]
00148 *
00149 * @par Lua interface prototype
00150 * poseSub(p1: table, p2: table) -> table
00151 *
00152 * @par Lua example
00153 * pose_sub = poseSub({0.2, 0.5, 0.1, 1.57, 0, 0},{0.2, 0.5, 0.6, 1.57, 0, 0})
00154 *
00155 * @par JSON-RPC request example
00156 * {"jsonrpc":"2.0","method":"Math.poseSub","params":[[0.2, 0.5, 0.1, 1.57,
00157 * 0, 0],[0.2, 0.5, 0.6, 1.57, 0, 0]],"id":1}
00158 *
00159 * @par JSON-RPC response example
00160 * {"id":1,"jsonrpc":"2.0","result":[0.0,0.0,-0.5,0.0,-0.0,0.0]}
00161 * \endenglish
00162 */
00163 std::vector<double> poseSub(const std::vector<double> &p1,
00164                             const std::vector<double> &p2);
00165
00166 /**
00167 * @ingroup Math
00168 * \chinese
00169 *
00170 *
00171 * @param p1 TCP
00172 * @param p2 TCP
00173 * @param alpha
00174 * 0<alpha<1 p1 p2 p1 alpha
00175 * alpha=0.3, p1 30
00176 * alpha>1, p2
00177 * alpha<0, p1
00178 * @return
00179 *
00180 * @par Python
00181 * interpolatePose(self: pyaubo_sdk.Math, arg0: List[float], arg1:
00182 * List[float], arg2: float) -> List[float]
00183 *
00184 * @par Lua
00185 * interpolatePose(p1: table, p2: table, alpha: number) -> table
00186 *
00187 * @par Lua
00188 * pose_interpolate = interpolatePose({0.2, 0.2, 0.4, 0, 0, 0},{0.2, 0.2, 0.6, 0, 0, 0},0.5)
00189 *
00190 * @par JSON-RPC
00191 * {"jsonrpc":"2.0","method":"Math.interpolatePose","params":[[0.2, 0.2,
00192 * 0.4, 0, 0, 0],[0.2, 0.2, 0.6, 0, 0, 0],0.5],"id":1}
00193 *
00194 * @par JSON-RPC
00195 * {"id":1,"jsonrpc":"2.0","result":[0.2,0.2,0.5,0.0,-0.0,0.0]}
00196 * \endchinese
00197 *
00198 * \english
00199 * Calculate linear interpolation
00200 *
00201 * @param p1 starting TCP pose
00202 * @param p2 ending TCP pose
00203 * @param alpha coefficient;
00204 * When 0<alpha<1 return a point between p1 & p2 that is closer to p1, at alpha percentage of the path
00205 * For example when alpha=0.3,point returned is closer to p1 at 30% of the total distance;
00206 * When alpha>1, return p2
00207 * When alpha<0,return p1
00208 * @return interpolation result
00209 *
00210 * @par Python interface prototype
00211 * interpolatePose(self: pyaubo_sdk.Math, arg0: List[float], arg1:
00212 * List[float], arg2: float) -> List[float]
00213 *
00214 * @par Lua interface prototype
00215 * interpolatePose(p1: table, p2: table, alpha: number) -> table
00216 *
00217 * @par Lua example
00218 * pose_interpolate = interpolatePose({0.2, 0.2, 0.4, 0, 0, 0},{0.2, 0.2, 0.6, 0, 0, 0},0.5)
00219 *
00220 * @par JSON-RPC request example
00221 * {"jsonrpc":"2.0","method":"Math.interpolatePose","params":[[0.2, 0.2,
00222 * 0.4, 0, 0, 0],[0.2, 0.2, 0.6, 0, 0, 0],0.5],"id":1}
00223 *
00224 * @par JSON-RPC response example
00225 * {"id":1,"jsonrpc":"2.0","result":[0.2,0.2,0.5,0.0,-0.0,0.0]}
00226 * \endenglish

```

```

00227 *
00228 */
00229 std::vector<double> interpolatePose(const std::vector<double> &p1,
00230                                   const std::vector<double> &p2,
00231                                   double alpha);
00232 /**
00233  * @ingroup Math
00234  * \chinese
00235  *
00236  *
00237  *      p_from      p_from_to
00238  *      p_from
00239  *      p_from_to
00240  *
00241  *
00242  *      p_from_to  p_from
00243  *      p_from      p_from_to
00244  *
00245  *
00246  * T_world->to = T_world->from * T_from->to
00247  * T_x->to = T_x->from * T_from->to
00248  *
00249  *
00250  *
00251  *      B A C B C A
00252  *      B A C B
00253  *      C A
00254  *
00255  * @param pose_from
00256  * @param pose_from_to
00257  * @return ( )
00258  *
00259  * @par Python
00260  * poseTrans(self: pyaubo_sdk.Math, arg0: List[float], arg1: List[float]) ->
00261  * List[float]
00262  *
00263  * @par Lua
00264  * poseTrans(pose_from: table, pose_from_to: table) -> table
00265  *
00266  * @par Lua
00267  * pose_trans = poseTrans({0.2, 0.5, 0.1, 1.57, 0, 0},{0.2, 0.5, 0.6, 1.57, 0, 0})
00268  *
00269  * @par JSON-RPC
00270  * {"jsonrpc":"2.0","method":"Math.poseTrans","params":[[0.2, 0.5,
00271  * 0.1, 1.57, 0, 0],[0.2, 0.5, 0.6, 1.57, 0, 0]], "id":1}
00272  *
00273  * @par JSON-RPC
00274  * {"id":1,"jsonrpc":"2.0","result":[0.4,-0.09960164640373415,0.6004776374923573,3.14,-0.0,0.0]}
00275  * \endchinese
00276  *
00277  * \english
00278  * Pose transformation
00279  *
00280  * The first argument, p_from, is used to transform the second argument,
00281  * p_from_to, and the result is then returned. This means that the result is
00282  * the resulting pose, when starting at the coordinate system of p_from, and
00283  * then in that coordinate system moving p_from_to.
00284  *
00285  * This function can be seen in two different views. Either the function
00286  * transforms, that is translates and rotates, p_from_to by the parameters
00287  * of p_from. Or the function is used to get the resulting pose, when first
00288  * making a move of p_from and then from there, a move of p_from_to. If the
00289  * poses were regarded as transformation matrices, it would look like:
00290  *
00291  * T_world->to = T_world->from * T_from->to
00292  * T_x->to = T_x->from * T_from->to
00293  *
00294  *
00295  * These two equations describes the foundations for pose transformation.
00296  * Based on a starting pose and the pose transformation relative to the starting pose, we can get the target pose.
00297  *
00298  * For example, we know pose of B relative to A, pose of C relative to B, find pose of C relative to A.
00299  * param 1 is pose of B relative to A param 2 is pose of C relative to B
00300  * the return value is the pose of C relative to A
00301  *
00302  * @param pose_from starting pose vector in 3D space
00303  * @param pose_from_to pose transformation relative to starting pose vector in 3D space
00304  * @return final pose (vector in 3D space)
00305  *
00306  * @par Python interface prototype
00307  * poseTrans(self: pyaubo_sdk.Math, arg0: List[float], arg1: List[float]) ->
00308  * List[float]
00309  *
00310  * @par Lua interface prototype
00311  * poseTrans(pose_from: table, pose_from_to: table) -> table
00312  *
00313  * @par Lua example

```



```

00314 * pose_trans = poseTrans({0.2, 0.5, 0.1, 1.57, 0, 0},{0.2, 0.5, 0.6, 1.57, 0, 0})
00315 *
00316 * @par JSON-RPC request example
00317 * {"jsonrpc":"2.0","method":"Math.poseTrans","params":[[0.2, 0.5,
00318 * 0.1, 1.57, 0, 0],[0.2, 0.5, 0.6, 1.57, 0, 0]],"id":1}
00319 *
00320 * @par JSON-RPC response example
00321 * {"id":1,"jsonrpc":"2.0","result":[0.4,-0.09960164640373415,0.6004776374923573,3.14,-0.0,0.0]}
00322 * \endenglish
00323 */
00324 std::vector<double> poseTrans(const std::vector<double> &pose_from,
00325                             const std::vector<double> &pose_to);
00326
00327 /**
00328 * \english
00329 * Pose inverse transformation
00330 *
00331 * Given pose of C relative to A, pose of C relative to B, find pose of B relative to A.
00332 * param 1 is pose of C relative to A param 2 is pose of C relative to B
00333 * the return value is the pose of B relative to A
00334 *
00335 * @param pose_from starting pose
00336 * @param pose_to_from pose transformation relative to final pose
00337 * @return resulting pose
00338 *
00339 * @par Python interface prototype
00340 * poseTransInv(self: pyaubo_sdk.Math, arg0: List[float], arg1: List[float])
00341 * -> List[float]
00342 *
00343 * @par Lua interface prototype
00344 * poseTransInv(pose_from: table, pose_to_from: table) -> table
00345 *
00346 * @par Lua example
00347 * pose_trans_inv = poseTransInv({0.4, -0.0996016, 0.600478, 3.14, 0, 0},{0.2, 0.5, 0.6, 1.57, 0, 0})
00348 *
00349 * @par JSON-RPC request example
00350 * {"jsonrpc":"2.0","method":"Math.poseTransInv","params":[[0.4, -0.0996016,
00351 * 0.600478, 3.14, 0, 0],[0.2, 0.5, 0.6, 1.57, 0, 0]],"id":1}
00352 *
00353 * @par JSON-RPC response example
00354 * {"id":1,"jsonrpc":"2.0","result":[0.2,0.5000000464037341,0.10000036250764266,1.57,-0.0,0.0]}
00355 * \endenglish
00356 *
00357 * @ingroup Math
00358 * \chinese
00359 *
00360 *
00361 * C A C B B A
00362 * C A C B
00363 * B A
00364 *
00365 * @param pose_from
00366 * @param pose_to_from
00367 * @return
00368 *
00369 * @par Python
00370 * poseTransInv(self: pyaubo_sdk.Math, arg0: List[float], arg1: List[float])
00371 * -> List[float]
00372 *
00373 * @par Lua
00374 * poseTransInv(pose_from: table, pose_to_from: table) -> table
00375 *
00376 * @par Lua
00377 * pose_trans_inv = poseTransInv({0.4, -0.0996016, 0.600478, 3.14, 0, 0},{0.2, 0.5, 0.6, 1.57, 0, 0})
00378 *
00379 * @par JSON-RPC
00380 * {"jsonrpc":"2.0","method":"Math.poseTransInv","params":[[0.4, -0.0996016,
00381 * 0.600478, 3.14, 0, 0],[0.2, 0.5, 0.6, 1.57, 0, 0]],"id":1}
00382 *
00383 * @par JSON-RPC
00384 * {"id":1,"jsonrpc":"2.0","result":[0.2,0.5000000464037341,0.10000036250764266,1.57,-0.0,0.0]}
00385 * \endchinese
00386 */
00387 std::vector<double> poseTransInv(const std::vector<double> &pose_from,
00388                                const std::vector<double> &pose_to_from);
00389
00390 /**
00391 * @ingroup Math
00392 * \chinese
00393 *
00394 *
00395 * @param pose
00396 * @return
00397 *
00398 * @par Python
00399 * poseInverse(self: pyaubo_sdk.Math, arg0: List[float]) -> List[float]
00400 *

```

```

00401 * @par Lua
00402 * poseInverse(pose: table) -> table
00403 *
00404 * @par Lua
00405 * pose_inverse = poseInverse({0.2, 0.5, 0.1, 1.57, 0, 3.14})
00406 *
00407 * @par JSON-RPC
00408 * {"jsonrpc": "2.0", "method": "Math.poseInverse", "params": [[0.2, 0.5,
00409 * 0.1, 1.57, 0, 3.14]], "id": 1}
00410 *
00411 * @par JSON-RPC
00412 * {"id": 1, "jsonrpc": "2.0", "result": [0.19920341988726448, -0.09960155178838484, -0.5003973704832628,
00413 * 1.569999989900404, -0.0015926530848129354, -3.1415913853161266]}
00414 *
00415 * \endchinese
00416 *
00417 * \english
00418 * Get the inverse of a pose
00419 *
00420 * @param pose tool pose (spatial vector)
00421 * @return inverse tool pose transformation (spatial vector)
00422 *
00423 * @par Python interface prototype
00424 * poseInverse(self: pyaubo_sdk.Math, arg0: List[float]) -> List[float]
00425 *
00426 * @par Lua interface prototype
00427 * poseInverse(pose: table) -> table
00428 *
00429 * @par Lua example
00430 * pose_inverse = poseInverse({0.2, 0.5, 0.1, 1.57, 0, 3.14})
00431 *
00432 * @par JSON-RPC request example
00433 * {"jsonrpc": "2.0", "method": "Math.poseInverse", "params": [[0.2, 0.5,
00434 * 0.1, 1.57, 0, 3.14]], "id": 1}
00435 *
00436 * @par JSON-RPC response example
00437 * {"id": 1, "jsonrpc": "2.0", "result": [0.19920341988726448, -0.09960155178838484, -0.5003973704832628,
00438 * 1.569999989900404, -0.0015926530848129354, -3.1415913853161266]}
00439 * \endenglish
00440 *
00441 */
00442 std::vector<double> poseInverse(const std::vector<double> &pose);
00443
00444 /**
00445 * @ingroup Math
00446 * \chinese
00447 *
00448 *
00449 * @param p1 1
00450 * @param p2 2
00451 * @return
00452 *
00453 * @par Lua
00454 * poseDistance(p1: table, p2: table) -> number
00455 *
00456 * @par Lua
00457 * pose_distance = poseDistance({0.1, 0.3, 0.1, 0.3142, 0.0, 1.571}, {0.2, 0.5, 0.6, 0, -0.172, 0.0})
00458 *
00459 * @par JSON-RPC
00460 * {"jsonrpc": "2.0", "method": "Math.poseDistance", "params": [[0.1, 0.3, 0.1,
00461 * 0.3142, 0.0, 1.571], [0.2, 0.5, 0.6, 0, -0.172, 0.0]], "id": 1}
00462 *
00463 * @par JSON-RPC
00464 * {"id": 1, "jsonrpc": "2.0", "result": 0.5477225575051661}
00465 * \endchinese
00466 *
00467 * \english
00468 * Calculate distance between two poses
00469 *
00470 * @param p1 pose 1
00471 * @param p2 pose 2
00472 * @return distance between the poses
00473 *
00474 * @par Lua function prototype
00475 * poseDistance(p1: table, p2: table) -> number
00476 *
00477 * @par Lua example
00478 * pose_distance = poseDistance({0.1, 0.3, 0.1, 0.3142, 0.0, 1.571}, {0.2, 0.5, 0.6, 0, -0.172, 0.0})
00479 *
00480 * @par JSON-RPC request example
00481 * {"jsonrpc": "2.0", "method": "Math.poseDistance", "params": [[0.1, 0.3, 0.1,
00482 * 0.3142, 0.0, 1.571], [0.2, 0.5, 0.6, 0, -0.172, 0.0]], "id": 1}
00483 *
00484 * @par JSON-RPC response example
00485 * {"id": 1, "jsonrpc": "2.0", "result": 0.5477225575051661}
00486 * \endenglish
00487 *

```

```

00488     */
00489     double poseDistance(const std::vector<double> &p1,
00490                        const std::vector<double> &p2);
00491
00492     /**
00493     * @ingroup Math
00494     * \chinese
00495     *
00496     * @param p1 1
00497     * @param p2 2
00498     * @return
00499     *
00500     * @par Lua
00501     * poseAngleDistance(p1: table, p2: table) -> number
00502     *
00503     * @par Lua
00504     * pose_angle_distance = poseAngleDistance({0.1, 0.3, 0.1, 0.3142, 0.0, 1.571},{0.2, 0.5, 0.6, 0, -0.172, 0.0})
00505     *
00506     * \endchinese
00507     *
00508     * \english
00509     * Calculate axis-angle difference between two poses
00510     *
00511     * @param p1 pose 1
00512     * @param p2 pose 2
00513     * @return axis angle difference
00514     *
00515     * @par Lua
00516     * poseAngleDistance(p1: table, p2: table) -> number
00517     *
00518     * @par Lua
00519     * pose_angle_distance = poseAngleDistance({0.1, 0.3, 0.1, 0.3142, 0.0, 1.571},{0.2, 0.5, 0.6, 0, -0.172, 0.0})
00520     *
00521     * \endenglish
00522     */
00523     double poseAngleDistance(const std::vector<double> &p1,
00524                             const std::vector<double> &p2);
00525
00526     /**
00527     * @ingroup Math
00528     * \chinese
00529     *
00530     * @param p1 1
00531     * @param p2 2
00532     * @param eps
00533     * @return true false
00534     *
00535     * @par Lua
00536     * poseEqual(p1: table, p2: table, eps: number) -> boolean
00537     *
00538     * @par Lua
00539     * pose_equal = poseEqual({0.1, 0.3, 0.1, 0.3142, 0.0, 1.571},{0.1, 0.3, 0.1, 0.3142, 0.0, 1.5711},0.01)
00540     *
00541     * @par JSON-RPC
00542     * {"jsonrpc":"2.0","method":"Math.poseEqual","params":[[0.1, 0.3, 0.1,
00543     * 0.3142, 0.0, 1.571],[0.1, 0.3, 0.1, 0.3142, 0.0, 1.5711],0.01]], "id":1}
00544     *
00545     * @par JSON-RPC
00546     * {"id":1,"jsonrpc":"2.0","result":0.0}
00547     * \endchinese
00548     *
00549     * \english
00550     * Determine if two poses are equivalent
00551     *
00552     * @param p1 pose 1
00553     * @param p2 pose 2
00554     * @param eps error margin
00555     * @return true or false
00556     *
00557     * @par Lua
00558     * poseEqual(p1: table, p2: table, eps: number) -> boolean
00559     *
00560     * @par Lua
00561     * pose_equal = poseEqual({0.1, 0.3, 0.1, 0.3142, 0.0, 1.571},{0.1, 0.3, 0.1, 0.3142, 0.0, 1.5711},0.01)
00562     *
00563     * @par JSON-RPC request example
00564     * {"jsonrpc":"2.0","method":"Math.poseEqual","params":[[0.1, 0.3, 0.1,
00565     * 0.3142, 0.0, 1.571],[0.1, 0.3, 0.1, 0.3142, 0.0, 1.5711],0.01]], "id":1}
00566     *
00567     * @par JSON-RPC response example
00568     * {"id":1,"jsonrpc":"2.0","result":0.0}
00569     * \endenglish
00570     */
00571     bool poseEqual(const std::vector<double> &p1, const std::vector<double> &p2,
00572                   const std::vector<double> &p2);

```

```

00575         double eps = 5e-5);
00576
00577 /**
00578  * @ingroup Math
00579  * \chinese
00580  * @param F_b_a_old
00581  * @param V_in_a
00582  * @param type
00583  * @return
00584  *
00585  * @par Python
00586  * transferRefFrame(self: pyaubo_sdk.Math, arg0: List[float], arg1:
00587  * List[float[3]], arg2: int) -> List[float]
00588  *
00589  * @par Lua
00590  * transferRefFrame(F_b_a_old: table, V_in_a: table, type: number) -> table
00591  * \endchinese
00592  *
00593  * \english
00594  * @param F_b_a_old
00595  * @param V_in_a
00596  * @param type
00597  * @return
00598  *
00599  * @par Python interface prototype
00600  * transferRefFrame(self: pyaubo_sdk.Math, arg0: List[float], arg1:
00601  * List[float[3]], arg2: int) -> List[float]
00602  *
00603  * @par Lua interface prototype
00604  * transferRefFrame(F_b_a_old: table, V_in_a: table, type: number) -> table
00605  * \endenglish
00606  */
00607 std::vector<double> transferRefFrame(const std::vector<double> &F_b_a_old,
00608                                     const Vector3d &V_in_a, int type);
00609
00610 /**
00611  * @ingroup Math
00612  * \chinese
00613  *
00614  *
00615  * @param pose
00616  * @param rotv
00617  * @return
00618  *
00619  * @par Python
00620  * poseRotation(self: pyaubo_sdk.Math, arg0: List[float], arg1: List[float])
00621  * -> List[float]
00622  *
00623  * @par Lua
00624  * poseRotation(pose: table, rotv: table) -> table
00625  * \endchinese
00626  *
00627  * \english
00628  * Pose rotation
00629  *
00630  * @param pose
00631  * @param rotv
00632  * @return
00633  *
00634  * @par Python interface prototype
00635  * poseRotation(self: pyaubo_sdk.Math, arg0: List[float], arg1: List[float])
00636  * -> List[float]
00637  *
00638  * @par Lua interface prototype
00639  * poseRotation(pose: table, rotv: table) -> table
00640  * \endenglish
00641  */
00642 std::vector<double> poseRotation(const std::vector<double> &pose,
00643                                 const std::vector<double> &rotv);
00644
00645 /**
00646  * @ingroup Math
00647  * \chinese
00648  *
00649  *
00650  * @param rpy
00651  * @return
00652  *
00653  * @par Python
00654  * rpyToQuaternion(self: pyaubo_sdk.Math, arg0: List[float]) -> List[float]
00655  *
00656  * @par Lua
00657  * rpyToQuaternion(rpy: table) -> table
00658  *
00659  * @par Lua
00660  * quaternion = rpyToQuaternion({0.611, 0.785, 0.960})
00661  *

```

```

00662 * @par JSON-RPC
00663 * {"jsonrpc":"2.0","method":"Math.rpyToQuaternion","params":[[0.611, 0.785,
00664 * 0.960]],"id":1}
00665 *
00666 * @par JSON-RPC
00667 * {"id":1,"jsonrpc":"2.0","result":[0.834721517970497,0.07804256900772265,0.4518931575790371,0.3048637712043723]}
00668 * \endchinese
00669 *
00670 * \english
00671 * Euler angles to quaternions
00672 *
00673 * @param rpy euler angles
00674 * @return quaternions
00675 *
00676 * @par Python interface prototype
00677 * rpyToQuaternion(self: pyaubo_sdk.Math, arg0: List[float]) -> List[float]
00678 *
00679 * @par Lua interface prototype
00680 * rpyToQuaternion(rpy: table) -> table
00681 *
00682 * @par Lua example
00683 * quaternion = rpyToQuaternion({0.611, 0.785, 0.960})
00684 *
00685 * @par JSON-RPC request example
00686 * {"jsonrpc":"2.0","method":"Math.rpyToQuaternion","params":[[0.611, 0.785,
00687 * 0.960]],"id":1}
00688 *
00689 * @par JSON-RPC response example
00690 * {"id":1,"jsonrpc":"2.0","result":[0.834721517970497,0.07804256900772265,0.4518931575790371,0.3048637712043723]}
00691 * \endenglish
00692 */
00693 std::vector<double> rpyToQuaternion(const std::vector<double> &rpy);
00694
00695 /**
00696 * @ingroup Math
00697 * \chinese
00698 *
00699 *
00700 * @param quat
00701 * @return
00702 *
00703 * @par Python
00704 * quaternionToRpy(self: pyaubo_sdk.Math, arg0: List[float]) -> List[float]
00705 *
00706 * @par Lua
00707 * quaternionToRpy(quat: table) -> table
00708 *
00709 * @par Lua
00710 * rpy = quaternionToRpy({0.834722,0.0780426, 0.451893, 0.304864})
00711 *
00712 * @par JSON-RPC
00713 * {"jsonrpc":"2.0","method":"Math.quaternionToRpy","params":[[0.834722,
00714 * 0.0780426, 0.451893, 0.304864]],"id":1}
00715 *
00716 * @par JSON-RPC
00717 * {"id":1,"jsonrpc":"2.0","result":[0.6110000520523781,0.7849996877683915,0.960000543982093]}
00718 * \endchinese
00719 *
00720 * \english
00721 * Quaternions to euler angles
00722 *
00723 * @param quat quaternions
00724 * @return euler angles
00725 *
00726 * @par Python interface prototype
00727 * quaternionToRpy(self: pyaubo_sdk.Math, arg0: List[float]) -> List[float]
00728 *
00729 * @par Lua interface prototype
00730 * quaternionToRpy(quat: table) -> table
00731 *
00732 * @par Lua example
00733 * rpy = quaternionToRpy({0.834722,0.0780426, 0.451893, 0.304864})
00734 *
00735 * @par JSON-RPC request example
00736 * {"jsonrpc":"2.0","method":"Math.quaternionToRpy","params":[[0.834722,
00737 * 0.0780426, 0.451893, 0.304864]],"id":1}
00738 *
00739 * @par JSON-RPC response example
00740 * {"id":1,"jsonrpc":"2.0","result":[0.6110000520523781,0.7849996877683915,0.960000543982093]}
00741 * \endenglish
00742 */
00743 std::vector<double> quaternionToRpy(const std::vector<double> &quat);
00744
00745 /**
00746 * @ingroup Math
00747 * \chinese
00748 * TCP

```

```

00749 *
00750 *
00751 *
00752 *
00753 * @param poses
00754 * @return TCP
00755 *
00756 * @par Python
00757 * tcpOffsetIdentify(self: pyaubo_sdk.Math, arg0: List[List[float]]) ->
00758 * Tuple[List[float], int]
00759 *
00760 * @par Lua
00761 * tcpOffsetIdentify(poses: table) -> table
00762 *
00763 * @par Lua
00764 * p1 = {0.48659,0.10456,0.24824,-3.135,0.004,1.569} \n
00765 * p2 = {0.38610,0.09096,0.22432,2.552,-0.437,1.008} \n
00766 * p3 = {0.38610,0.05349,0.16791,-3.021,-0.981,0.517} \n
00767 * p4 = {0.49675,0.06417,0.21408,-2.438,0.458,0.651} \n
00768 * p = {p1,p2,p3,p4} \n
00769 * tcp_result = tcpOffsetIdentify(p)
00770 *
00771 * \endchinese
00772 *
00773 * \english
00774 * Four point method calibration for TCP offset
00775 *
00776 * About a sharp point, move the robot's tcp in four different poses. Difference between each pose should be drastic.
00777 * Result can be obtained based on these four poses
00778 *
00779 * @param poses combination of four different poses
00780 * @return TCP calibration result and whether successful
00781 *
00782 * @par Python interface prototype
00783 * tcpOffsetIdentify(self: pyaubo_sdk.Math, arg0: List[List[float]]) ->
00784 * Tuple[List[float], int]
00785 *
00786 * @par Lua interface prototype
00787 * tcpOffsetIdentify(poses: table) -> table
00788 *
00789 * @par Lua example
00790 * p1 = {0.48659,0.10456,0.24824,-3.135,0.004,1.569} \n
00791 * p2 = {0.38610,0.09096,0.22432,2.552,-0.437,1.008} \n
00792 * p3 = {0.38610,0.05349,0.16791,-3.021,-0.981,0.517} \n
00793 * p4 = {0.49675,0.06417,0.21408,-2.438,0.458,0.651} \n
00794 * p = {p1,p2,p3,p4} \n
00795 * tcp_result = tcpOffsetIdentify(p)
00796 *
00797 * \endenglish
00798 */
00799 ResultWithErrno tcpOffsetIdentify(
00800     const std::vector<std::vector<double>> &poses);
00801
00802 /**
00803 * @ingroup Math
00804 * \chinese
00805 *
00806 *
00807 * @param poses
00808 * @param type : \n
00809 * 0 - oxy x xy y \n
00810 * 1 - oxz x xz z \n
00811 * 2 - oyz y yz z \n
00812 * 3 - oyx y yx x \n
00813 * 4 - ozx z zx x \n
00814 * 5 - ozy z zy y \n
00815 * @return
00816 *
00817 * @par Lua
00818 * calibrateCoordinate(poses: table, type: int) -> table, number
00819 *
00820 * @par Lua
00821 * p1 = {0.55462,0.06219,0.37175,-3.142,0.0,1.580} \n
00822 * p2 = {0.63746,0.11805,0.37175,-3.142,0.0,1.580} \n
00823 * p3 = {0.40441,0.28489,0.37174,-3.142,0.0,1.580} \n
00824 * p = {p1,p2,p3} \n
00825 * coord_pose, cal_result = calibrateCoordinate(p,0)
00826 *
00827 * @par JSON-RPC
00828 * {"jsonrpc":"2.0","method":"Math.calibrateCoordinate","params":[[[0.55462,0.06219,0.37175,-3.142,0.0,1.580],
00829 * [0.63746,0.11805,0.37175,-3.142,0.0,1.580],[0.40441,0.28489,0.37174,-3.142,0.0,1.580]],0],"id":1}
00830 *
00831 * @par JSON-RPC
00832 * {"id":1,"jsonrpc":"2.0","result":[[0.55462,0.06219,0.37175,-3.722688983883945e-05,-1.6940658945086007e-
00833 21,0.5932768162455785],0]}
00834 * \endchinese

```

```

00835 * \english
00836 * Calibrate coordinate system with 3 points
00837 *
00838 * @param poses set of 3 poses
00839 * @param type type:\n
00840 *     0 - oxy origin, +x axis, xy plane (+y direction) \n
00841 *     1 - oxz origin, +x axis, xz plane (+z direction) \n
00842 *     2 - oyz origin, +y axis, yz plane (+z direction) \n
00843 *     3 - oyx origin, +y axis, yx plane (+x direction) \n
00844 *     4 - ozx origin, +z axis, zx plane (+x direction) \n
00845 *     5 - ozy origin, +z axis, zy plane (+y direction) \n
00846 * @return Coordinate system calibration result and whether the calibration result is valid
00847 *
00848 * @par Lua function prototype
00849 * calibrateCoordinate(poses: table, type: int) -> table, number
00850 *
00851 * @par Lua example
00852 * p1 = {0.55462,0.06219,0.37175,-3.142,0.0,1.580} \n
00853 * p2 = {0.63746,0.11805,0.37175,-3.142,0.0,1.580} \n
00854 * p3 = {0.40441,0.28489,0.37174,-3.142,0.0,1.580} \n
00855 * p = {p1,p2,p3} \n
00856 * coord_pose, cal_result = calibrateCoordinate(p,0)
00857 *
00858 * @par JSON-RPC request example
00859 * {"jsonrpc":"2.0","method":"Math.calibrateCoordinate","params":[[[0.55462,0.06219,0.37175,-3.142,0.0,1.580],
00860 * [0.63746,0.11805,0.37175,-3.142,0.0,1.580],[0.40441,0.28489,0.37174,-3.142,0.0,1.580]],0],"id":1}
00861 *
00862 * @par JSON-RPC response example
00863 * {"id":1,"jsonrpc":"2.0","result":[[0.55462,0.06219,0.37175,-3.722688983883945e-05,-1.6940658945086007e-
21,0.5932768162455785],0]}
00864 * \endenglish
00865 */
00866 ResultWithErrno calibrateCoordinate(
00867     const std::vector<std::vector<double>> &poses, int type);
00868
00869 /**
00870 * @ingroup Math
00871 * \chinese
00872 *
00873 *
00874 * @param p1
00875 * @param p2
00876 * @param p3
00877 * @param mode mode 1
00878 * mode 0
00879 *
00880 * @return
00881 *
00882 * @par Lua
00883 * calculateCircleFourthPoint(p1: table, p2: table, p3: table, mode: int)
00884 *
00885 * @par Lua
00886 * center_pose, cal_result = calculateCircleFourthPoint({0.5488696249770836,-
0.1214996547187204,0.2631931199112321,-3.14159198038469,-3.673205103150083e-06,1.570796326792424},
00887 * {0.5488696249770835,-0.1214996547187207,0.3599720701808493,-3.14159198038469,-3.6732051029273e-
06,1.570796326792423},
00888 * {0.5488696249770836,-0.0389996547187214,0.3599720701808496,-3.141591980384691,-3.673205102557476e-
06,1.570796326792422},1)
00889 *
00890 * @par JSON-RPC
00891 * {"jsonrpc":"2.0","method":"Math.calculateCircleFourthPoint","params":[[0.5488696249770836,-
0.1214996547187204,0.2631931199112321,-3.14159198038469,-3.673205103150083e-06,1.570796326792424],
00892 * [0.5488696249770835,-0.1214996547187207,0.3599720701808493,-3.14159198038469,-3.6732051029273e-
06,1.570796326792423],
00893 * [0.5488696249770836,-0.0389996547187214,0.3599720701808496,-3.141591980384691,-3.673205102557476e-06,
00894 * 1.570796326792422],1],"id":1}
00895 *
00896 * @par JSON-RPC
00897 * {"id":1,"jsonrpc":"2.0","result":[[0.5488696249770837,-0.031860179583911546,0.27033259504604207,-
3.1415919803846903,-3.67320510285378e-06,1.570796326792423],1]}
00898 *
00899 * \endchinese
00900 *
00901 * \english
00902 * Based on three points on an arc, calculate the position of the midpoint of the other half of the fitted circle's arc
00903 *
00904 * @param p1 start point of the arc
00905 * @param p2 middle point of the arc
00906 * @param p3 end point of the arc
00907 * @param mode when mode = 1, need to plan for orientation around arc;
00908 * when mode = 0, do not need to plan for orientation around arc.
00909 * @return position of the midpoint of the other half of the fitted circle's arc and whether the result is valid.
00910 *
00911 *
00912 * @par Lua
00913 * calculateCircleFourthPoint(p1: table, p2: table, p3: table, mode: int)
00914 *

```

```

00915 * @par Lua
00916 * center_pose, cal_result = calculateCircleFourthPoint({0.5488696249770836,-
0.1214996547187204,0.2631931199112321,-3.14159198038469,-3.673205103150083e-06,1.570796326792424},
00917 * {0.5488696249770835,-0.1214996547187207,0.3599720701808493,-3.14159198038469,-3.6732051029273e-
06,1.570796326792423},
00918 * {0.5488696249770836,-0.0389996547187214,0.3599720701808496,-3.141591980384691,-3.673205102557476e-
06,1.570796326792422},1)
00919 *
00920 * @par JSON-RPC request example
00921 * {"jsonrpc":"2.0","method":"Math.calculateCircleFourthPoint","params":[[0.5488696249770836,-
0.1214996547187204,0.2631931199112321,-3.14159198038469,-3.673205103150083e-06,1.570796326792424],
00922 * [0.5488696249770835,-0.1214996547187207,0.3599720701808493,-3.14159198038469,-3.6732051029273e-
06,1.570796326792423],
00923 * [0.5488696249770836,-0.0389996547187214,0.3599720701808496,-3.141591980384691,-3.673205102557476e-06,
00924 * 1.570796326792422],1],"id":1}
00925 *
00926 * @par JSON-RPC response example
00927 * {"id":1,"jsonrpc":"2.0","result":[[0.5488696249770837,-0.031860179583911546,0.27033259504604207,-
3.1415919803846903,-3.67320510285378e-06,1.570796326792423],1]}
00928 *
00929 * \endenglish
00930 */
00931 ResultWithErrno calculateCircleFourthPoint(const std::vector<double> &p1,
00932                                           const std::vector<double> &p2,
00933                                           const std::vector<double> &p3,
00934                                           int mode);
00935 /**
00936 * @ingroup Math
00937 * \chinese
00938 * @brief forceTrans:
00939 *     force_in_b = pose_a_in_b * force_in_a
00940 * @param pose_a_in_b: a    b
00941 * @param force_in_a:    a
00942 * @return force_in_b    b
00943 * \endchinese
00944 *
00945 * \english
00946 * @brief forceTrans:
00947 * Transform the reference frame of force and torque: force_in_b = pose_a_in_b * force_in_a
00948 * @param pose_a_in_b: pose of frame a in frame b
00949 * @param force_in_a: force and torque described in frame a
00950 * @return Force_in_b, force and torque described in frame b
00951 * \endenglish
00952 */
00953 std::vector<double> forceTrans(const std::vector<double> &pose_a_in_b,
00954                               const std::vector<double> &force_in_a);
00955 /**
00956 * @ingroup Math
00957 * \chinese
00958 * @brief
00959 * @param distances: N    , N >=3
00960 * @param position:
00961 * @param radius:    tcp
00962 * @param track_scale:    , (0, 1], 1
00963 * @return
00964 * \endchinese
00965 *
00966 * \english
00967 * @brief Calculate pose increment in tool coordinate system based on sensor data
00968 * @param distances: N distances, N >= 3
00969 * @param position: reference height to maintain from the trajectory
00970 * @param radius: effective radius from sensor center to tool TCP
00971 * @param track_scale: tracking ratio, range (0, 1], 1 means faster tracking
00972 * @return Pose increment in tool coordinate system
00973 * \endenglish
00974 */
00975 std::vector<double> getDeltaPoseBySensorDistance(
00976     const std::vector<double> &distances, double position, double radius,
00977     double track_scale);
00978 /**
00979 * @ingroup Math
00980 * \chinese
00981 * @brief changeFTFrame:
00982 * @param pose_a_in_b: a    b
00983 * @param ft_in_a:    a    a
00984 * @return ft_in_b    b    b
00985 * \endchinese
00986 *
00987 * \english
00988 * @brief changeFTFrame: Transform the reference frame of force and torque
00989 * @param pose_a_in_b: pose of frame a in frame b
00990 * @param ft_in_a: force and torque applied at point a, described in frame a
00991 * @return ft_in_b, force and torque applied at point b, described in frame b
00992 * \endenglish
00993 *
00994 * \endenglish
00995 */

```



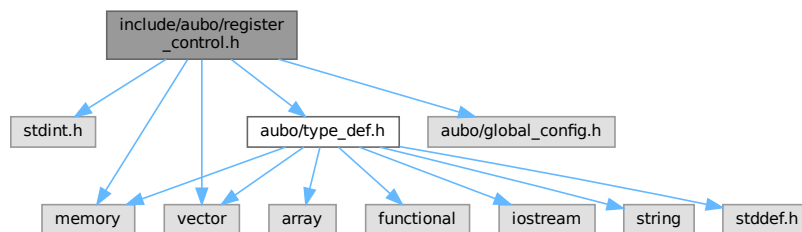
```

00996 std::vector<double> deltaPoseTrans(const std::vector<double> &pose_a_in_b,
00997                                   const std::vector<double> &ft_in_a);
00998
00999 /**
01000  * @ingroup Math
01001  * \chinese
01002  * @brief addDeltaPose:
01003  * @param pose_a_in_b: a b
01004  * @param v_in_b: a b
01005  * @return pose_in_b, b
01006  * \endchinese
01007  *
01008  * \english
01009  * @brief addDeltaPose: Calculate the pose after unit time given a velocity
01010  * @param pose_a_in_b: current pose of a relative to b
01011  * @param v_in_b: velocity of frame a described in frame b at current time
01012  * @return pose_in_b, pose after unit time described in frame b
01013  * \endenglish
01014  */
01015 std::vector<double> deltaPoseAdd(const std::vector<double> &pose_a_in_b,
01016                                 const std::vector<double> &v_in_b);
01017
01018 /**
01019  * @ingroup Math
01020  * \chinese
01021  * @brief changePoseWithXYRef: pose_tar , pose_ref
01022  * @param pose_tar:
01023  * @param pose_ref:
01024  * @return pose_tar xyz z
01025  * \endchinese
01026  *
01027  * \english
01028  * @brief changePoseWithXYRef: Modify the XY axis direction of pose_tar to be as consistent as possible with pose_ref
01029  * @param pose_tar: target pose to be modified
01030  * @param pose_ref: reference pose
01031  * @return Modified pose, using the xyz coordinates and z axis direction of pose_tar
01032  * \endenglish
01033  */
01034 std::vector<double> changePoseWithXYRef(
01035     const std::vector<double> &pose_tar,
01036     const std::vector<double> &pose_ref);
01037
01038 /**
01039  * @ingroup Math
01040  * \chinese
01041  * @brief homMatrixToPose:
01042  * @param homMatrix: 4*4 ,
01043  * @return
01044  * \endchinese
01045  *
01046  * \english
01047  * @brief homMatrixToPose: Get pose from homogeneous transformation matrix
01048  * @param homMatrix: 4x4 homogeneous transformation matrix, input elements are arranged row-wise
01049  * @return corresponding pose
01050  * \endenglish
01051  */
01052 std::vector<double> homMatrixToPose(const std::vector<double> &homMatrix);
01053
01054 /**
01055  * @ingroup Math
01056  * \chinese
01057  * @brief poseToHomMatrix:
01058  * @param pose:
01059  * @return ,
01060  * \endchinese
01061  *
01062  * \english
01063  * @brief poseToHomMatrix: Get homogeneous transformation matrix from pose
01064  * @param pose: input pose
01065  * @return output homogeneous transformation matrix, elements arranged row-wise
01066  * \endenglish
01067  */
01068 std::vector<double> poseToHomMatrix(const std::vector<double> &pose);
01069
01070 protected:
01071 void *d_;
01072 };
01073 using MathPtr = std::shared_ptr<Math>;
01074
01075 } // namespace common_interface
01076 } // namespace arcs
01077 #endif

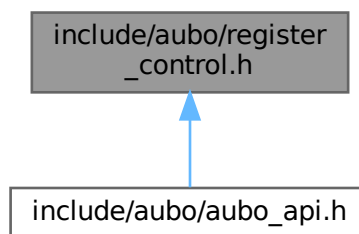
```

12.18 include/aubo/register_control.h File Reference

```
#include <stdint.h>
#include <memory>
#include <vector>
#include <aubo/type_def.h>
#include <aubo/global_config.h>
Include dependency graph for register_control.h:
```



This graph shows which files directly or indirectly include this file:



Classes

- class [arcs::common_interface::RegisterControl](#)

Namespaces

- namespace [arcs](#)
- namespace [arcs::common_interface](#)

Typedefs

- using [arcs::common_interface::RegisterControlPtr](#) = `std::shared_ptr<RegisterControl>`

Enumerations

- enum [ModbusErrorNum](#) {
[MB_ERR_NOT_INIT](#) = -1 , [MB_ERR_DISCONNECTED](#) = -2 , [MB_ERR_ILLEGAL_FUNCTION](#) = 1 , [MB_ERR_ILLEGAL_DATA_ADDRESS](#) = 2 ,
[MB_ERR_ILLEGAL_DATA_VALUE](#) = 3 , [MB_ERR_SLAVE_DEVICE_FAILURE](#) = 4 ,
[MB_ERR_ACKNOWLEDGE](#) = 5 , [MB_ERR_SLAVE_DEVICE_BUSY](#) = 6 }

12.18.1 Detailed Description

Definition in file [register_control.h](#).

12.18.2 Enumeration Type Documentation

12.18.2.1 ModbusErrorNum

enum [ModbusErrorNum](#)

Enumerator

MB_ERR_NOT_INIT	MODBUS unit not initialized.
MB_ERR_DISCONNECTED	MODBUS unit disconnected.
MB_ERR_ILLEGAL_FUNCTION	The function code received in the query is not an allowable action for the server (or slave).
MB_ERR_ILLEGAL_DATA_ADDRESS	The function code received in the query is not an allowable action for the server (or slave), check that the entered signal address corresponds to the setup of the remote MODBUS server.
MB_ERR_ILLEGAL_DATA_VALUE	A value contained in the query data field is not an allowable value for server (or slave), check that the entered signal value is valid for the specified address on the remote MODBUS server.
MB_ERR_SLAVE_DEVICE_FAILURE	An unrecoverable error occurred while the server (or slave) was attempting to perform the requested action.
MB_ERR_ACKNOWLEDGE	Specialized use in conjunction with programming commands sent to the remote MODBUS unit.
MB_ERR_SLAVE_DEVICE_BUSY	Specialized use in conjunction with programming commands sent to the remote MODBUS unit, the slave (server) is not able to respond now.

Definition at line 14 of file [register_control.h](#).

12.19 register_control.h

[Go to the documentation of this file.](#)

```

00001 /** @file register_control.h
00002  * @brief
00003  */
00004 #ifndef AUBO_SDK_REGISTER_CONTROL_INTERFACE_H
00005 #define AUBO_SDK_REGISTER_CONTROL_INTERFACE_H
00006
00007 #include <stdint.h>
00008 #include <memory>
00009 #include <vector>
00010
00011 #include <aubo/type_def.h>
00012 #include <aubo/global_config.h>
00013
00014 enum ModbusErrorNum
00015 {
00016     /** MODBUS unit not initialized
00017     */
00018     MB_ERR_NOT_INIT = -1,
00019
00020     /** MODBUS unit disconnected
00021     */
00022     MB_ERR_DISCONNECTED = -2,
00023
00024     /** The function code received in the query is not an allowable action for
00025     * the server (or slave).
00026     */
00027     MB_ERR_ILLEGAL_FUNCTION = 1,
00028
00029     /** The function code received in the query is not an allowable action for
00030     * the server (or slave), check that the entered signal address corresponds
00031     * to the setup of the remote MODBUS server.
00032     */
00033     MB_ERR_ILLEGAL_DATA_ADDRESS = 2,
00034
00035     /** A value contained in the query data field is not an allowable value for
00036     * server (or slave), check that the entered signal value is valid for the
00037     * specified address on the remote MODBUS server.
00038     */
00039     MB_ERR_ILLEGAL_DATA_VALUE = 3,
00040
00041     /** An unrecoverable error occurred while the server (or slave) was
00042     * attempting to perform the requested action.
00043     */
00044     MB_ERR_SLAVE_DEVICE_FAILURE = 4,
00045
00046     /** Specialized use in conjunction with programming commands sent to the
00047     * remote MODBUS unit.
00048     */
00049     MB_ERR_ACKNOWLEDGE = 5,
00050
00051     /** Specialized use in conjunction with programming commands sent to the
00052     * remote MODBUS unit, the slave (server) is not able to respond now
00053     */
00054     MB_ERR_SLAVE_DEVICE_BUSY = 6,
00055 };
00056
00057 namespace arcs {
00058     namespace common_interface {
00059
00060         /**
00061         * @defgroup RegisterControl RegisterControl ( )
00062         * \-chinese \-english General Registers
00063         */
00064         class ARCS_ABI_EXPORT RegisterControl
00065         {
00066         public:
00067             RegisterControl();
00068             virtual ~RegisterControl();
00069
00070             /**
00071             * @ingroup RegisterControl
00072             * \-chinese
00073             *
00074             *
00075             *
00076             *
00077             * @param address 0:127
00078             * @return true false
00079             *
00080             * @note [0:63] FieldBus/PLC
00081             * [64:127] FieldBus/PLC RTDE
00082             *

```

```

00083 * @par Python
00084 * getBoolInput(self: pyaubo_sdk.RegisterControl, arg0: int) -> bool
00085 *
00086 * @par Lua
00087 * getBoolInput(address: number) -> boolean
00088 *
00089 * @par Lua
00090 * BoolInput_0 = getBoolInput(0)
00091 *
00092 * @par JSON-RPC
00093 * {"jsonrpc":"2.0","method":"RegisterControl.getBoolInput","params":[0],"id":1}
00094 *
00095 * @par JSON-RPC
00096 * {"id":1,"jsonrpc":"2.0","result":false}
00097 * \endchinese
00098 *
00099 * \english
00100 * Reads the boolean from one of the input registers, which can also be
00101 * accessed by a Field bus. Note, uses its own memory space.
00102 *
00103 * @param address Address of the register (0:127)
00104 * @return Boolean value held by the register (true, false)
00105 *
00106 * @note The lower range of the boolean input registers [0:63] is reserved
00107 * for FieldBus/PLC interface usage. The upper range [64:127] cannot be
00108 * accessed by FieldBus/PLC interfaces, since it is reserved for external
00109 * RTDE clients.
00110 *
00111 * @par Python interface prototype
00112 * getBoolInput(self: pyaubo_sdk.RegisterControl, arg0: int) -> bool
00113 *
00114 * @par Lua interface prototype
00115 * getBoolInput(address: number) -> boolean
00116 *
00117 * @par Lua example
00118 * BoolInput_0 = getBoolInput(0)
00119 *
00120 * @par JSON-RPC request example
00121 * {"jsonrpc":"2.0","method":"RegisterControl.getBoolInput","params":[0],"id":1}
00122 *
00123 * @par JSON-RPC response example
00124 * {"id":1,"jsonrpc":"2.0","result":false}
00125 * \endenglish
00126 */
00127 bool getBoolInput(uint32_t address);
00128
00129 /**
00130 * @ingroup RegisterControl
00131 * \chinese
00132 *
00133 * @param address
00134 * @param value
00135 * @return
00136 *
00137 * @note RTDE/Modbus Slave/PLC
00138 *
00139 * @par Python
00140 * setBoolInput(self: pyaubo_sdk.RegisterControl, arg0: int, arg1: bool) ->
00141 * int
00142 *
00143 * @par Lua
00144 * setBoolInput(address: number, value: boolean) -> nil
00145 *
00146 * @par Lua
00147 * setBoolInput(0)
00148 *
00149 * @par JSON-RPC
00150 * {"jsonrpc":"2.0","method":"RegisterControl.setBoolInput","params":[0,true],"id":1}
00151 *
00152 * @par JSON-RPC
00153 * {"id":1,"jsonrpc":"2.0","result":0}
00154 *
00155 * \endchinese
00156 * \english
00157 *
00158 * @param address Address of the register (0:127)
00159 * @param value Boolean value to set (true or false)
00160 * @return Returns 0 on success, or an error code
00161 *
00162 * @note Only used when implementing RTDE/Modbus Slave/PLC server
00163 *
00164 * @par Python interface prototype
00165 * setBoolInput(self: pyaubo_sdk.RegisterControl, arg0: int, arg1: bool) ->
00166 * int
00167 *
00168 * @par Lua interface prototype
00169 * setBoolInput(address: number, value: boolean) -> nil

```

```

00170 *
00171 * @par Lua example
00172 * setBoolInput(0)
00173 *
00174 * @par JSON-RPC request example
00175 * {"jsonrpc":"2.0","method":"RegisterControl.setBoolInput","params":[0,true],"id":1}
00176 *
00177 * @par JSON-RPC response example
00178 * {"id":1,"jsonrpc":"2.0","result":0}
00179 *
00180 * \endenglish
00181 */
00182 int setBoolInput(uint32_t address, bool value);
00183
00184 /**
00185 * @ingroup RegisterControl
00186 * \chinese
00187 *
00188 *
00189 *
00190 *
00191 * @param address 0:47
00192 * @return [-2,147,483,648 : 2,147,483,647]
00193 *
00194 * @note [0:23] FieldBus/PLC
00195 * [24:47] FieldBus/PLC RTDE
00196 *
00197 * @par Python
00198 * getInt32Input(self: pyaubo_sdk.RegisterControl, arg0: int) -> int
00199 *
00200 * @par Lua
00201 * getInt32Input(address: number) -> number
00202 *
00203 * @par Lua
00204 * Int32Input_0 = getInt32Input(0)
00205 *
00206 * @par JSON-RPC
00207 * {"jsonrpc":"2.0","method":"RegisterControl.getInt32Input","params":[0],"id":1}
00208 *
00209 * @par JSON-RPC
00210 * {"id":1,"jsonrpc":"2.0","result":0}
00211 * \endchinese
00212 *
00213 * \english
00214 * Reads the integer from one of the input registers, which can also be
00215 * accessed by a FieldBus. Note, uses it's own memory space.
00216 *
00217 * @param address Address of the register (0:47)
00218 * @return The value held by the register [-2,147,483,648 : 2,147,483,647]
00219 *
00220 * @note The lower range of the integer input registers [0:23] is reserved
00221 * for FieldBus/PLC interface usage. The upper range [24:47] cannot be
00222 * accessed by FieldBus/PLC interfaces, since it is reserved for external
00223 * RTDE clients.
00224 *
00225 * @par Python interface prototype
00226 * getInt32Input(self: pyaubo_sdk.RegisterControl, arg0: int) -> int
00227 *
00228 * @par Lua interface prototype
00229 * getInt32Input(address: number) -> number
00230 *
00231 * @par Lua example
00232 * Int32Input_0 = getInt32Input(0)
00233 *
00234 * @par JSON-RPC request example
00235 * {"jsonrpc":"2.0","method":"RegisterControl.getInt32Input","params":[0],"id":1}
00236 *
00237 * @par JSON-RPC response example
00238 * {"id":1,"jsonrpc":"2.0","result":0}
00239 * \endchinese
00240 */
00241 int getInt32Input(uint32_t address);
00242
00243 /**
00244 * @ingroup RegisterControl
00245 * @ingroup RegisterControl
00246 * \chinese
00247 *
00248 * @param address 0:47
00249 * @param value
00250 * @return 0
00251 *
00252 * @note RTDE/Modbus Slave/PLC
00253 *
00254 * @par Python
00255 * setInt32Input(self: pyaubo_sdk.RegisterControl, arg0: int, arg1: int) ->
00256 * int

```

```

00257 *
00258 * @par Lua
00259 * setInt32Input(address: number, value: number) -> nil
00260 *
00261 * @par Lua
00262 * setInt32Input(0)
00263 *
00264 * @par JSON-RPC
00265 * {"jsonrpc":"2.0","method":"RegisterControl.setInt32Input","params":[0,33],"id":1}
00266 *
00267 * @par JSON-RPC
00268 * {"id":1,"jsonrpc":"2.0","result":0}
00269 *
00270 * \endchinese
00271 * \english
00272 *
00273 * @param address Address of the register (0:47)
00274 * @param value Integer value to set
00275 * @return Returns 0 on success, or an error code
00276 *
00277 * @note Only used when implementing RTDE/Modbus Slave/PLC server
00278 *
00279 * @par Python interface prototype
00280 * setInt32Input(self: pyaubo_sdk.RegisterControl, arg0: int, arg1: int) ->
00281 * int
00282 *
00283 * @par Lua interface prototype
00284 * setInt32Input(address: number, value: number) -> nil
00285 *
00286 * @par Lua example
00287 * setInt32Input(0)
00288 *
00289 * @par JSON-RPC request example
00290 * {"jsonrpc":"2.0","method":"RegisterControl.setInt32Input","params":[0,33],"id":1}
00291 *
00292 * @par JSON-RPC response example
00293 * {"id":1,"jsonrpc":"2.0","result":0}
00294 *
00295 * \endenglish
00296 */
00297 int setInt32Input(uint32_t address, int value);
00298
00299 /**
00300 * @ingroup RegisterControl
00301 * \chinese
00302 * Reads the float from one of the input registers, which can also be
00303 * accessed by a Field bus. Note, uses it's own memory space.
00304 *
00305 *
00306 *
00307 *
00308 * @param address Address of the register (0:47)
00309 * 0:47
00310 * @return The value held by the register (float)
00311 *
00312 *
00313 * @note The lower range of the float input registers [0:23] is reserved
00314 * for FieldBus/PLC interface usage. The upper range [24:47] cannot be
00315 * accessed by FieldBus/PLC interfaces, since it is reserved for external
00316 * RTDE clients.
00317 * [0:23] /PLC
00318 * [24:47] /PLC RTDE
00319 *
00320 * @par Python
00321 * getFloatInput(self: pyaubo_sdk.RegisterControl, arg0: int) -> float
00322 *
00323 * @par Lua
00324 * getFloatInput(address: number) -> number
00325 *
00326 * @par Lua
00327 * FloatInput_0 = getFloatInput(0)
00328 *
00329 * @par JSON-RPC
00330 * {"jsonrpc":"2.0","method":"RegisterControl.getFloatInput","params":[0],"id":1}
00331 *
00332 * @par JSON-RPC
00333 * {"id":1,"jsonrpc":"2.0","result":0.0}
00334 *
00335 * \endchinese
00336 *
00337 * \english
00338 * Reads the float from one of the input registers, which can also be
00339 * accessed by a Field bus. Note, uses it's own memory space.
00340 *
00341 * @param address Address of the register (0:47)
00342 * @return The value held by the register (float)
00343 *

```

```

00344 * @note The lower range of the float input registers [0:23] is reserved
00345 * for FieldBus/PLC interface usage. The upper range [24:47] cannot be
00346 * accessed by FieldBus/PLC interfaces, since it is reserved for external
00347 * RTDE clients.
00348 *
00349 * @par Python interface prototype
00350 * getFloatInput(self: pyaubo_sdk.RegisterControl, arg0: int) -> float
00351 *
00352 * @par Lua interface prototype
00353 * getFloatInput(address: number) -> number
00354 *
00355 * @par Lua example
00356 * FloatInput_0_0 = getFloatInput(0)
00357 *
00358 * @par JSON-RPC request example
00359 * {"jsonrpc": "2.0", "method": "RegisterControl.getFloatInput", "params": [0], "id": 1}
00360 *
00361 * @par JSON-RPC response example
00362 * {"id": 1, "jsonrpc": "2.0", "result": 0.0}
00363 */
00364 float getFloatInput(uint32_t address);
00365
00366 /**
00367 * @ingroup RegisterControl
00368 * \chinese
00369 *
00370 * @param address    0:47
00371 * @param value
00372 * @return 0
00373 *
00374 * @note    RTDE/Modbus Slave/PLC
00375 *
00376 * @par Python
00377 * setFloatInput(self: pyaubo_sdk.RegisterControl, arg0: int, arg1: float)
00378 * -> int
00379 *
00380 * @par Lua
00381 * setFloatInput(address: number, value: number) -> nil
00382 *
00383 * @par Lua
00384 * setFloatInput(0, 3.3)
00385 *
00386 * @par JSON-RPC
00387 * {"jsonrpc": "2.0", "method": "RegisterControl.setFloatInput", "params": [0, 3.3], "id": 1}
00388 *
00389 * @par JSON-RPC
00390 * {"id": 1, "jsonrpc": "2.0", "result": 0}
00391 *
00392 * \endchinese
00393 * \english
00394 *
00395 * @param address Address of the register (0:47)
00396 * @param value Float value to set
00397 * @return Returns 0 on success, or an error code
00398 *
00399 * @note Only used when implementing RTDE/Modbus Slave/PLC server
00400 *
00401 * @par Python interface prototype
00402 * setFloatInput(self: pyaubo_sdk.RegisterControl, arg0: int, arg1: float)
00403 * -> int
00404 *
00405 * @par Lua interface prototype
00406 * setFloatInput(address: number, value: number) -> nil
00407 *
00408 * @par Lua example
00409 * setFloatInput(0, 3.3)
00410 *
00411 * @par JSON-RPC request example
00412 * {"jsonrpc": "2.0", "method": "RegisterControl.setFloatInput", "params": [0, 3.3], "id": 1}
00413 *
00414 * @par JSON-RPC response example
00415 * {"id": 1, "jsonrpc": "2.0", "result": 0}
00416 *
00417 * \endenglish
00418 */
00419 int setFloatInput(uint32_t address, float value);
00420
00421 /**
00422 * @ingroup RegisterControl
00423 * \chinese
00424 *
00425 *
00426 *
00427 *
00428 * @param address    0:47
00429 * @return
00430 *

```



```

00431 * @par Python
00432 * getDoubleInput(self: pyaubo_sdk.RegisterControl, arg0: int) -> float
00433 *
00434 * @par Lua
00435 * getDoubleInput(address: number) -> number
00436 *
00437 * @par Lua
00438 * DoubleInput_0 = getDoubleInput(0)
00439 *
00440 * @par JSON-RPC
00441 * {"jsonrpc": "2.0", "method": "RegisterControl.getDoubleInput", "params": [0], "id": 1}
00442 *
00443 * @par JSON-RPC
00444 * {"id": 1, "jsonrpc": "2.0", "result": 0.0}
00445 *
00446 * \endchinese
00447 * \english
00448 *
00449 * Reads the double value from one of the input registers, which can also be
00450 * accessed by a FieldBus. Note, uses its own memory space.
00451 *
00452 * @param address Address of the register (0:47)
00453 * @return The double value held by the register
00454 *
00455 * @par Python interface prototype
00456 * getDoubleInput(self: pyaubo_sdk.RegisterControl, arg0: int) -> float
00457 *
00458 * @par Lua interface prototype
00459 * getDoubleInput(address: number) -> number
00460 *
00461 * @par Lua example
00462 * DoubleInput_0 = getDoubleInput(0)
00463 *
00464 * @par JSON-RPC request example
00465 * {"jsonrpc": "2.0", "method": "RegisterControl.getDoubleInput", "params": [0], "id": 1}
00466 *
00467 * @par JSON-RPC response example
00468 * {"id": 1, "jsonrpc": "2.0", "result": 0.0}
00469 *
00470 * \endenglish
00471 */
00472 double getDoubleInput(uint32_t address);
00473
00474 /**
00475 * \english
00476 * @param address
00477 * @param value
00478 * @return
00479 *
00480 * @note Only used when implementing RTDE/Modbus Slave/PLC server
00481 *
00482 * @par Python interface prototype
00483 * setDoubleInput(self: pyaubo_sdk.RegisterControl, arg0: int, arg1: float)
00484 * -> int
00485 *
00486 * @par Lua interface prototype
00487 * setDoubleInput(address: number, value: number) -> nil
00488 *
00489 * @par Lua example
00490 * setDoubleInput(0, 3.3)
00491 *
00492 * @par JSON-RPC request example
00493 * {"jsonrpc": "2.0", "method": "RegisterControl.setDoubleInput", "params": [0, 6.6], "id": 1}
00494 *
00495 * @par JSON-RPC response example
00496 * {"id": 1, "jsonrpc": "2.0", "result": 0}
00497 * \endenglish
00498 *
00499 * @ingroup RegisterControl
00500 * \chinese
00501 * @param address
00502 * @param value
00503 * @return
00504 *
00505 * @note RTDE/Modbus Slave/PLC
00506 *
00507 * @par Python
00508 * setDoubleInput(self: pyaubo_sdk.RegisterControl, arg0: int, arg1: float)
00509 * -> int
00510 *
00511 * @par Lua
00512 * setDoubleInput(address: number, value: number) -> nil
00513 *
00514 * @par Lua
00515 * setDoubleInput(0, 3.3)
00516 *
00517 * @par JSON-RPC

```

```

00518 * {"jsonrpc":"2.0","method":"RegisterControl.setDoubleInput","params":[0,6.6],"id":1}
00519 *
00520 * @par JSON-RPC
00521 * {"id":1,"jsonrpc":"2.0","result":0}
00522 * \endchinese
00523 */
00524 int setDoubleInput(uint32_t address, double value);
00525
00526 /**
00527 * @ingroup RegisterControl
00528 * \chinese
00529 *
00530 *
00531 *
00532 * @param address 0:127
00533 * @return true, false
00534 *
00535 * @note [0:63] /PLC
00536 * [64:127] /PLC RTDE
00537 *
00538 * @par Python
00539 * getBoolOutput(self: pyaubo_sdk.RegisterControl, arg0: int) -> bool
00540 *
00541 * @par Lua
00542 * getBoolOutput(address: number) -> boolean
00543 *
00544 * @par Lua
00545 * BoolOutput_0 = getBoolOutput(0)
00546 *
00547 * @par JSON-RPC
00548 * {"jsonrpc":"2.0","method":"RegisterControl.getBoolOutput","params":[0],"id":1}
00549 *
00550 * @par JSON-RPC
00551 * {"id":1,"jsonrpc":"2.0","result":false}
00552 * \endchinese
00553 *
00554 * \english
00555 * Reads the boolean from one of the output registers, which can also be
00556 * accessed by a Field bus.
00557 * Note, uses its own memory space.
00558 *
00559 * @param address Address of the register (0:127)
00560 * @return The boolean value held by the register (true, false)
00561 *
00562 * @note The lower range of the boolean output registers [0:63] is reserved
00563 * for FieldBus/PLC interface usage. The upper range [64:127] cannot be
00564 * accessed by FieldBus/PLC interfaces, since it is reserved for external
00565 * RTDE clients.
00566 *
00567 * @par Python interface prototype
00568 * getBoolOutput(self: pyaubo_sdk.RegisterControl, arg0: int) -> bool
00569 *
00570 * @par Lua interface prototype
00571 * getBoolOutput(address: number) -> boolean
00572 *
00573 * @par Lua example
00574 * BoolOutput_0 = getBoolOutput(0)
00575 *
00576 * @par JSON-RPC request example
00577 * {"jsonrpc":"2.0","method":"RegisterControl.getBoolOutput","params":[0],"id":1}
00578 *
00579 * @par JSON-RPC response example
00580 * {"id":1,"jsonrpc":"2.0","result":false}
00581 * \endenglish
00582 */
00583 bool getBoolOutput(uint32_t address);
00584
00585 /**
00586 * @ingroup RegisterControl
00587 * \chinese
00588 *
00589 * @param address 0:127
00590 * @param value true false
00591 * @return 0
00592 *
00593 * @par Python
00594 * setBoolOutput(self: pyaubo_sdk.RegisterControl, arg0: int, arg1: bool) ->
00595 * int
00596 *
00597 * @par Lua
00598 * setBoolOutput(address: number, value: boolean) -> nil
00599 *
00600 * @par Lua
00601 * setBoolOutput(0)
00602 *
00603 * @par JSON-RPC
00604 * {"jsonrpc":"2.0","method":"RegisterControl.setBoolOutput","params":[0,false],"id":1}

```

```

00605  *
00606  * @par JSON-RPC
00607  * {"id":1,"jsonrpc":"2.0","result":0}
00608  *
00609  * \endchinese
00610  * \english
00611  *
00612  * @param address Address of the register (0:127)
00613  * @param value Boolean value to set (true or false)
00614  * @return Returns 0 on success, or an error code
00615  *
00616  * @par Python interface prototype
00617  * setBoolOutput(self: pyaubo_sdk.RegisterControl, arg0: int, arg1: bool) ->
00618  * int
00619  *
00620  * @par Lua interface prototype
00621  * setBoolOutput(address: number, value: boolean) -> nil
00622  *
00623  * @par Lua example
00624  * setBoolOutput(0)
00625  *
00626  * @par JSON-RPC request example
00627  * {"jsonrpc":"2.0","method":"RegisterControl.setBoolOutput","params":[0,false],"id":1}
00628  *
00629  * @par JSON-RPC response example
00630  * {"id":1,"jsonrpc":"2.0","result":0}
00631  *
00632  * \endenglish
00633  */
00634 int setBoolOutput(uint32_t address, bool value);
00635
00636 /**
00637  * @ingroup RegisterControl
00638  * \chinese
00639  *
00640  *
00641  *
00642  * @param address 0:47
00643  * @return -2,147,483,648 : 2,147,483,647
00644  *
00645  * @note [0:23] /PLC
00646  * [24:47] /PLC RTDE
00647  *
00648  * @par Python
00649  * getInt32Output(self: pyaubo_sdk.RegisterControl, arg0: int) -> int
00650  *
00651  * @par Lua
00652  * getInt32Output(address: number) -> number
00653  *
00654  * @par Lua
00655  * Int32Output_0 = getInt32Output(0)
00656  *
00657  * @par JSON-RPC
00658  * {"jsonrpc":"2.0","method":"RegisterControl.getInt32Output","params":[0],"id":1}
00659  *
00660  * @par JSON-RPC
00661  * {"id":1,"jsonrpc":"2.0","result":0}
00662  * \endchinese
00663  *
00664  * \english
00665  * Reads the integer from one of the output registers, which can also be
00666  * accessed by a FieldBus. Note, uses its own memory space.
00667  *
00668  * @param address Address of the register (0:47)
00669  * @return The int value held by the register [-2,147,483,648 :
00670  * 2,147,483,647]
00671  *
00672  * @note The lower range of the integer output registers [0:23] is reserved
00673  * for FieldBus/PLC interface usage. The upper range [24:47] cannot be
00674  * accessed by FieldBus/PLC interfaces, since it is reserved for external
00675  * RTDE clients.
00676  *
00677  * @par Python interface prototype
00678  * getInt32Output(self: pyaubo_sdk.RegisterControl, arg0: int) -> int
00679  *
00680  * @par Lua interface prototype
00681  * getInt32Output(address: number) -> number
00682  *
00683  * @par Lua example
00684  * Int32Output_0 = getInt32Output(0)
00685  *
00686  * @par JSON-RPC request example
00687  * {"jsonrpc":"2.0","method":"RegisterControl.getInt32Output","params":[0],"id":1}
00688  *
00689  * @par JSON-RPC response example
00690  * {"id":1,"jsonrpc":"2.0","result":0}
00691  * \endenglish

```

```

00692     */
00693     int getInt32Output(uint32_t address);
00694
00695     /**
00696     * @ingroup RegisterControl
00697     * \chinese
00698     *
00699     * @param address    0:47
00700     * @param value
00701     * @return 0
00702     *
00703     * @par Python
00704     * setInt32Output(self: pyaubo_sdk.RegisterControl, arg0: int, arg1: int) ->
00705     * int
00706     *
00707     * @par Lua
00708     * setInt32Output(address: number, value: number) -> nil
00709     *
00710     * @par Lua
00711     * setInt32Output(0, 100)
00712     *
00713     * @par JSON-RPC
00714     * {"jsonrpc":"2.0","method":"RegisterControl.setInt32Output","params":[0,100],"id":1}
00715     *
00716     * @par JSON-RPC
00717     * {"id":1,"jsonrpc":"2.0","result":0}
00718     *
00719     * \endchinese
00720     * \english
00721     *
00722     * @param address Address of the register (0:47)
00723     * @param value Integer value to set
00724     * @return Returns 0 on success, or an error code
00725     *
00726     * @par Python interface prototype
00727     * setInt32Output(self: pyaubo_sdk.RegisterControl, arg0: int, arg1: int) ->
00728     * int
00729     *
00730     * @par Lua interface prototype
00731     * setInt32Output(address: number, value: number) -> nil
00732     *
00733     * @par Lua example
00734     * setInt32Output(0, 100)
00735     *
00736     * @par JSON-RPC request example
00737     * {"jsonrpc":"2.0","method":"RegisterControl.setInt32Output","params":[0,100],"id":1}
00738     *
00739     * @par JSON-RPC response example
00740     * {"id":1,"jsonrpc":"2.0","result":0}
00741     *
00742     * \endenglish
00743     */
00744     int setInt32Output(uint32_t address, int value);
00745
00746     /**
00747     * @ingroup RegisterControl
00748     * \chinese
00749     *
00750     *
00751     *
00752     * @param address    0:47
00753     * @return          float
00754     *
00755     * @note            [0:23] /PLC
00756     *                  [24:47] /PLC      RTDE
00757     *
00758     * @par Python
00759     * getFloatOutput(self: pyaubo_sdk.RegisterControl, arg0: int) -> float
00760     *
00761     * @par Lua
00762     * getFloatOutput(address: number) -> number
00763     *
00764     * @par Lua
00765     * FloatOutput_0 = getFloatOutput(0)
00766     *
00767     * @par JSON-RPC
00768     * {"jsonrpc":"2.0","method":"RegisterControl.getFloatOutput","params":[0],"id":1}
00769     *
00770     * @par JSON-RPC
00771     * {"id":1,"jsonrpc":"2.0","result":3.3}
00772     * \endchinese
00773     *
00774     * \english
00775     * Reads the float from one of the output registers, which can also be
00776     * accessed by a FieldBus. Note, uses its own memory space.
00777     *
00778     * @param address Address of the register (0:47)

```

```

00779 * @return The value held by the register (float)
00780 *
00781 * @note The lower range of the float output registers [0:23] is reserved
00782 * for FieldBus/PLC interface usage. The upper range [24:47] cannot be
00783 * accessed by FieldBus/PLC interfaces, since it is reserved for external
00784 * RTDE clients.
00785 *
00786 * @par Python interface prototype
00787 * getFloatOutput(self: pyaubo_sdk.RegisterControl, arg0: int) -> float
00788 *
00789 * @par Lua interface prototype
00790 * getFloatOutput(address: number) -> number
00791 *
00792 * @par Lua example
00793 * FloatOutput_0 = getFloatOutput(0)
00794 *
00795 * @par JSON-RPC request example
00796 * {"jsonrpc": "2.0", "method": "RegisterControl.getFloatOutput", "params": [0], "id": 1}
00797 *
00798 * @par JSON-RPC response example
00799 * {"id": 1, "jsonrpc": "2.0", "result": 3.3}
00800 * \endenglish
00801 */
00802 float getFloatOutput(uint32_t address);
00803
00804 /**
00805 * @ingroup RegisterControl
00806 * \chinese
00807 *
00808 * @param address 0:47
00809 * @param value
00810 * @return 0
00811 *
00812 * @par Python
00813 * setFloatOutput(self: pyaubo_sdk.RegisterControl, arg0: int, arg1: float)
00814 * -> int
00815 *
00816 * @par Lua
00817 * setFloatOutput(address: number, value: number) -> nil
00818 *
00819 * @par Lua
00820 * setFloatOutput(0,5.5)
00821 *
00822 * @par JSON-RPC
00823 * {"jsonrpc": "2.0", "method": "RegisterControl.setFloatOutput", "params": [0,5.5], "id": 1}
00824 *
00825 * @par JSON-RPC
00826 * {"id": 1, "jsonrpc": "2.0", "result": 0}
00827 *
00828 * \endchinese
00829 * \english
00830 *
00831 * @param address Address of the register (0:47)
00832 * @param value Float value to set
00833 * @return Returns 0 on success, or an error code
00834 *
00835 * @par Python interface prototype
00836 * setFloatOutput(self: pyaubo_sdk.RegisterControl, arg0: int, arg1: float)
00837 * -> int
00838 *
00839 * @par Lua interface prototype
00840 * setFloatOutput(address: number, value: number) -> nil
00841 *
00842 * @par Lua example
00843 * setFloatOutput(0,5.5)
00844 *
00845 * @par JSON-RPC request example
00846 * {"jsonrpc": "2.0", "method": "RegisterControl.setFloatOutput", "params": [0,5.5], "id": 1}
00847 *
00848 * @par JSON-RPC response example
00849 * {"id": 1, "jsonrpc": "2.0", "result": 0}
00850 *
00851 * \endenglish
00852 */
00853 int setFloatOutput(uint32_t address, float value);
00854
00855 /**
00856 * @ingroup RegisterControl
00857 * \chinese
00858 *
00859 *
00860 *
00861 * @param address
00862 * @return
00863 *
00864 * @par Python
00865 * getDoubleOutput(self: pyaubo_sdk.RegisterControl, arg0: int) -> float

```

```

00866 *
00867 * @par Lua
00868 * getDoubleOutput(address: number) -> number
00869 *
00870 * @par Lua
00871 * DoubleOutput_0 = getDoubleOutput(0)
00872 *
00873 * @par JSON-RPC
00874 * {"jsonrpc":"2.0","method":"RegisterControl.getDoubleOutput","params":[0],"id":1}
00875 *
00876 * @par JSON-RPC
00877 * {"id":1,"jsonrpc":"2.0","result":0.0}
00878 *
00879 * \endchinese
00880 * \english
00881 *
00882 * Reads the double value from one of the output registers.
00883 *
00884 * @param address Address of the register
00885 * @return The double value held by the register
00886 *
00887 * @par Python interface prototype
00888 * getDoubleOutput(self: pyaubo_sdk.RegisterControl, arg0: int) -> float
00889 *
00890 * @par Lua interface prototype
00891 * getDoubleOutput(address: number) -> number
00892 *
00893 * @par Lua example
00894 * DoubleOutput_0 = getDoubleOutput(0)
00895 *
00896 * @par JSON-RPC request example
00897 * {"jsonrpc":"2.0","method":"RegisterControl.getDoubleOutput","params":[0],"id":1}
00898 *
00899 * @par JSON-RPC response example
00900 * {"id":1,"jsonrpc":"2.0","result":0.0}
00901 *
00902 * \endenglish
00903 */
00904 double getDoubleOutput(uint32_t address);
00905
00906 /**
00907 * @ingroup RegisterControl
00908 * \chinese
00909 *
00910 * @param address
00911 * @param value
00912 * @return 0
00913 *
00914 * @par Python
00915 * setDoubleOutput(self: pyaubo_sdk.RegisterControl, arg0: int, arg1: float)
00916 * -> int
00917 *
00918 * @par Lua
00919 * setDoubleOutput(address: number, value: number) -> nil
00920 *
00921 * @par Lua
00922 * setDoubleOutput(0,4.4)
00923 *
00924 * @par JSON-RPC
00925 * {"jsonrpc":"2.0","method":"RegisterControl.setDoubleOutput","params":[0,4.4],"id":1}
00926 *
00927 * @par JSON-RPC
00928 * {"id":1,"jsonrpc":"2.0","result":0}
00929 *
00930 * \endchinese
00931 * \english
00932 *
00933 * @param address Address of the register
00934 * @param value Double value to set
00935 * @return Returns 0 on success, or an error code
00936 *
00937 * @par Python interface prototype
00938 * setDoubleOutput(self: pyaubo_sdk.RegisterControl, arg0: int, arg1: float)
00939 * -> int
00940 *
00941 * @par Lua interface prototype
00942 * setDoubleOutput(address: number, value: number) -> nil
00943 *
00944 * @par Lua example
00945 * setDoubleOutput(0,4.4)
00946 *
00947 * @par JSON-RPC request example
00948 * {"jsonrpc":"2.0","method":"RegisterControl.setDoubleOutput","params":[0,4.4],"id":1}
00949 *
00950 * @par JSON-RPC response example
00951 * {"id":1,"jsonrpc":"2.0","result":0}
00952 *

```

```

00953     * \endenglish
00954     */
00955     int setDoubleOutput(uint32_t address, double value);
00956
00957     /**
00958     * @ingroup RegisterControl
00959     * \chinese
00960     *   Modbus Slave
00961     *
00962     * @param address
00963     * @return
00964     *
00965     * @par Python
00966     * getInt16Register(self: pyaubo_sdk.RegisterControl, arg0: int) -> int
00967     *
00968     * @par Lua
00969     * getInt16Register(address: number) -> number
00970     *
00971     * @par Lua
00972     * Int16Register_0 = getInt16Register(0)
00973     *
00974     * @par JSON-RPC
00975     * {"jsonrpc":"2.0","method":"RegisterControl.getInt16Register","params":[0],"id":1}
00976     *
00977     * @par JSON-RPC
00978     * {"id":1,"jsonrpc":"2.0","result":0}
00979     *
00980     * \endchinese
00981     * \english
00982     * Used for Modbus Slave
00983     *
00984     * @param address
00985     * @return
00986     *
00987     * @par Python interface prototype
00988     * getInt16Register(self: pyaubo_sdk.RegisterControl, arg0: int) -> int
00989     *
00990     * @par Lua interface prototype
00991     * getInt16Register(address: number) -> number
00992     *
00993     * @par Lua example
00994     * Int16Register_0 = getInt16Register(0)
00995     *
00996     * @par JSON-RPC request example
00997     * {"jsonrpc":"2.0","method":"RegisterControl.getInt16Register","params":[0],"id":1}
00998     *
00999     * @par JSON-RPC response example
01000     * {"id":1,"jsonrpc":"2.0","result":0}
01001     *
01002     * \endenglish
01003     */
01004     int16_t getInt16Register(uint32_t address);
01005
01006     /**
01007     * @ingroup RegisterControl
01008     * \chinese
01009     *
01010     * @param address
01011     * @param value
01012     * @return 0
01013     *
01014     * @par Python
01015     * setInt16Register(self: pyaubo_sdk.RegisterControl, arg0: int, arg1: int)
01016     * -> int
01017     *
01018     * @par Lua
01019     * setInt16Register(address: number, value: number) -> nil
01020     *
01021     * @par Lua
01022     * setInt16Register(0,4.4)
01023     *
01024     * @par JSON-RPC
01025     * {"jsonrpc":"2.0","method":"RegisterControl.setInt16Register","params":[0,0],"id":1}
01026     *
01027     * @par JSON-RPC
01028     * {"id":1,"jsonrpc":"2.0","result":0}
01029     *
01030     * \endchinese
01031     * \english
01032     *
01033     * @param address Register address
01034     * @param value Value to set
01035     * @return Returns 0 on success, otherwise error code
01036     *
01037     * @par Python interface prototype
01038     * setInt16Register(self: pyaubo_sdk.RegisterControl, arg0: int, arg1: int)
01039     * -> int

```

```

01040 *
01041 * @par Lua interface prototype
01042 * setInt16Register(address: number, value: number) -> nil
01043 *
01044 * @par Lua example
01045 * setInt16Register(0,4.4)
01046 *
01047 * @par JSON-RPC request example
01048 * {"jsonrpc":"2.0","method":"RegisterControl.setInt16Register","params":[0,0],"id":1}
01049 *
01050 * @par JSON-RPC response example
01051 * {"id":1,"jsonrpc":"2.0","result":0}
01052 *
01053 * \endenglish
01054 */
01055 int setInt16Register(uint32_t address, int16_t value);
01056
01057 /**
01058 * @ingroup RegisterControl
01059 * \chinese
01060 *   Int16   bit
01061 *
01062 * @param address: Int16
01063 * @param bit_offset:   bit 0 ~ 15
01064 * @return bit   true 1 false 0
01065 *
01066 * @par Python
01067 * getInt16RegisterBit(self: pyaubo_sdk.RegisterControl, address: int,
01068 * bit_offset: int) -> bool
01069 *
01070 * @par Lua
01071 * getInt16RegisterBit(address: number, bit_offset: number) -> boolean
01072 *
01073 * @par JSON-RPC
01074 * {"jsonrpc":"2.0","method":"RegisterControl.getInt16RegisterBit","params":[1,
01075 * 5],"id":1}
01076 *
01077 * @par JSON-RPC
01078 * {"id":1,"jsonrpc":"2.0","result":true}
01079 *
01080 * \endchinese
01081 * \english
01082 * Get the status of a specific bit in an Int16 register
01083 *
01084 * @param address Int16 register address
01085 * @param bit_offset Bit offset in the register, 0 ~ 15
01086 * @return Status of the specified bit, true for 1, false for 0 or invalid
01087 * parameters
01088 *
01089 * @par Python interface prototype
01090 * getInt16RegisterBit(self: pyaubo_sdk.RegisterControl, address: int,
01091 * bit_offset: int) -> bool
01092 *
01093 * @par Lua interface prototype
01094 * getInt16RegisterBit(address: number, bit_offset: number) -> boolean
01095 *
01096 * @par JSON-RPC request example
01097 * {"jsonrpc":"2.0","method":"RegisterControl.getInt16RegisterBit","params":[1,
01098 * 5],"id":1}
01099 *
01100 * @par JSON-RPC response example
01101 * {"id":1,"jsonrpc":"2.0","result":true}
01102 *
01103 * \endenglish
01104 */
01105 bool getInt16RegisterBit(uint32_t address, uint8_t bit_offset);
01106
01107 /**
01108 * @ingroup RegisterControl
01109 * \chinese
01110 *   Int16   bit
01111 *
01112 * @param address: Int16
01113 * @param bit_offset:   bit 0 ~ 15
01114 * @param value:   true 1 false 0
01115 * @return 0
01116 *
01117 * @par Python
01118 * setInt16RegisterBit(self: pyaubo_sdk.RegisterControl, address: int,
01119 * bit_offset: int, value: bool) -> int
01120 *
01121 * @par Lua
01122 * setInt16RegisterBit(address: number, bit_offset: number, value: boolean)
01123 * -> number
01124 *
01125 * @par JSON-RPC
01126 * {"jsonrpc":"2.0","method":"RegisterControl.setInt16RegisterBit","params":[1,

```



```

01127     * 5, true], "id": 1}
01128     *
01129     * @par JSON-RPC
01130     * {"id": 1, "jsonrpc": "2.0", "result": 0}
01131     *
01132     * \endchinese
01133     * \english
01134     * Set the status of a specific bit in an Int16 register
01135     *
01136     * @param address Int16 register address
01137     * @param bit_offset Bit offset in the register, 0 ~ 15
01138     * @param value Value to set, true to set to 1, false to clear to 0
01139     * @return Returns 0 on success, error code on failure
01140     *
01141     * @par Python interface prototype
01142     * setInt16RegisterBit(self: pyaubo_sdk.RegisterControl, address: int,
01143     * bit_offset: int, value: bool) -> int
01144     *
01145     * @par Lua interface prototype
01146     * setInt16RegisterBit(address: number, bit_offset: number, value: boolean)
01147     * -> number
01148     *
01149     * @par JSON-RPC request example
01150     * {"jsonrpc": "2.0", "method": "RegisterControl.setInt16RegisterBit", "params": [1,
01151     * 5, true], "id": 1}
01152     *
01153     * @par JSON-RPC response example
01154     * {"id": 1, "jsonrpc": "2.0", "result": 0}
01155     *
01156     * \endenglish
01157     */
01158 int setInt16RegisterBit(uint32_t address, uint8_t bit_offset, bool value);
01159
01160 /**
01161     * @ingroup RegisterControl
01162     * \chinese
01163     *
01164     * @param key
01165     * @return
01166     *
01167     * @par Lua
01168     * hasNamedVariable(key: string) -> boolean
01169     *
01170     * @par Lua
01171     * NamedVariable = hasNamedVariable("custom")
01172     *
01173     * @par JSON-RPC
01174     * {"jsonrpc": "2.0", "method": "RegisterControl.hasNamedVariable", "params": ["custom"], "id": 1}
01175     *
01176     * @par JSON-RPC
01177     * {"id": 1, "jsonrpc": "2.0", "result": false}
01178     *
01179     * \endchinese
01180     * \english
01181     * Whether the named variable exists
01182     *
01183     * @param key Variable name
01184     * @return
01185     *
01186     * @par Lua interface prototype
01187     * hasNamedVariable(key: string) -> boolean
01188     *
01189     * @par Lua example
01190     * NamedVariable = hasNamedVariable("custom")
01191     *
01192     * @par JSON-RPC request example
01193     * {"jsonrpc": "2.0", "method": "RegisterControl.hasNamedVariable", "params": ["custom"], "id": 1}
01194     *
01195     * @par JSON-RPC response example
01196     * {"id": 1, "jsonrpc": "2.0", "result": false}
01197     *
01198     * \endenglish
01199     */
01200
01201 bool hasNamedVariable(const std::string &key);
01202
01203 /**
01204     * @ingroup RegisterControl
01205     * \chinese
01206     *
01207     * @param key
01208     * @return
01209     *
01210     * @par Lua
01211     * getNamedVariableType(key: string) -> string
01212     *
01213     *

```

```

01214 * @par Lua
01215 * NamedVariableType = getNamedVariableType("custom")
01216 *
01217 * @par JSON-RPC
01218 * {"jsonrpc":"2.0","method":"RegisterControl.getNamedVariableType","params":["custom"],"id":1}
01219 *
01220 * @par JSON-RPC
01221 * {"id":1,"jsonrpc":"2.0","result":"NONE"}
01222 *
01223 * \endchinese
01224 * \english
01225 * Get the type of a named variable
01226 *
01227 * @param key
01228 * @return
01229 *
01230 * @par Lua interface prototype
01231 * getNamedVariableType(key: string) -> string
01232 *
01233 * @par Lua example
01234 * NamedVariableType = getNamedVariableType("custom")
01235 *
01236 * @par JSON-RPC request example
01237 * {"jsonrpc":"2.0","method":"RegisterControl.getNamedVariableType","params":["custom"],"id":1}
01238 *
01239 * @par JSON-RPC response example
01240 * {"id":1,"jsonrpc":"2.0","result":"NONE"}
01241 *
01242 * \endenglish
01243 */
01244 std::string getNamedVariableType(const std::string &key);
01245
01246 /**
01247 * @ingroup RegisterControl
01248 * \chinese
01249 *
01250 *
01251 * @param key
01252 * @param since
01253 * @return
01254 *
01255 * @par Python
01256 * variableUpdated(self: pyaubo_sdk.RegisterControl, arg0: str, arg1: int)
01257 * -> bool
01258 *
01259 * @par Lua
01260 * variableUpdated(key: string, since: number) -> boolean
01261 *
01262 * @par Lua
01263 * Variable_Updated = variableUpdated("custom" , 0)
01264 *
01265 * \endchinese
01266 *
01267 * \english
01268 * Whether the named variable has been updated
01269 *
01270 * @param key
01271 * @param since
01272 * @return
01273 *
01274 * @par Python interface prototype
01275 * variableUpdated(self: pyaubo_sdk.RegisterControl, arg0: str, arg1: int)
01276 * -> bool
01277 *
01278 * @par Lua interface prototype
01279 * variableUpdated(key: string, since: number) -> boolean
01280 *
01281 * @par Lua example
01282 * Variable_Updated = variableUpdated("custom" , 0)
01283 *
01284 * \endchinese
01285 */
01286 bool variableUpdated(const std::string &key, uint64_t since);
01287
01288 /**
01289 * @ingroup RegisterControl
01290 * \chinese
01291 *
01292 *
01293 * @param key
01294 * @param default_value
01295 * @return
01296 *
01297 * @par Python
01298 * getBool(self: pyaubo_sdk.RegisterControl, arg0: str, arg1: bool) -> bool
01299 *
01300 * @par Lua

```

```

01301 * getBool(key: string, default_value: boolean) -> boolean
01302 *
01303 * @par Lua
01304 * Bool_var = getBool("custom",false)
01305 *
01306 * @par JSON-RPC
01307 * {"jsonrpc":"2.0","method":"RegisterControl.getBool","params":["custom",false],"id":1}
01308 *
01309 * @par JSON-RPC
01310 * {"id":1,"jsonrpc":"2.0","result":true}
01311 *
01312 * \endchinese
01313 * \english
01314 * Get variable value
01315 *
01316 * @param key
01317 * @param default_value
01318 * @return
01319 *
01320 * @par Python interface prototype
01321 * getBool(self: pyaubo_sdk.RegisterControl, arg0: str, arg1: bool) -> bool
01322 *
01323 * @par Lua interface prototype
01324 * getBool(key: string, default_value: boolean) -> boolean
01325 *
01326 * @par Lua example
01327 * Bool_var = getBool("custom",false)
01328 *
01329 * @par JSON-RPC request example
01330 * {"jsonrpc":"2.0","method":"RegisterControl.getBool","params":["custom",false],"id":1}
01331 *
01332 * @par JSON-RPC response example
01333 * {"id":1,"jsonrpc":"2.0","result":true}
01334 *
01335 * \endenglish
01336 */
01337 bool getBool(const std::string &key, bool default_value);
01338
01339 /**
01340 * @ingroup RegisterControl
01341 * \chinese
01342 * /
01343 *
01344 * @param key
01345 * @param value
01346 * @return
01347 *
01348 * @par Python
01349 * setBool(self: pyaubo_sdk.RegisterControl, arg0: str, arg1: bool) -> int
01350 *
01351 * @par Lua
01352 * setBool(key: string, value: boolean) -> nil
01353 *
01354 * @par Lua
01355 * setBool("custom",true)
01356 *
01357 * @par JSON-RPC
01358 * {"jsonrpc":"2.0","method":"RegisterControl.setBool","params":["custom",true],"id":1}
01359 *
01360 * @par JSON-RPC
01361 * {"id":1,"jsonrpc":"2.0","result":0}
01362 *
01363 * \endchinese
01364 * \english
01365 * Set or update the variable value
01366 *
01367 * @param key
01368 * @param value
01369 * @return Returns 0 on success, otherwise error code
01370 *
01371 * @par Python interface prototype
01372 * setBool(self: pyaubo_sdk.RegisterControl, arg0: str, arg1: bool) -> int
01373 *
01374 * @par Lua interface prototype
01375 * setBool(key: string, value: boolean) -> nil
01376 *
01377 * @par Lua example
01378 * setBool("custom",true)
01379 *
01380 * @par JSON-RPC request example
01381 * {"jsonrpc":"2.0","method":"RegisterControl.setBool","params":["custom",true],"id":1}
01382 *
01383 * @par JSON-RPC response example
01384 * {"id":1,"jsonrpc":"2.0","result":0}
01385 *
01386 * \endenglish
01387 */

```

```

01388 int setBool(const std::string &key, bool value);
01389
01390 /**
01391  * @ingroup RegisterControl
01392  * \chinese
01393  *
01394  *
01395  * @param key
01396  * @param default_value
01397  * @return
01398  *
01399  * @par Python
01400  * getVecChar(self: pyaubo_sdk.RegisterControl, arg0: str, arg1: List[str])
01401  * -> List[str]
01402  *
01403  * @par Lua
01404  * getVecChar(key: string, default_value: table) -> table
01405  *
01406  * @par Lua
01407  * VecChar = getVecChar("custom",{})
01408  *
01409  * @par JSON-RPC
01410  * {"jsonrpc":"2.0","method":"RegisterControl.getVecChar","params":["custom",[]],"id":1}
01411  *
01412  * @par JSON-RPC
01413  * {"id":1,"jsonrpc":"2.0","result":[0,1,0]}
01414  *
01415  * \endchinese
01416  * \english
01417  * Get variable value
01418  *
01419  * @param key
01420  * @param default_value
01421  * @return
01422  *
01423  * @par Python interface prototype
01424  * getVecChar(self: pyaubo_sdk.RegisterControl, arg0: str, arg1: List[str])
01425  * -> List[str]
01426  *
01427  * @par Lua interface prototype
01428  * getVecChar(key: string, default_value: table) -> table
01429  *
01430  * @par Lua example
01431  * VecChar = getVecChar("custom",{})
01432  *
01433  * @par JSON-RPC request example
01434  * {"jsonrpc":"2.0","method":"RegisterControl.getVecChar","params":["custom",[]],"id":1}
01435  *
01436  * @par JSON-RPC response example
01437  * {"id":1,"jsonrpc":"2.0","result":[0,1,0]}
01438  *
01439  * \endenglish
01440  */
01441 std::vector<char> getVecChar(const std::string &key,
01442                             const std::vector<char> &default_value);
01443
01444 /**
01445  * @ingroup RegisterControl
01446  * \chinese
01447  * /
01448  *
01449  * @param key
01450  * @param value
01451  * @return
01452  *
01453  * @par Python
01454  * setVecChar(self: pyaubo_sdk.RegisterControl, arg0: str, arg1: List[str])
01455  * -> int
01456  *
01457  * @par Lua
01458  * setVecChar(key: string, value: table) -> nil
01459  *
01460  * @par Lua
01461  * setVecChar("custom",{0,1,0})
01462  *
01463  * @par JSON-RPC
01464  * {"jsonrpc":"2.0","method":"RegisterControl.setVecChar","params":["custom",[0,1,0]],"id":1}
01465  *
01466  * @par JSON-RPC
01467  * {"id":1,"jsonrpc":"2.0","result":0}
01468  *
01469  * \endchinese
01470  * \english
01471  * Set or update the variable value
01472  *
01473  * @param key
01474  * @param value

```

```

01475 * @return Returns 0 on success, otherwise error code
01476 *
01477 * @par Python interface prototype
01478 * setVecChar(self: pyaubo_sdk.RegisterControl, arg0: str, arg1: List[str])
01479 * -> int
01480 *
01481 * @par Lua interface prototype
01482 * setVecChar(key: string, value: table) -> nil
01483 *
01484 * @par Lua example
01485 * setVecChar("custom",{0,1,0})
01486 *
01487 * @par JSON-RPC request example
01488 * {"jsonrpc":"2.0","method":"RegisterControl.setVecChar","params":["custom",[0,1,0]],"id":1}
01489 *
01490 * @par JSON-RPC response example
01491 * {"id":1,"jsonrpc":"2.0","result":0}
01492 *
01493 * \endenglish
01494 */
01495 int setVecChar(const std::string &key, const std::vector<char> &value);
01496
01497 /**
01498 * @ingroup RegisterControl
01499 * \chinese
01500 *
01501 *
01502 * @param key
01503 * @param default_value
01504 * @return
01505 *
01506 * @par Python
01507 * getInt32(self: pyaubo_sdk.RegisterControl, arg0: str, arg1: int) -> int
01508 *
01509 * @par Lua
01510 * getInt32(key: string, default_value: number) -> number
01511 *
01512 * @par Lua
01513 * Int32 = getInt32("custom",0)
01514 *
01515 * @par JSON-RPC
01516 * {"jsonrpc":"2.0","method":"RegisterControl.getInt32","params":["custom",0],"id":1}
01517 *
01518 * @par JSON-RPC
01519 * {"id":1,"jsonrpc":"2.0","result":6}
01520 *
01521 * \endchinese
01522 * \english
01523 * Get variable value
01524 *
01525 * @param key
01526 * @param default_value
01527 * @return
01528 *
01529 * @par Python interface prototype
01530 * getInt32(self: pyaubo_sdk.RegisterControl, arg0: str, arg1: int) -> int
01531 *
01532 * @par Lua interface prototype
01533 * getInt32(key: string, default_value: number) -> number
01534 *
01535 * @par Lua example
01536 * Int32 = getInt32("custom",0)
01537 *
01538 * @par JSON-RPC request example
01539 * {"jsonrpc":"2.0","method":"RegisterControl.getInt32","params":["custom",0],"id":1}
01540 *
01541 * @par JSON-RPC response example
01542 * {"id":1,"jsonrpc":"2.0","result":6}
01543 *
01544 * \endenglish
01545 */
01546 int getInt32(const std::string &key, int default_value);
01547
01548 /**
01549 * @ingroup RegisterControl
01550 * \chinese
01551 * /
01552 *
01553 * @param key
01554 * @param value
01555 * @return
01556 *
01557 * @par Python
01558 * setInt32(self: pyaubo_sdk.RegisterControl, arg0: str, arg1: int) -> int
01559 *
01560 * @par Lua
01561 * setInt32(key: string, value: number) -> nil

```

```

01562 *
01563 * @par Lua
01564 * setInt32("custom",6)
01565 *
01566 * @par JSON-RPC
01567 * {"jsonrpc":"2.0","method":"RegisterControl.setInt32","params":["custom",6],"id":1}
01568 *
01569 * @par JSON-RPC
01570 * {"id":1,"jsonrpc":"2.0","result":0}
01571 *
01572 * \endchinese
01573 * \english
01574 * Set or update the variable value
01575 *
01576 * @param key
01577 * @param value
01578 * @return
01579 *
01580 * @par Python interface prototype
01581 * setInt32(self: pyaubo_sdk.RegisterControl, arg0: str, arg1: int) -> int
01582 *
01583 * @par Lua interface prototype
01584 * setInt32(key: string, value: number) -> nil
01585 *
01586 * @par Lua example
01587 * setInt32("custom",6)
01588 *
01589 * @par JSON-RPC request example
01590 * {"jsonrpc":"2.0","method":"RegisterControl.setInt32","params":["custom",6],"id":1}
01591 *
01592 * @par JSON-RPC response example
01593 * {"id":1,"jsonrpc":"2.0","result":0}
01594 *
01595 * \endenglish
01596 */
01597 int setInt32(const std::string &key, int value);
01598
01599 /**
01600 * @ingroup RegisterControl
01601 * \chinese
01602 *
01603 *
01604 * @param key
01605 * @param default_value
01606 * @return
01607 *
01608 * @par Python
01609 * getVecInt32(self: pyaubo_sdk.RegisterControl, arg0: str, arg1: List[int])
01610 * -> List[int]
01611 *
01612 * @par Lua
01613 * getVecInt32(key: string, default_value: table) -> table
01614 *
01615 * @par Lua
01616 * VecInt32 = getVecInt32("custom",{})
01617 *
01618 * @par JSON-RPC
01619 * {"jsonrpc":"2.0","method":"RegisterControl.getVecInt32","params":["custom",[]],"id":1}
01620 *
01621 * @par JSON-RPC
01622 * {"id":1,"jsonrpc":"2.0","result":[1,2,3,4]}
01623 *
01624 * \endchinese
01625 * \english
01626 * Get variable value
01627 *
01628 * @param key
01629 * @param default_value
01630 * @return
01631 *
01632 * @par Python interface prototype
01633 * getVecInt32(self: pyaubo_sdk.RegisterControl, arg0: str, arg1: List[int])
01634 * -> List[int]
01635 *
01636 * @par Lua interface prototype
01637 * getVecInt32(key: string, default_value: table) -> table
01638 *
01639 * @par Lua example
01640 * VecInt32 = getVecInt32("custom",{})
01641 *
01642 * @par JSON-RPC request example
01643 * {"jsonrpc":"2.0","method":"RegisterControl.getVecInt32","params":["custom",[]],"id":1}
01644 *
01645 * @par JSON-RPC response example
01646 * {"id":1,"jsonrpc":"2.0","result":[1,2,3,4]}
01647 *
01648 * \endenglish

```

```

01649     */
01650     std::vector<int32_t> getVecInt32(const std::string &key,
01651                                   const std::vector<int32_t> &default_value);
01652
01653     /**
01654     * @ingroup RegisterControl
01655     * \chinese
01656     * /
01657     *
01658     * @param key
01659     * @param value
01660     * @return
01661     *
01662     * @par Python
01663     * setVecInt32(self: pyaubo_sdk.RegisterControl, arg0: str, arg1: List[int])
01664     * -> int
01665     *
01666     * @par Lua
01667     * setVecInt32(key: string, value: table) -> nil
01668     *
01669     * @par Lua
01670     * setVecInt32("custom",{1,2,3,4})
01671     *
01672     * @par JSON-RPC
01673     * {"jsonrpc":"2.0","method":"RegisterControl.setVecInt32","params":["custom",[1,2,3,4]],"id":1}
01674     *
01675     * @par JSON-RPC
01676     * {"id":1,"jsonrpc":"2.0","result":0}
01677     *
01678     * \endchinese
01679     * \english
01680     * Set or update the variable value
01681     *
01682     * @param key
01683     * @param value
01684     * @return
01685     *
01686     * @par Python interface prototype
01687     * setVecInt32(self: pyaubo_sdk.RegisterControl, arg0: str, arg1: List[int])
01688     * -> int
01689     *
01690     * @par Lua interface prototype
01691     * setVecInt32(key: string, value: table) -> nil
01692     *
01693     * @par Lua example
01694     * setVecInt32("custom",{1,2,3,4})
01695     *
01696     * @par JSON-RPC request example
01697     * {"jsonrpc":"2.0","method":"RegisterControl.setVecInt32","params":["custom",[1,2,3,4]],"id":1}
01698     *
01699     * @par JSON-RPC response example
01700     * {"id":1,"jsonrpc":"2.0","result":0}
01701     *
01702     * \endenglish
01703     */
01704     int setVecInt32(const std::string &key, const std::vector<int32_t> &value);
01705
01706     /**
01707     * @ingroup RegisterControl
01708     * \chinese
01709     *
01710     *
01711     * @param key
01712     * @param default_value
01713     * @return
01714     *
01715     * @par Python
01716     * getFloat(self: pyaubo_sdk.RegisterControl, arg0: str, arg1: float) ->
01717     * float
01718     *
01719     * @par Lua
01720     * getFloat(key: string, default_value: number) -> number
01721     *
01722     * @par Lua
01723     * var_Float = getFloat("custom",0.0)
01724     *
01725     * @par JSON-RPC
01726     * {"jsonrpc":"2.0","method":"RegisterControl.getFloat","params":["custom",0.0],"id":1}
01727     *
01728     * @par JSON-RPC
01729     * {"id":1,"jsonrpc":"2.0","result":4.400000095367432}
01730     *
01731     * \endchinese
01732     * \english
01733     * Get variable value
01734     *
01735     * @param key

```

```

01736 * @param default_value
01737 * @return
01738 *
01739 * @par Python interface prototype
01740 * getFloat(self: pyaubo_sdk.RegisterControl, arg0: str, arg1: float) ->
01741 * float
01742 *
01743 * @par Lua interface prototype
01744 * getFloat(key: string, default_value: number) -> number
01745 *
01746 * @par Lua example
01747 * var_Float = getFloat("custom",0.0)
01748 *
01749 * @par JSON-RPC request example
01750 * {"jsonrpc":"2.0","method":"RegisterControl.getFloat","params":["custom",0.0],"id":1}
01751 *
01752 * @par JSON-RPC response example
01753 * {"id":1,"jsonrpc":"2.0","result":4.400000095367432}
01754 *
01755 * \endenglish
01756 */
01757 float getFloat(const std::string &key, float default_value);
01758
01759 /**
01760 * @ingroup RegisterControl
01761 * \chinese
01762 * /
01763 *
01764 * @param key
01765 * @param value
01766 * @return
01767 *
01768 * @par Python
01769 * setFloat(self: pyaubo_sdk.RegisterControl, arg0: str, arg1: float) -> int
01770 *
01771 * @par Lua
01772 * setFloat(key: string, value: number) -> nil
01773 *
01774 * @par Lua
01775 * setFloat("custom",4.4)
01776 *
01777 * @par JSON-RPC
01778 * {"jsonrpc":"2.0","method":"RegisterControl.setFloat","params":["custom",4.4],"id":1}
01779 *
01780 * @par JSON-RPC
01781 * {"id":1,"jsonrpc":"2.0","result":0}
01782 *
01783 * \endchinese
01784 * \english
01785 * Set or update the variable value
01786 *
01787 * @param key
01788 * @param value
01789 * @return
01790 *
01791 * @par Python interface prototype
01792 * setFloat(self: pyaubo_sdk.RegisterControl, arg0: str, arg1: float) -> int
01793 *
01794 * @par Lua interface prototype
01795 * setFloat(key: string, value: number) -> nil
01796 *
01797 * @par Lua example
01798 * setFloat("custom",4.4)
01799 *
01800 * @par JSON-RPC request example
01801 * {"jsonrpc":"2.0","method":"RegisterControl.setFloat","params":["custom",4.4],"id":1}
01802 *
01803 * @par JSON-RPC response example
01804 * {"id":1,"jsonrpc":"2.0","result":0}
01805 *
01806 * \endenglish
01807 */
01808 int setFloat(const std::string &key, float value);
01809
01810 /**
01811 * @ingroup RegisterControl
01812 * \chinese
01813 *
01814 *
01815 * @param key
01816 * @param default_value
01817 * @return
01818 *
01819 * @par Python
01820 * getVecFloat(self: pyaubo_sdk.RegisterControl, arg0: str, arg1:
01821 * List[float]) -> List[float]
01822 *

```



```

01823 * @par Lua
01824 * getVecFloat(key: string, default_value: table) -> table
01825 *
01826 * @par Lua
01827 * VecFloat = getVecFloat("custom",{})
01828 *
01829 * @par JSON-RPC
01830 * {"jsonrpc":"2.0","method":"RegisterControl.getVecFloat","params":["custom",[]],"id":1}
01831 *
01832 * @par JSON-RPC
01833 * {"id":1,"jsonrpc":"2.0","result":[0.0,0.10000000149011612,3.299999952316284]}
01834 *
01835 * \endchinese
01836 * \english
01837 * Get variable value
01838 *
01839 * @param key
01840 * @param default_value
01841 * @return
01842 *
01843 * @par Python interface prototype
01844 * getVecFloat(self: pyaubo_sdk.RegisterControl, arg0: str, arg1:
01845 * List[float]) -> List[float]
01846 *
01847 * @par Lua interface prototype
01848 * getVecFloat(key: string, default_value: table) -> table
01849 *
01850 * @par Lua example
01851 * VecFloat = getVecFloat("custom",{})
01852 *
01853 * @par JSON-RPC request example
01854 * {"jsonrpc":"2.0","method":"RegisterControl.getVecFloat","params":["custom",[]],"id":1}
01855 *
01856 * @par JSON-RPC response example
01857 * {"id":1,"jsonrpc":"2.0","result":[0.0,0.10000000149011612,3.299999952316284]}
01858 *
01859 * \endenglish
01860 */
01861 std::vector<float> getVecFloat(const std::string &key,
01862                               const std::vector<float> &default_value);
01863
01864 /**
01865 * @ingroup RegisterControl
01866 * \chinese
01867 * /
01868 *
01869 * @param key
01870 * @param value
01871 * @return
01872 *
01873 * @par Python
01874 * setVecFloat(self: pyaubo_sdk.RegisterControl, arg0: str, arg1:
01875 * List[float]) -> int
01876 *
01877 * @par Lua
01878 * setVecFloat(key: string, value: table) -> nil
01879 *
01880 * @par Lua
01881 * setVecFloat("custom", {0.0,0.1,3.3})
01882 *
01883 * @par JSON-RPC
01884 * {"jsonrpc":"2.0","method":"RegisterControl.setVecFloat","params":["custom",[0.0,0.1,3.3]],"id":1}
01885 *
01886 * @par JSON-RPC
01887 * {"id":1,"jsonrpc":"2.0","result":0}
01888 *
01889 * \endchinese
01890 * \english
01891 * Set or update the variable value
01892 *
01893 * @param key
01894 * @param value
01895 * @return
01896 *
01897 * @par Python interface prototype
01898 * setVecFloat(self: pyaubo_sdk.RegisterControl, arg0: str, arg1:
01899 * List[float]) -> int
01900 *
01901 * @par Lua interface prototype
01902 * setVecFloat(key: string, value: table) -> nil
01903 *
01904 * @par Lua example
01905 * setVecFloat("custom", {0.0,0.1,3.3})
01906 *
01907 * @par JSON-RPC request example
01908 * {"jsonrpc":"2.0","method":"RegisterControl.setVecFloat","params":["custom",[0.0,0.1,3.3]],"id":1}
01909 *

```

```

01910 * @par JSON-RPC response example
01911 * {"id":1,"jsonrpc":"2.0","result":0}
01912 *
01913 * \endenglish
01914 */
01915 int setVecFloat(const std::string &key, const std::vector<float> &value);
01916
01917 /**
01918 * @ingroup RegisterControl
01919 * \chinese
01920 *
01921 *
01922 * @param key
01923 * @param default_value
01924 * @return
01925 *
01926 * @par Python
01927 * getDouble(self: pyaubo_sdk.RegisterControl, arg0: str, arg1: float) ->
01928 * float
01929 *
01930 * @par Lua
01931 * getDouble(key: string, default_value: number) -> number
01932 *
01933 * @par Lua
01934 * var_Double = getDouble("custom",0.0)
01935 *
01936 * @par JSON-RPC
01937 * {"jsonrpc":"2.0","method":"RegisterControl.getDouble","params":["custom",0.0],"id":1}
01938 *
01939 * @par JSON-RPC
01940 * {"id":1,"jsonrpc":"2.0","result":0.0}
01941 *
01942 * \endchinese
01943 * \english
01944 * Get variable value
01945 *
01946 * @param key
01947 * @param default_value
01948 * @return
01949 *
01950 * @par Python interface prototype
01951 * getDouble(self: pyaubo_sdk.RegisterControl, arg0: str, arg1: float) ->
01952 * float
01953 *
01954 * @par Lua interface prototype
01955 * getDouble(key: string, default_value: number) -> number
01956 *
01957 * @par Lua example
01958 * var_Double = getDouble("custom",0.0)
01959 *
01960 * @par JSON-RPC request example
01961 * {"jsonrpc":"2.0","method":"RegisterControl.getDouble","params":["custom",0.0],"id":1}
01962 *
01963 * @par JSON-RPC response example
01964 * {"id":1,"jsonrpc":"2.0","result":0.0}
01965 *
01966 * \endenglish
01967 */
01968 double getDouble(const std::string &key, double default_value);
01969
01970 /**
01971 * @ingroup RegisterControl
01972 * \chinese
01973 * /
01974 *
01975 * @param key
01976 * @param value
01977 * @return
01978 *
01979 * @par Python
01980 * setDouble(self: pyaubo_sdk.RegisterControl, arg0: str, arg1: float) ->
01981 * int
01982 *
01983 * @par Lua
01984 * setDouble(key: string, value: number) -> nil
01985 *
01986 * @par Lua
01987 * setDouble("custom",6.6)
01988 *
01989 * @par JSON-RPC
01990 * {"jsonrpc":"2.0","method":"RegisterControl.setDouble","params":["custom",6.6],"id":1}
01991 *
01992 * @par JSON-RPC
01993 * {"id":1,"jsonrpc":"2.0","result":0}
01994 *
01995 * \endchinese
01996 * \english

```

```

01997     * Set or update the variable value
01998     *
01999     * @param key
02000     * @param value
02001     * @return
02002     *
02003     * @par Python interface prototype
02004     * setDouble(self: pyaubo_sdk.RegisterControl, arg0: str, arg1: float) ->
02005     * int
02006     *
02007     * @par Lua interface prototype
02008     * setDouble(key: string, value: number) -> nil
02009     *
02010     * @par Lua example
02011     * setDouble("custom",6.6)
02012     *
02013     * @par JSON-RPC request example
02014     * {"jsonrpc":"2.0","method":"RegisterControl.setDouble","params":["custom",6.6],"id":1}
02015     *
02016     * @par JSON-RPC response example
02017     * {"id":1,"jsonrpc":"2.0","result":0}
02018     *
02019     * \endenglish
02020     */
02021 int setDouble(const std::string &key, double value);
02022
02023 /**
02024  * @ingroup RegisterControl
02025  * \chinese
02026  *
02027  *
02028  * @param key
02029  * @param default_value
02030  * @return
02031  *
02032  * @par Python
02033  * getVecDouble(self: pyaubo_sdk.RegisterControl, arg0: str, arg1:
02034  * List[float]) -> List[float]
02035  *
02036  * @par Lua
02037  * getVecDouble(key: string, default_value: table) -> table
02038  *
02039  * @par Lua
02040  * VecDouble = getVecDouble("custom",{})
02041  *
02042  * @par JSON-RPC
02043  * {"jsonrpc":"2.0","method":"RegisterControl.getVecDouble","params":["custom",[]],"id":1}
02044  *
02045  * @par JSON-RPC
02046  * {"id":1,"jsonrpc":"2.0","result":[0.1,0.2,0.3]}
02047  *
02048  * \endchinese
02049  * \english
02050  * Get variable value
02051  *
02052  * @param key
02053  * @param default_value
02054  * @return
02055  *
02056  * @par Python interface prototype
02057  * getVecDouble(self: pyaubo_sdk.RegisterControl, arg0: str, arg1:
02058  * List[float]) -> List[float]
02059  *
02060  * @par Lua interface prototype
02061  * getVecDouble(key: string, default_value: table) -> table
02062  *
02063  * @par Lua example
02064  * VecDouble = getVecDouble("custom",{})
02065  *
02066  * @par JSON-RPC request example
02067  * {"jsonrpc":"2.0","method":"RegisterControl.getVecDouble","params":["custom",[]],"id":1}
02068  *
02069  * @par JSON-RPC response example
02070  * {"id":1,"jsonrpc":"2.0","result":[0.1,0.2,0.3]}
02071  *
02072  * \endenglish
02073  */
02074 std::vector<double> getVecDouble(const std::string &key,
02075                                const std::vector<double> &default_value);
02076
02077 /**
02078  * @ingroup RegisterControl
02079  * \chinese
02080  * /
02081  *
02082  * @param key
02083  * @param value

```

```

02084 * @return
02085 *
02086 * @par Python
02087 * setVecDouble(self: pyaubo_sdk.RegisterControl, arg0: str, arg1:
02088 * List[float]) -> int
02089 *
02090 * @par Lua
02091 * setVecDouble(key: string, value: table) -> nil
02092 *
02093 * @par Lua
02094 * setVecDouble("custom",{0.1,0.2,0.3})
02095 *
02096 * @par JSON-RPC
02097 * {"jsonrpc":"2.0","method":"RegisterControl.setVecDouble","params":["custom",[0.1,0.2,0.3]],"id":1}
02098 *
02099 * @par JSON-RPC
02100 * {"id":1,"jsonrpc":"2.0","result":0}
02101 *
02102 * \endchinese
02103 * \english
02104 * Set or update the variable value
02105 *
02106 * @param key
02107 * @param value
02108 * @return
02109 *
02110 * @par Python interface prototype
02111 * setVecDouble(self: pyaubo_sdk.RegisterControl, arg0: str, arg1:
02112 * List[float]) -> int
02113 *
02114 * @par Lua interface prototype
02115 * setVecDouble(key: string, value: table) -> nil
02116 *
02117 * @par Lua example
02118 * setVecDouble("custom",{0.1,0.2,0.3})
02119 *
02120 * @par JSON-RPC request example
02121 * {"jsonrpc":"2.0","method":"RegisterControl.setVecDouble","params":["custom",[0.1,0.2,0.3]],"id":1}
02122 *
02123 * @par JSON-RPC response example
02124 * {"id":1,"jsonrpc":"2.0","result":0}
02125 *
02126 * \endenglish
02127 */
02128 int setVecDouble(const std::string &key, const std::vector<double> &value);
02129
02130 /**
02131 * @ingroup RegisterControl
02132 * \chinese
02133 *
02134 *
02135 * @param key
02136 * @param default_value
02137 * @return
02138 *
02139 * @par Python
02140 * getString(self: pyaubo_sdk.RegisterControl, arg0: str, arg1: str) -> str
02141 *
02142 * @par Lua
02143 * getString(key: string, default_value: string) -> string
02144 *
02145 * @par Lua
02146 * var_String = getString("custom","")
02147 *
02148 * @par JSON-RPC
02149 * {"jsonrpc":"2.0","method":"RegisterControl.getString","params":["custom",""],"id":1}
02150 *
02151 * @par JSON-RPC
02152 * {"id":1,"jsonrpc":"2.0","result":"test"}
02153 *
02154 * \endchinese
02155 * \english
02156 * Get variable value
02157 *
02158 * @param key
02159 * @param default_value
02160 * @return
02161 *
02162 * @par Python interface prototype
02163 * getString(self: pyaubo_sdk.RegisterControl, arg0: str, arg1: str) -> str
02164 *
02165 * @par Lua interface prototype
02166 * getString(key: string, default_value: string) -> string
02167 *
02168 * @par Lua example
02169 * var_String = getString("custom","")
02170 *

```

```

02171 * @par JSON-RPC request example
02172 * {"jsonrpc":"2.0","method":"RegisterControl.getString","params":["custom",""],"id":1}
02173 *
02174 * @par JSON-RPC response example
02175 * {"id":1,"jsonrpc":"2.0","result":"test"}
02176 *
02177 * \endenglish
02178 */
02179 std::string getString(const std::string &key,
02180                      const std::string &default_value);
02181
02182 /**
02183 * @ingroup RegisterControl
02184 * \chinese
02185 * /
02186 *
02187 * @param key
02188 * @param value
02189 * @return
02190 *
02191 * @par Python
02192 * setString(self: pyaubo_sdk.RegisterControl, arg0: str, arg1: str) -> int
02193 *
02194 * @par Lua
02195 * setString(key: string, value: string) -> nil
02196 *
02197 * @par Lua
02198 * setString("custom","test")
02199 *
02200 * @par JSON-RPC
02201 * {"jsonrpc":"2.0","method":"RegisterControl.setString","params":["custom","test"],"id":1}
02202 *
02203 * @par JSON-RPC
02204 * {"id":1,"jsonrpc":"2.0","result":0}
02205 *
02206 * \endchinese
02207 * \english
02208 * Set or update the variable value
02209 *
02210 * @param key
02211 * @param value
02212 * @return Returns 0 on success, otherwise error code
02213 *
02214 * @par Python interface prototype
02215 * setString(self: pyaubo_sdk.RegisterControl, arg0: str, arg1: str) -> int
02216 *
02217 * @par Lua interface prototype
02218 * setString(key: string, value: string) -> nil
02219 *
02220 * @par Lua example
02221 * setString("custom","test")
02222 *
02223 * @par JSON-RPC request example
02224 * {"jsonrpc":"2.0","method":"RegisterControl.setString","params":["custom","test"],"id":1}
02225 *
02226 * @par JSON-RPC response example
02227 * {"id":1,"jsonrpc":"2.0","result":0}
02228 *
02229 * \endenglish
02230 */
02231 int setString(const std::string &key, const std::string &value);
02232
02233 /**
02234 * @ingroup RegisterControl
02235 * \chinese
02236 *
02237 *
02238 * @param key
02239 * @return
02240 *
02241 * @par Python
02242 * clearNamedVariable(self: pyaubo_sdk.RegisterControl, arg0: str) -> int
02243 *
02244 * @par Lua
02245 * clearNamedVariable(key: string) -> nil
02246 *
02247 * @par Lua
02248 * clearNamedVariable("custom")
02249 *
02250 * @par JSON-RPC
02251 * {"jsonrpc":"2.0","method":"RegisterControl.clearNamedVariable","params":["custom"],"id":1}
02252 *
02253 * @par JSON-RPC
02254 * {"id":1,"jsonrpc":"2.0","result":1}
02255 *
02256 * \endchinese
02257 * \english

```

```

02258 * Clear variable
02259 *
02260 * @param key
02261 * @return
02262 *
02263 * @par Python interface prototype
02264 * clearNamedVariable(self: pyaubo_sdk.RegisterControl, arg0: str) -> int
02265 *
02266 * @par Lua interface prototype
02267 * clearNamedVariable(key: string) -> nil
02268 *
02269 * @par Lua example
02270 * clearNamedVariable("custom")
02271 *
02272 * @par JSON-RPC request example
02273 * {"jsonrpc": "2.0", "method": "RegisterControl.clearNamedVariable", "params": ["custom"], "id": 1}
02274 *
02275 * @par JSON-RPC response example
02276 * {"id": 1, "jsonrpc": "2.0", "result": 1}
02277 *
02278 * \endenglish
02279 */
02280 int clearNamedVariable(const std::string &key);
02281
02282 /**
02283 * @ingroup RegisterControl
02284 * \chinese
02285 *
02286 *
02287 *
02288 *
02289 * @param key
02290 * @param timeout (s) 0.1s
02291 * @param action
02292 * NONE (0):
02293 * PAUSE(1):
02294 * STOP (2): /
02295 * PROTECTIVE_STOP (3):
02296 * @return
02297 * \endchinese
02298 * \english
02299 * Set the watchdog
02300 *
02301 * After the watchdog is triggered, the controller will perform the
02302 * corresponding action and automatically delete the watchdog.
02303 *
02304 * @param key
02305 * @param timeout Timeout in seconds (s), minimum timeout is 0.1s
02306 * @param action
02307 * NONE (0): No action
02308 * PAUSE(1): Pause runtime
02309 * STOP (2): Stop runtime/stop robot motion
02310 * PROTECTIVE_STOP (3): Trigger protective stop
02311 * @return
02312 * \endenglish
02313 */
02314 int setWatchDog(const std::string &key, double timeout, int action);
02315
02316 /**
02317 * @ingroup RegisterControl
02318 * \chinese
02319 *
02320 *
02321 * @param key
02322 * @return
02323 * \endchinese
02324 * \english
02325 * Get the watchdog action
02326 *
02327 * @param key
02328 * @return
02329 * \endenglish
02330 */
02331 int getWatchDogAction(const std::string &key);
02332
02333 /**
02334 * @ingroup RegisterControl
02335 * \chinese
02336 *
02337 *
02338 * @param key
02339 * @return
02340 * \endchinese
02341 * \english
02342 * Get the watchdog timeout value
02343 *
02344 * @param key

```

```

02345 * @return
02346 * \endenglish
02347 */
02348 int getWatchDogTimeout(const std::string &key);
02349
02350 /**
02351 * @ingroup RegisterControl
02352 * \chinese
02353 * Modbus
02354 *
02355 * @param device_info
02356 * RTU "serial_port,baud,parity,data_bit,stop_bit"
02357 * (1)serial_port Linux "/dev/ttyS0" "/dev/ttyUSB0" Windows "\\.\COM10"
02358 * (2)baud 9600 19200 57600 115200
02359 * (3)parity N E O
02360 * (4)data_bit 5 6 7 8
02361 * (5)stop_bit 1 2
02362 *
02363 * TCP "ip address,port"
02364 * (1)ip address IP
02365 * (2)port
02366 * @param slave_number 255 0 255
02367 * @param signal_address
02368 * Modbus
02369 * @param signal_type 0 = 1 = 2 =
02370 * 3 =
02371 * @param signal_name
02372 * 20
02373 * @param sequential_mode
02374 * True Modbus fieldbus
02375 * @return
02376 *
02377 * @par Python
02378 * modbusAddSignal(self: pyaubo_sdk.RegisterControl, arg0: str, arg1: int,
02379 * arg2: int, arg3: int, arg4: str, arg5: bool) -> int
02380 *
02381 * @par Lua
02382 * modbusAddSignal(device_info: string, slave_number: number,
02383 * signal_address: number, signal_type: number, signal_name: string,
02384 * sequential_mode: boolean) -> nil
02385 *
02386 * @par Lua
02387 * modbusAddSignal("/dev/ttyRobotTool,115200,N,8,1", 1, signal_address:
02388 * number, 264, "Modbus_0", false)
02389 *
02390 * @par JSON-RPC
02391 * {"json-
rpc":"2.0","method":"RegisterControl.modbusAddSignal","params":["/dev/ttyRobotTool,115200,N,8,1",1,264,3,"Modbus_0",false],"id":1}
02392 *
02393 * @par JSON-RPC
02394 * {"id":1,"jsonrpc":"2.0","result":0}
02395 * \endchinese
02396 * \english
02397 * Adds a new modbus signal for the controller to supervise. Expects no
02398 * response.
02399 *
02400 * @param device_info is rtu format.
02401 * eg,"serial_port,baud,parity,data_bit,stop_bit"
02402 * (1)The serial_port argument specifies the name of the serial port eg. On
02403 * Linux,"/dev/ttyS0" or "/dev/ttyUSB0". On Windows, "\\.\COM10". (2)The
02404 * baud argument specifies the baud rate of the communication, eg. 9600,
02405 * 19200, 57600, 115200, etc. (3)parity:N for none,E for even,O for odd.
02406 * (4)data_bit:The data_bits argument specifies the number of bits of data,
02407 * the allowed values are 5, 6, 7 and 8. (5)stop_bit:The stop_bits argument
02408 * specifies the bits of stop, the allowed values are 1 and 2.
02409 *
02410 * device_info is tcp format, eg,"ip address,port"
02411 * (1)The ip address parameter specifies the ip address of the server
02412 * (2)The port parameter specifies the port number that the server is
02413 * listening on.
02414 * @param slave_number An integer normally not used and set to 255, but is a
02415 * free choice between 0 and 255.
02416 * @param signal_address An integer specifying the address of the either the
02417 * coil or the register that this new signal should reflect. Consult the
02418 * configuration of the modbus unit for this information.
02419 * @param signal_type An integer specifying the type of signal to add. 0 =
02420 * digital input, 1 = digital output, 2 = register input and 3 = register
02421 * output.
02422 * @param signal_name A string uniquely identifying the signal. If a string
02423 * is supplied which is equal to an already added signal, the new signal
02424 * will replace the old one. The length of the string cannot exceed 20
02425 * characters.
02426 * @param sequential_mode Setting to True forces the modbus client to wait
02427 * for a response before sending the next request. This mode is required by
02428 * some fieldbus units (Optional).
02429 * @return
02430 *

```

```

02431 * @par Python interface prototype
02432 * modbusAddSignal(self: pyaubo_sdk.RegisterControl, arg0: str, arg1: int,
02433 * arg2: int, arg3: int, arg4: str, arg5: bool) -> int
02434 *
02435 * @par Lua interface prototype
02436 * modbusAddSignal(device_info: string, slave_number: number,
02437 * signal_address: number, signal_type: number, signal_name: string,
02438 * sequential_mode: boolean) -> nil
02439 *
02440 * @par Lua example
02441 * modbusAddSignal("/dev/ttyRobotTool,115200,N,8,1", 1, signal_address:
02442 * number, 264, "Modbus_0", false)
02443 *
02444 * @par JSON-RPC request example
02445 * {"json-
rpc":"2.0","method":"RegisterControl.modbusAddSignal","params":["/dev/ttyRobotTool,115200,N,8,1",1,264,3,"Modbus_0",false],"id":1}
02446 *
02447 * @par JSON-RPC response example
02448 * {"id":1,"jsonrpc":"2.0","result":0}
02449 * \endenglish
02450 */
02451 int modbusAddSignal(const std::string &device_info, int slave_number,
02452 int signal_address, int signal_type,
02453 const std::string &signal_name, bool sequential_mode);
02454
02455 /**
02456 * @ingroup RegisterControl
02457 * \chinese
02458 *
02459 *
02460 * @param signal_name
02461 * @return
02462 *
02463 * @par Python
02464 * modbusDeleteSignal(self: pyaubo_sdk.RegisterControl, arg0: str) -> int
02465 *
02466 * @par Lua
02467 * modbusDeleteSignal(signal_name: string) -> nil
02468 *
02469 * @par Lua
02470 * modbusDeleteSignal("Modbus_1")
02471 *
02472 * @par JSON-RPC
02473 * {"jsonrpc":"2.0","method":"RegisterControl.modbusDeleteSignal","params":["Modbus_1"],"id":1}
02474 *
02475 * @par JSON-RPC
02476 * {"id":1,"jsonrpc":"2.0","result":0}
02477 * \endchinese
02478 * \english
02479 * Deletes the signal identified by the supplied signal name.
02480 *
02481 * @param signal_name A string equal to the name of the signal that should
02482 * be deleted.
02483 * @return
02484 *
02485 * @par Python interface prototype
02486 * modbusDeleteSignal(self: pyaubo_sdk.RegisterControl, arg0: str) -> int
02487 *
02488 * @par Lua interface prototype
02489 * modbusDeleteSignal(signal_name: string) -> nil
02490 *
02491 * @par Lua example
02492 * modbusDeleteSignal("Modbus_1")
02493 *
02494 * @par JSON-RPC request example
02495 * {"jsonrpc":"2.0","method":"RegisterControl.modbusDeleteSignal","params":["Modbus_1"],"id":1}
02496 *
02497 * @par JSON-RPC response example
02498 * {"id":1,"jsonrpc":"2.0","result":0}
02499 * \endenglish
02500 */
02501 int modbusDeleteSignal(const std::string &signal_name);
02502
02503 /**
02504 * @ingroup RegisterControl
02505 * \chinese
02506 * modbus
02507 *
02508 * @return
02509 *
02510 * @par JSON-RPC
02511 * {"jsonrpc":"2.0","method":"RegisterControl.modbusDeleteAllSignals","params":[],"id":1}
02512 *
02513 * @par JSON-RPC
02514 * {"id":1,"jsonrpc":"2.0","result":0}
02515 *
02516 * \endchinese

```



```

02517 * \english
02518 * Delete all modbus signals
02519 *
02520 * @return
02521 *
02522 * @par JSON-RPC request example
02523 * {"jsonrpc":"2.0","method":"RegisterControl.modbusDeleteAllSignals","params":[],"id":1}
02524 *
02525 * @par JSON-RPC response example
02526 * {"id":1,"jsonrpc":"2.0","result":0}
02527 *
02528 * \endenglish
02529 */
02530 int modbusDeleteAllSignals();
02531
02532 /**
02533 * @ingroup RegisterControl
02534 * \chinese
02535 *
02536 *
02537 * @param signal_name
02538 * @return 1 0
02539 *         -1
02540 *
02541 * @par Python
02542 * modbusGetSignalStatus(self: pyaubo_sdk.RegisterControl, arg0: str) -> int
02543 *
02544 * @par Lua
02545 * modbusGetSignalStatus(signal_name: string) -> number
02546 *
02547 * @par Lua
02548 * var0 = modbusGetSignalStatus("Modbus_0")
02549 *
02550 * @par JSON-RPC
02551 * {"jsonrpc":"2.0","method":"RegisterControl.modbusGetSignalStatus","params":["Modbus_0"],"id":1}
02552 *
02553 * @par JSON-RPC
02554 * {"id":1,"jsonrpc":"2.0","result":1}
02555 * \endchinese
02556 * \english
02557 * Reads the current value of a specific signal.
02558 *
02559 * @param signal_name A string equal to the name of the signal for which the
02560 * value should be gotten.
02561 * @return An integer or a boolean. For digital signals: 1 or 0. For
02562 * register signals: The register value expressed as an integer. If the
02563 * value is -1, it means the signal does not exist.
02564 *
02565 * @par Python interface prototype
02566 * modbusGetSignalStatus(self: pyaubo_sdk.RegisterControl, arg0: str) -> int
02567 *
02568 * @par Lua interface prototype
02569 * modbusGetSignalStatus(signal_name: string) -> number
02570 *
02571 * @par Lua example
02572 * var0 = modbusGetSignalStatus("Modbus_0")
02573 *
02574 * @par JSON-RPC request example
02575 * {"jsonrpc":"2.0","method":"RegisterControl.modbusGetSignalStatus","params":["Modbus_0"],"id":1}
02576 *
02577 * @par JSON-RPC response example
02578 * {"id":1,"jsonrpc":"2.0","result":1}
02579 * \endenglish
02580 */
02581 int modbusGetSignalStatus(const std::string &signal_name);
02582
02583 /**
02584 * @ingroup RegisterControl
02585 * \chinese
02586 *
02587 *
02588 * @return
02589 *
02590 * @par JSON-RPC
02591 * {"jsonrpc":"2.0","method":"RegisterControl.modbusGetSignalNames","params":[],"id":1}
02592 *
02593 * @par JSON-RPC
02594 * {"id":1,"jsonrpc":"2.0","result":["Modbus_0"]}
02595 *
02596 * \endchinese
02597 * \english
02598 * Get the collection of all signal names
02599 *
02600 * @return Collection of all signal names
02601 *
02602 * @par JSON-RPC request example
02603 * {"jsonrpc":"2.0","method":"RegisterControl.modbusGetSignalNames","params":[],"id":1}

```

```

02604      *
02605      * @par JSON-RPC response example
02606      * {"id":1,"jsonrpc":"2.0","result":["Modbus_0"]}
02607      *
02608      * \endenglish
02609      */
02610      std::vector<std::string> modbusGetSignalNames();
02611
02612      /**
02613      * @ingroup RegisterControl
02614      * \chinese
02615      *
02616      *
02617      * @return
02618      *
02619      * @par JSON-RPC
02620      * {"jsonrpc":"2.0","method":"RegisterControl.modbusGetSignalTypes","params":[],"id":1}
02621      *
02622      * @par JSON-RPC
02623      * {"id":1,"jsonrpc":"2.0","result":[1,0,2,3]}
02624      *
02625      * \endchinese
02626      * \english
02627      * Get the collection of all signal types
02628      *
02629      * @return Collection of all signal types
02630      *
02631      * @par JSON-RPC request example
02632      * {"jsonrpc":"2.0","method":"RegisterControl.modbusGetSignalTypes","params":[],"id":1}
02633      *
02634      * @par JSON-RPC response example
02635      * {"id":1,"jsonrpc":"2.0","result":[1,0,2,3]}
02636      *
02637      * \endenglish
02638      */
02639      std::vector<int> modbusGetSignalTypes();
02640
02641      /**
02642      * @ingroup RegisterControl
02643      * \chinese
02644      *
02645      *
02646      * @return
02647      *
02648      * @par JSON-RPC
02649      * {"jsonrpc":"2.0","method":"RegisterControl.modbusGetSignalValues","params":[],"id":1}
02650      *
02651      * @par JSON-RPC
02652      * {"id":1,"jsonrpc":"2.0","result":[1,1,88,33]}
02653      *
02654      * \endchinese
02655      * \english
02656      * Get the collection of all signal values
02657      *
02658      * @return Collection of all signal values
02659      *
02660      * @par JSON-RPC request example
02661      * {"jsonrpc":"2.0","method":"RegisterControl.modbusGetSignalValues","params":[],"id":1}
02662      *
02663      * @par JSON-RPC response example
02664      * {"id":1,"jsonrpc":"2.0","result":[1,1,88,33]}
02665      *
02666      * \endenglish
02667      */
02668      std::vector<int> modbusGetSignalValues();
02669
02670      /**
02671      * @ingroup RegisterControl
02672      * \chinese
02673      *      (0: , : )
02674      *
02675      * @return ModbusErrorNum
02676      *
02677      * @par JSON-RPC
02678      * {"jsonrpc":"2.0","method":"RegisterControl.modbusGetSignalErrors","params":[],"id":1}
02679      *
02680      * @par JSON-RPC
02681      * {"id":1,"jsonrpc":"2.0","result":[6,6,6,6]}
02682      *
02683      * \endchinese
02684      * \english
02685      * Get the error status of all signal requests (0: no error, others: error)
02686      * as a collection
02687      *
02688      * @return ModbusErrorNum
02689      *
02690      * @par JSON-RPC request example

```

```

02691 * {"jsonrpc":"2.0","method":"RegisterControl.modbusGetSignalErrors","params":[],"id":1}
02692 *
02693 * @par JSON-RPC response example
02694 * {"id":1,"jsonrpc":"2.0","result":[6,6,6,6]}
02695 *
02696 * \endenglish
02697 */
02698 std::vector<int> modbusGetSignalErrors();
02699
02700 /**
02701 * @ingroup RegisterControl
02702 * \chinese
02703 * IP Modbus
02704 *
02705 *
02706 * Modbus
02707 *
02708 * @param device_info
02709 * RTU "serial_port,baud,parity,data_bit,stop_bit"
02710 * (1)serial_port Linux "/dev/ttyS0" "/dev/ttyUSB0" Windows "\\.\COM10"
02711 * (2)baud 9600 19200 57600 115200
02712 * (3)parity N E O
02713 * (4)data_bit 5 6 7 8
02714 * (5)stop_bit 1 2
02715 *
02716 * TCP "ip address,port"
02717 * (1)ip address IP
02718 * (2)port
02719 * @param slave_number
02720 * @param function_code
02721 *
02722 * Modbus
02723 * MODBUS_FC_READ_COILS 0x01
02724 * MODBUS_FC_READ_DISCRETE_INPUTS 0x02
02725 * MODBUS_FC_READ_HOLDING_REGISTERS 0x03
02726 * MODBUS_FC_READ_INPUT_REGISTERS 0x04
02727 * MODBUS_FC_WRITE_SINGLE_COIL 0x05
02728 * MODBUS_FC_WRITE_SINGLE_REGISTER 0x06
02729 * MODBUS_FC_READ_EXCEPTION_STATUS 0x07
02730 * MODBUS_FC_WRITE_MULTIPLE_COILS 0x0F
02731 * MODBUS_FC_WRITE_MULTIPLE_REGISTERS 0x10
02732 * MODBUS_FC_REPORT_SLAVE_ID 0x11
02733 * MODBUS_FC_MASK_WRITE_REGISTER 0x16
02734 * MODBUS_FC_WRITE_AND_READ_REGISTERS 0x17
02735 *
02736 * @param data 0-255
02737 * @return
02738 *
02739 * @par Python
02740 * modbusSendCustomCommand(self: pyaubo_sdk.RegisterControl, arg0: str,
02741 * arg1: int, arg2: int, arg3: List[int]) -> int
02742 *
02743 * @par Lua
02744 * modbusSendCustomCommand(device_info: string, slave_number: number,
02745 * function_code: number, data: table) -> nil
02746 *
02747 * @par Lua
02748 * modbusSendCustomCommand("/dev/ttyRobotTool,115200,N,8,1", 1, 10,
02749 * {1,2,0,2,4,0,0,0}) -> nil
02750 *
02751 * @par JSON-RPC
02752 * {"json-
rpc":"2.0","method":"RegisterControl.modbusSendCustomCommand","params":["/dev/ttyRobotTool,115200,N,8,1",1,10,[1,2,0,2,4,0,0,0]],"id":1}
02753 *
02754 * @par JSON-RPC
02755 * {"id":1,"jsonrpc":"2.0","result":0}
02756 * \endchinese
02757 * \english
02758 * Sends a command specified by the user to the modbus unit located on the
02759 * specified IP address. Cannot be used to request data, since the response
02760 * will not be received. The user is responsible for supplying data which
02761 * is meaningful to the supplied function code. The builtin function takes
02762 * care of constructing the modbus frame, so the user should not be
02763 * concerned with the length of the command.
02764 *
02765 * @param device_info is rtu format.
02766 * eg,"serial_port,baud,parity,data_bit,stop_bit"
02767 * (1)The serial_port argument specifies the name of the serial port eg. On
02768 * Linux ,"/dev/ttyS0" or "/dev/ttyUSB0". On Windows, "\\.\COM10".
02769 * (2)The baud argument specifies the baud rate of the communication, eg.
02770 * 9600, 19200, 57600, 115200, etc.
02771 * (3)parity:N for none,E for even,O for odd.
02772 * (4)data_bit:The data_bits argument specifies the number of bits of data,
02773 * the allowed values are 5, 6, 7 and 8.
02774 * (5)stop_bit:The stop_bits argument
02775 * specifies the bits of stop, the allowed values are 1 and 2.
02776 *

```

```

02777 * device_info is tcp format, eg, "ip address, port"
02778 * (1) The ip address parameter specifies the ip address of the server
02779 * (2) The port parameter specifies the port number that the server is
02780 * listening on.
02781 * @param slave_number An integer specifying the slave number to use for
02782 * the custom command.
02783 * @param function_code An integer specifying the function code for the
02784 * custom command.
02785 *
02786 * Modbus function codes
02787 * MODBUS_FC_READ_COILS 0x01
02788 * MODBUS_FC_READ_DISCRETE_INPUTS 0x02
02789 * MODBUS_FC_READ_HOLDING_REGISTERS 0x03
02790 * MODBUS_FC_READ_INPUT_REGISTERS 0x04
02791 * MODBUS_FC_WRITE_SINGLE_COIL 0x05
02792 * MODBUS_FC_WRITE_SINGLE_REGISTER 0x06
02793 * MODBUS_FC_READ_EXCEPTION_STATUS 0x07
02794 * MODBUS_FC_WRITE_MULTIPLE_COILS 0x0F
02795 * MODBUS_FC_WRITE_MULTIPLE_REGISTERS 0x10
02796 * MODBUS_FC_REPORT_SLAVE_ID 0x11
02797 * MODBUS_FC_MASK_WRITE_REGISTER 0x16
02798 * MODBUS_FC_WRITE_AND_READ_REGISTERS 0x17
02799 *
02800 * @param data An array of integers in which each entry must be a valid
02801 * byte (0-255) value.
02802 * @return
02803 *
02804 * @par Python interface prototype
02805 * modbusSendCustomCommand(self: pyaubo_sdk.RegisterControl, arg0: str,
02806 * arg1: int, arg2: int, arg3: List[int]) -> int
02807 *
02808 * @par Lua interface prototype
02809 * modbusSendCustomCommand(device_info: string, slave_number: number,
02810 * function_code: number, data: table) -> nil
02811 *
02812 * @par Lua example
02813 * modbusSendCustomCommand("/dev/ttyRobotTool,115200,N,8,1", 1, 10,
02814 * {1,2,0,2,4,0,0,0}) -> nil
02815 *
02816 * @par JSON-RPC request example
02817 * {"json-
rpc":"2.0","method":"RegisterControl.modbusSendCustomCommand","params":["/dev/ttyRobotTool,115200,N,8,1",1,10,[1,2,0,2,4,0,0,0]],"id":1}
02818 *
02819 * @par JSON-RPC response example
02820 * {"id":1,"jsonrpc":"2.0","result":0}
02821 * \endenglish
02822 */
02823 int modbusSendCustomCommand(const std::string &device_info,
02824 int slave_number, int function_code,
02825 const std::vector<uint8_t> &data);
02826
02827 /**
02828 * @ingroup RegisterControl
02829 * \chinese
02830 * "default" "freedrive"
02831 *
02832 * @param robot_name
02833 * @param signal_name
02834 * @param action "default" "freedrive"
02835 * @return
02836 *
02837 * @par Python
02838 * modbusSetDigitalInputAction(self: pyaubo_sdk.RegisterControl, arg0: str,
02839 * arg1: str, arg2: int)
02840 *
02841 * @par Lua
02842 * modbusSetDigitalInputAction(robot_name: string, signal_name: string,
02843 * action: number) -> nil
02844 *
02845 * @par Lua
02846 * modbusSetDigitalInputAction("rob1", "Modbus_0", "Handguide")
02847 *
02848 * @par JSON-RPC
02849 * {"json-
rpc":"2.0","method":"RegisterControl.modbusSetDigitalInputAction","params":["rob1","Modbus_0","Handguide"],"id":1}
02850 *
02851 * @par JSON-RPC
02852 * {"id":1,"jsonrpc":"2.0","result":0}
02853 * \endchinese
02854 * \english
02855 * Sets the selected digital input signal to either a "default" or
02856 * "freedrive" action.
02857 *
02858 * @param robot_name A string identifying a robot name that connected robot
02859 * @param signal_name A string identifying a digital input signal that was
02860 * previously added.
02861 * @param action The type of action. The action can either be "default" or

```

```

02862 * "freedrive". (string)
02863 * @return
02864 *
02865 * @par Python interface prototype
02866 * modbusSetDigitalInputAction(self: pyaubo_sdk.RegisterControl, arg0: str,
02867 * arg1: str, arg2: int)
02868 *
02869 * @par Lua interface prototype
02870 * modbusSetDigitalInputAction(robot_name: string, signal_name: string,
02871 * action: number) -> nil
02872 *
02873 * @par Lua example
02874 * modbusSetDigitalInputAction("rob1", "Modbus_0", "Handguide")
02875 *
02876 * @par JSON-RPC request example
02877 * {"jsonrpc": "2.0", "method": "RegisterControl.modbusSetDigitalInputAction", "params": ["rob1", "Modbus_0", "Handguide"], "id": 1}
02878 *
02879 * @par JSON-RPC response example
02880 * {"id": 1, "jsonrpc": "2.0", "result": 0}
02881 * \endenglish
02882 */
02883 int modbusSetDigitalInputAction(const std::string &robot_name,
02884                                const std::string &signal_name,
02885                                StandardInputAction action);
02886
02887 /**
02888 * @ingroup RegisterControl
02889 * \chinese
02890 *   Modbus
02891 *
02892 * @param robot_name
02893 * @param signal_name
02894 * @param runstate
02895 * @return
02896 *
02897 * @par JSON-RPC
02898 * {"jsonrpc": "2.0", "method": "RegisterControl.modbusSetOutputRunstate", "params": ["rob1", "Modbus_0", "None"], "id": 1}
02899 *
02900 * @par JSON-RPC
02901 * {"id": 1, "jsonrpc": "2.0", "result": 0}
02902 *
02903 * \endchinese
02904 * \english
02905 * Set Modbus signal output action
02906 *
02907 * @param robot_name
02908 * @param signal_name
02909 * @param runstate
02910 * @return
02911 *
02912 * @par JSON-RPC request example
02913 * {"jsonrpc": "2.0", "method": "RegisterControl.modbusSetOutputRunstate", "params": ["rob1", "Modbus_0", "None"], "id": 1}
02914 *
02915 * @par JSON-RPC response example
02916 * {"id": 1, "jsonrpc": "2.0", "result": 0}
02917 *
02918 * \endenglish
02919 */
02920 int modbusSetOutputRunstate(const std::string &robot_name,
02921                             const std::string &signal_name,
02922                             StandardOutputRunState runstate);
02923
02924 /**
02925 * @ingroup RegisterControl
02926 * \chinese
02927 *
02928 *
02929 * @param signal_name
02930 * @param value      0-65535
02931 * @return
02932 *
02933 * @par Python
02934 * modbusSetOutputSignal(self: pyaubo_sdk.RegisterControl, arg0: str, arg1:
02935 * int) -> int
02936 *
02937 * @par Lua
02938 * modbusSetOutputSignal(signal_name: string, value: number) -> nil
02939 *
02940 * @par Lua
02941 * modbusSetOutputSignal("Modbus_0", 0)
02942 *
02943 * @par JSON-RPC
02944 * {"jsonrpc": "2.0", "method": "RegisterControl.modbusSetOutputSignal", "params": ["Modbus_0", 0], "id": 1}
02945 *

```

```

02946 * @par JSON-RPC
02947 * {"id":1,"jsonrpc":"2.0","result":0}
02948 * \endchinese
02949 * \english
02950 * Sets the output register signal identified by the given
02951 * value.
02952 *
02953 * @param signal_name A string identifying an output register signal that in
02954 * advance has been added.
02955 * @param value An integer which must be a valid word (0-65535)
02956 * @return
02957 *
02958 * @par Python interface prototype
02959 * modbusSetOutputSignal(self: pyaubo_sdk.RegisterControl, arg0: str, arg1:
02960 * int) -> int
02961 *
02962 * @par Lua interface prototype
02963 * modbusSetOutputSignal(signal_name: string, value: number) -> nil
02964 *
02965 * @par Lua example
02966 * modbusSetOutputSignal("Modbus_0",0)
02967 *
02968 * @par JSON-RPC request example
02969 * {"jsonrpc":"2.0","method":"RegisterControl.modbusSetOutputSignal","params":["Modbus_0",0],"id":1}
02970 *
02971 * @par JSON-RPC response example
02972 * {"id":1,"jsonrpc":"2.0","result":0}
02973 * \endenglish
02974 */
02975 int modbusSetOutputSignal(const std::string &signal_name, uint16_t value);
02976
02977 /**
02978 * @ingroup RegisterControl
02979 * \chinese
02980 *
02981 *
02982 * @param signal_name
02983 * @param values 0-65535
02984 * @return
02985 *
02986 * @par Python
02987 * modbusSetOutputSignal1(self: pyaubo_sdk.RegisterControl, arg0: str, arg1:
02988 * list[int]) -> int
02989 *
02990 * @par Lua
02991 * modbusSetOutputSignal1(signal_name: string, value: table) -> nil
02992 *
02993 * @par Lua
02994 * modbusSetOutputSignal1("Modbus_0", {0, 1, 2})
02995 *
02996 * @par JSON-RPC
02997 * {"jsonrpc":"2.0","method":"RegisterControl.modbusSetOutputSignal1","params":["Modbus_0",[0,1,2]],"id":1}
02998 *
02999 * @par JSON-RPC
03000 * {"id":1,"jsonrpc":"2.0","result":0}
03001 * \endchinese
03002 * \english
03003 * Sets multiple consecutive output register signals starting from the given
03004 * name to the given values.
03005 *
03006 * @param signal_name A string identifying the starting output register
03007 * signal that in advance has been added.
03008 * @param values An array of integers where each element must be a valid
03009 * word (0-65535), corresponding to multiple consecutive signals
03010 * @return
03011 *
03012 * @par Python interface prototype
03013 * modbusSetOutputSignal1(self: pyaubo_sdk.RegisterControl, arg0: str, arg1:
03014 * list[int]) -> int
03015 *
03016 * @par Lua interface prototype
03017 * modbusSetOutputSignal1(signal_name: string, value: table) -> nil
03018 *
03019 * @par Lua example
03020 * modbusSetOutputSignal1("Modbus_0", {0, 1, 2})
03021 *
03022 * @par JSON-RPC request example
03023 * {"jsonrpc":"2.0","method":"RegisterControl.modbusSetOutputSignal1","params":["Modbus_0",[0,1,2]],"id":1}
03024 *
03025 * @par JSON-RPC response example
03026 * {"id":1,"jsonrpc":"2.0","result":0}
03027 * \endenglish
03028 */
03029 int modbusSetOutputSignal1(const std::string &signal_name,
03030                          const std::vector<uint16_t> &values);
03031
03032 /**

```

```

03033 * \chinese
03034 *
03035 *
03036 * @param signal_name
03037 * @param value 0-65535
03038 * @param timeout
03039 * @return
03040 *
03041 * @par Python
03042 * modbusSetOutputSignalWithTimeout(self: pyaubo_sdk.RegisterControl, arg0:
03043 * str, arg1: int, arg2: double) -> int
03044 *
03045 * @par Lua
03046 * modbusSetOutputSignalWithTimeout(signal_name: string, value: number,
03047 * timeout: number) -> nil
03048 *
03049 * @par Lua
03050 * modbusSetOutputSignalWithTimeout("Modbus_0",0,0.5)
03051 *
03052 * @par JSON-RPC
03053 *
03054 * {"jsonrpc":"2.0","method":"RegisterControl.modbusSetOutputSignalWithTimeout","params":["Modbus_0",0,0.5],"id":1}
03055 *
03056 * @par JSON-RPC
03057 * {"id":1,"jsonrpc":"2.0","result":0}
03058 * \endchinese
03059 * \english
03060 * Sets the output register signal identified by the given name to the
03061 * given, with timeout value.
03062 *
03063 * @param signal_name A string identifying an output register signal that in
03064 * advance has been added.
03065 * @param value An integer which must be a valid word (0-65535)
03066 * @param timeout seconds
03067 * @return
03068 *
03069 * @par Python interface prototype
03070 * modbusSetOutputSignalWithTimeout(self: pyaubo_sdk.RegisterControl, arg0:
03071 * str, arg1: int, arg2: double) -> int
03072 *
03073 * @par Lua interface prototype
03074 * modbusSetOutputSignalWithTimeout(signal_name: string, value: number,
03075 * timeout: number) -> nil
03076 *
03077 * @par Lua example
03078 * modbusSetOutputSignalWithTimeout("Modbus_0",0,0.5)
03079 *
03080 * @par JSON-RPC request example
03081 * {"jsonrpc":"2.0","method":"RegisterControl.modbusSetOutputSignalWithTimeout","params":["Modbus_0",0,0.5],"id":1}
03082 *
03083 * @par JSON-RPC response example
03084 * {"id":1,"jsonrpc":"2.0","result":0}
03085 * \endenglish
03086 */
03087 int modbusSetOutputSignalWithTimeout(const std::string &signal_name,
03088                                     uint16_t value, double timeout);
03089 /**
03090 * @ingroup RegisterControl
03091 * \chinese
03092 * modbus ( )
03093 *
03094 * @param signal_name:
03095 * @param value: 0-65535
03096 * @param duration:
03097 * @return
03098 *
03099 * @par Python
03100 * modbusSetOutputSignalPulse(self: pyaubo_sdk.RegisterControl, arg0: str,
03101 * arg1: int, arg2 double) -> int
03102 *
03103 * @par Lua
03104 * modbusSetOutputSignalPulse(signal_name: string, value: number, duration:
03105 * number) -> nil
03106 *
03107 * @par Lua
03108 * modbusSetOutputSignalPulse("Modbus_0",1,0.5)
03109 *
03110 * @par JSON-RPC
03111 * {"jsonrpc":"2.0","method":"RegisterControl.modbusSetOutputSignalPulse","params":["Modbus_0",1,0.5],"id":1}
03112 *
03113 * @par JSON-RPC
03114 * {"id":1,"jsonrpc":"2.0","result":0}
03115 *
03116 * \endchinese
03117 * \english

```

```

03118 * Set modbus signal output pulse (only supports coil output type)
03119 *
03120 * @param signal_name: A string identifying an output register signal that
03121 * has been added in advance.
03122 * @param value: An integer which must be a valid word (0-65535)
03123 * @param duration: Duration of the signal, in seconds
03124 * @return
03125 *
03126 * @par Python interface prototype
03127 * modbusSetOutputSignalPulse(self: pyaubo_sdk.RegisterControl, arg0: str,
03128 * arg1: int, arg2: double) -> int
03129 *
03130 * @par Lua interface prototype
03131 * modbusSetOutputSignalPulse(signal_name: string, value: number, duration:
03132 * number) -> nil
03133 *
03134 * @par Lua example
03135 * modbusSetOutputSignalPulse("Modbus_0",1,0.5)
03136 *
03137 * @par JSON-RPC request example
03138 * {"jsonrpc":"2.0","method":"RegisterControl.modbusSetOutputSignalPulse","params":["Modbus_0",1,0.5],"id":1}
03139 *
03140 * @par JSON-RPC response example
03141 * {"id":1,"jsonrpc":"2.0","result":0}
03142 *
03143 * \endenglish
03144 */
03145 int modbusSetOutputSignalPulse(const std::string &signal_name,
03146                               uint16_t value, double duration);
03147
03148 /**
03149 * @ingroup RegisterControl
03150 * \chinese
03151 *     Modbus
03152 *
03153 * @param signal_name
03154 * @param update_frequency      0-125
03155 * @return
03156 *
03157 * @par Python
03158 * modbusSetSignalUpdateFrequency(self: pyaubo_sdk.RegisterControl, arg0:
03159 * str, arg1: int) -> int
03160 *
03161 * @par Lua
03162 * modbusSetSignalUpdateFrequency(signal_name: string, update_frequency:
03163 * number) -> nil
03164 *
03165 * @par Lua
03166 * modbusSetSignalUpdateFrequency("Modbus_0",1)
03167 *
03168 * @par JSON-RPC
03169 * {"jsonrpc":"2.0","method":"RegisterControl.modbusSetSignalUpdateFrequency","params":["Modbus_0",1],"id":1}
03170 *
03171 * @par JSON-RPC
03172 * {"id":1,"jsonrpc":"2.0","result":0}
03173 * \endchinese
03174 * \english
03175 * Sets the frequency with which the robot will send requests to the Modbus
03176 * controller to either read or write the signal value.
03177 *
03178 * @param signal_name A string identifying an output digital signal that
03179 * in advance has been added.
03180 * @param update_frequency An integer in the range 0-125 specifying the
03181 * update frequency in Hz.
03182 * @return
03183 *
03184 * @par Python interface prototype
03185 * modbusSetSignalUpdateFrequency(self: pyaubo_sdk.RegisterControl, arg0:
03186 * str, arg1: int) -> int
03187 *
03188 * @par Lua interface prototype
03189 * modbusSetSignalUpdateFrequency(signal_name: string, update_frequency:
03190 * number) -> nil
03191 *
03192 * @par Lua example
03193 * modbusSetSignalUpdateFrequency("Modbus_0",1)
03194 *
03195 * @par JSON-RPC request example
03196 * {"jsonrpc":"2.0","method":"RegisterControl.modbusSetSignalUpdateFrequency","params":["Modbus_0",1],"id":1}
03197 *
03198 * @par JSON-RPC response example
03199 * {"id":1,"jsonrpc":"2.0","result":0}
03200 * \endenglish
03201 */
03202 int modbusSetSignalUpdateFrequency(const std::string &signal_name,
03203                                   int update_frequency);
03204

```



```

03205  /**
03206  * @ingroup RegisterControl
03207  * \chinese
03208  *     modbus    0    -1
03209  *
03210  * @param signal_name
03211  * @return
03212  *
03213  * @par JSON-RPC
03214  * {"jsonrpc":"2.0","method":"RegisterControl.modbusGetSignalIndex","params":["Modbus_0"],"id":1}
03215  *
03216  * @par JSON-RPC
03217  * {"id":1,"jsonrpc":"2.0","result":0}
03218  *
03219  * \endchinese
03220  * \english
03221  * Get the index of the specified modbus signal, starting from 0. Returns -1
03222  * if it does not exist.
03223  *
03224  * @param signal_name
03225  * @return
03226  *
03227  * @par JSON-RPC request example
03228  * {"jsonrpc":"2.0","method":"RegisterControl.modbusGetSignalIndex","params":["Modbus_0"],"id":1}
03229  *
03230  * @par JSON-RPC response example
03231  * {"id":1,"jsonrpc":"2.0","result":0}
03232  *
03233  * \endenglish
03234  */
03235  int modbusGetSignalIndex(const std::string &signal_name);
03236
03237  /**
03238  * @ingroup RegisterControl
03239  * \chinese
03240  *     modbus
03241  *
03242  * @param signal_name
03243  * @return    ModbusErrorNum
03244  *
03245  * @par JSON-RPC
03246  * {"jsonrpc":"2.0","method":"RegisterControl.modbusGetSignalError","params":["Modbus_0"],"id":1}
03247  *
03248  * @par JSON-RPC
03249  * {"id":1,"jsonrpc":"2.0","result":6}
03250  * \endchinese
03251  * \english
03252  * Get the error status of the specified modbus signal
03253  *
03254  * @param signal_name
03255  * @return Returns error code ModbusErrorNum
03256  *
03257  * @par JSON-RPC request example
03258  * {"jsonrpc":"2.0","method":"RegisterControl.modbusGetSignalError","params":["Modbus_0"],"id":1}
03259  *
03260  * @par JSON-RPC response example
03261  * {"id":1,"jsonrpc":"2.0","result":6}
03262  * \endenglish
03263  */
03264  int modbusGetSignalError(const std::string &signal_name);
03265
03266  /**
03267  * @ingroup RegisterControl
03268  * \chinese
03269  *     modbus
03270  *
03271  * @param device_name
03272  *     TCP  "ip:port",  "127.0.0.1:502" \n
03273  *     RTU  "serial_port",  "/dev/ttyUSB0" \n
03274  *
03275  * @return
03276  * 0:
03277  * -1:
03278  * -2:
03279  *
03280  * @par JSON-RPC
03281  * {"jsonrpc":"2.0","method":"RegisterControl.getModbusDeviceStatus","params":["172.16.26.248:502"],"id":1}
03282  *
03283  * @par JSON-RPC
03284  * {"id":1,"jsonrpc":"2.0","result":0}
03285  *
03286  * \endchinese
03287  * \english
03288  * Get the connection status of the specified modbus device
03289  *
03290  * @param device_name
03291  * Device name in TCP format: "ip:port", e.g. "127.0.0.1:502" \n

```

```

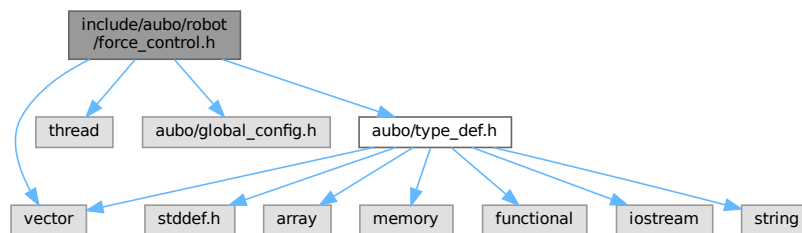
03292 * Device name in RTU format: "serial_port", e.g. "/dev/ttyUSB0" \n
03293 *
03294 * @return
03295 * 0: Device is connected
03296 * -1: Device does not exist
03297 * -2: Device is disconnected
03298 *
03299 * @par JSON-RPC request example
03300 * {"jsonrpc":"2.0","method":"RegisterControl.getModbusDeviceStatus","params":["172.16.26.248:502"],"id":1}
03301 *
03302 * @par JSON-RPC response example
03303 * {"id":1,"jsonrpc":"2.0","result":0}
03304 *
03305 * \endenglish
03306 */
03307 int getModbusDeviceStatus(const std::string &device_name);
03308
03309 /**
03310 * @ingroup RegisterControl
03311 * \chinese
03312 *   modbus
03313 *
03314 * @param encoder_id 0
03315 * @param signal_name modbus
03316 * @return
03317 * \endchinese
03318 * \english
03319 * Use a modbus register signal as an encoder
03320 *
03321 * @param encoder_id Must not be 0
03322 * @param signal_name Name of the modbus signal, must be of register type
03323 * @return
03324 * \endenglish
03325 */
03326 int addModbusEncoder(int encoder_id, int range_id,
03327                     const std::string &signal_name);
03328
03329 /**
03330 * @ingroup RegisterControl
03331 * \chinese
03332 *   Int32
03333 *
03334 * @param encoder_id ID
03335 * @param range_id ID
03336 * @param key
03337 * @return
03338 * \endchinese
03339 * \english
03340 * Add a virtual encoder for an Int32 register
03341 *
03342 * @param encoder_id Encoder ID
03343 * @param range_id Range ID
03344 * @param key Variable name
03345 * @return
03346 * \endenglish
03347 */
03348 int addInt32RegEncoder(int encoder_id, int range_id,
03349                       const std::string &key);
03350
03351 /**
03352 * @ingroup RegisterControl
03353 * \chinese
03354 *
03355 *
03356 * @param encoder_id
03357 * @return
03358 * \endchinese
03359 * \english
03360 * Delete virtual encoder
03361 *
03362 * @param encoder_id
03363 * @return
03364 * \endenglish
03365 */
03366 int deleteVirtualEncoder(int encoder_id);
03367
03368 protected:
03369 void *d_;
03370 };
03371 using RegisterControlPtr = std::shared_ptr<RegisterControl>;
03372
03373 } // namespace common_interface
03374 } // namespace arcs
03375
03376 #endif // AUBO_SDK_REGISTER_CONTROL_INTERFACE_H

```

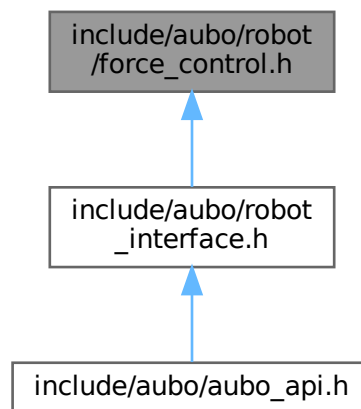
12.20 include/aubo/robot/force_control.h File Reference

```
#include <vector>
#include <thread>
#include <aubo/global_config.h>
#include <aubo/type_def.h>
```

Include dependency graph for force_control.h:



This graph shows which files directly or indirectly include this file:



Classes

- class [arcs::common_interface::ForceControl](#)

Namespaces

- namespace [arcs](#)
- namespace [arcs::common_interface](#)

Typedefs

- using [arcs::common_interface::ForceControlPtr](#) = std::shared_ptr<[ForceControl](#)>

12.20.1 Detailed Description

Force control interface

Force control limitations When the robot is force controlled, the following functionality is not accessible:

- Collision Detection (option 613-1)
- SoftMove (option 885-1)
- Tracking functionality like Conveyor Tracking (option 606-1), Optical Tracking (6601) and Weld Guide (815-2)
- Sensor Synchronization or Analog Synchronization
- World Zones (option 608-1)
- Independent Axes (option 610-1)
- Path Offset (option 612-1)
- Arc options
- PickMaster options
- Joint soft servo (instruction SoftAct)
- Force Control cannot be activated when the robot is running in MultiMove Coordinated mode (option 604-1).
- If Force Control is used together with SafeMove (option 810-2) or Electronic Position Switches (option 810-1), the function Operational Safety Range must be used. See the respective manual for these options.
- RAPID instructions such as FCAct, FCDeact, FCConditionWaitWhile and FCRefStop can only be called from normal level in a motion task.

Applications: polishing, grinding, cleaning

FC Pressure

Set the z direction of the trajectory coordinate system as the force control axis, set spring to 0

Before contact, set output force to 0, spring to a fixed value (determined by vel)

Leaving the contact surface: set output force to 0, spring to a fixed value

Piston assembly

Forward clutch hub

Set force control termination mode

Drag teaching based on end force sensor

spring = 0; force_ref = 0; reference trajectory point arbitrary

Definition in file [force_control.h](#).

12.21 force_control.h

[Go to the documentation of this file.](#)

```
00001 /**
00002  * \chinese
00003  * @file force_control.h
00004  * @brief
00005  *
00006  *
00007  *
00008  *
00009  * • 613-1 \n
00010  * • SoftMove 885-1 \n
00011  * • 606-1 6601 815-2 \n
00012  * • \n
00013  * • 608-1 \n
00014  * • 610-1 \n
```

```

00015 * • 612-1 \n
00016 * • \n
00017 * • PickMaster \n
00018 * • SoftAct \n
00019 * • MultiMove 604-1 \n
00020 * • SafeMove 810-2 810-1 \n
00021 * • RAPID FCAct FCDeact FCCConditionWaitWhile FCRefStop
00022 *
00023 * \n
00024 * FC Pressure\n
00025 * z spring 0\n
00026 * 0 spring vel \n
00027 * 0 spring \n
00028 *
00029 * \n
00030 * Forward clutch hub\n
00031 *
00032 *
00033 * \n
00034 * spring = 0; force_ref = 0;
00035 * \endchinese
00036 * \english
00037 * @file force_control.h
00038 * @brief Force control interface
00039 *
00040 * Force control limitations
00041 * When the robot is force controlled, the following functionality is not accessible:
00042 *
00043 * • Collision Detection (option 613-1) \n
00044 * • SoftMove (option 885-1) \n
00045 * • Tracking functionality like Conveyor Tracking (option 606-1), Optical Tracking (6601) and Weld Guide (815-2) \n
00046 * • Sensor Synchronization or Analog Synchronization \n
00047 * • World Zones (option 608-1) \n
00048 * • Independent Axes (option 610-1) \n
00049 * • Path Offset (option 612-1) \n
00050 * • Arc options \n
00051 * • PickMaster options \n
00052 * • Joint soft servo (instruction SoftAct) \n
00053 * • Force Control cannot be activated when the robot is running in MultiMove Coordinated mode (option 604-1). \n
00054 * • If Force Control is used together with SafeMove (option 810-2) or Electronic Position Switches (option 810-1), the
function Operational Safety Range must be used. See the respective manual for these options. \n
00055 * • RAPID instructions such as FCAct, FCDeact, FCCConditionWaitWhile and FCRefStop can only be called from normal
level in a motion task.
00056 *
00057 * Applications: polishing, grinding, cleaning \n
00058 * FC Pressure \n
00059 * Set the z direction of the trajectory coordinate system as the force control axis, set spring to 0 \n
00060 * Before contact, set output force to 0, spring to a fixed value (determined by vel) \n
00061 * Leaving the contact surface: set output force to 0, spring to a fixed value \n
00062 *
00063 * Piston assembly \n
00064 * Forward clutch hub \n
00065 * Set force control termination mode
00066 *
00067 * Drag teaching based on end force sensor \n
00068 * spring = 0; force_ref = 0; reference trajectory point arbitrary
00069 * \endenglish
00070 */
00071 #ifndef AUBO_SDK_FORCE_CONTROL_INTERFACE_H
00072 #define AUBO_SDK_FORCE_CONTROL_INTERFACE_H
00073
00074 #include <vector>
00075 #include <thread>
00076
00077 #include <aubo/global_config.h>
00078 #include <aubo/type_def.h>
00079
00080 namespace arcs {
00081 namespace common_interface {
00082
00083 /**
00084 * @defgroup ForceControl ForceControl( )
00085 * @ingroup RobotInterface
00086 * \chinese
00087 *
00088 *
00089 * \endchinese
00090 * \english
00091 * Abstract class for force control interface
00092 * \endenglish
00093 */
00094 class ARCS_ABI_EXPORT ForceControl
00095 {
00096 public:
00097 ForceControl();
00098 virtual ~ForceControl();
00099

```

```

00100  /**
00101  * @ingroup ForceControl
00102  * \chinese
00103  *
00104  * fcEnable
00105  * fcEnable
00106  *     fcEnable
00107  *
00108  *         fcEnable
00109  *
00110  * @return 0
00111  * AUBO_BUSY
00112  * AUBO_BAD_STATE
00113  * -AUBO_BAD_STATE
00114  *
00115  * @throws arcs::common_interface::AuboException
00116  *
00117  * @par Python
00118  * fcEnable(self: pyaubo_sdk.ForceControl) -> int
00119  *
00120  * @par Lua
00121  * fcEnable() -> nil
00122  *
00123  * @par Lua
00124  * fcEnable()
00125  *
00126  * @par JSON-RPC
00127  * {"jsonrpc":"2.0","method":"rob1.ForceControl.fcEnable","params":[],"id":1}
00128  *
00129  * @par JSON-RPC
00130  * {"id":1,"jsonrpc":"2.0","result":0}
00131  *
00132  * \endchinese
00133  *
00134  * \english
00135  * Start force control
00136  *
00137  * fcEnable is used to enable Force Control. At the same time as Force
00138  * Control is enabled, fcEnable is used to define the coordinate system
00139  * for Force Control, and tune the force and torque damping. If a coordinate
00140  * system is not specified in fcEnable a default force control coordinate
00141  * system is created with the same orientation as the work object coordinate
00142  * system. All Force Control supervisions are activated by fcEnable.
00143  *
00144  *
00145  * @return Return 0 if succeeded; return error code if failed
00146  * AUBO_BUSY
00147  * AUBO_BAD_STATE
00148  * -AUBO_BAD_STATE
00149  *
00150  * @throws arcs::common_interface::AuboException
00151  *
00152  * @par Python Function Prototype
00153  * fcEnable(self: pyaubo_sdk.ForceControl) -> int
00154  *
00155  * @par Lua Function Prototype
00156  * fcEnable() -> nil
00157  *
00158  * @par Lua example
00159  * fcEnable()
00160  *
00161  * @par JSON-RPC Request example
00162  * {"jsonrpc":"2.0","method":"rob1.ForceControl.fcEnable","params":[],"id":1}
00163  *
00164  * @par JSON-RPC Response example
00165  * {"id":1,"jsonrpc":"2.0","result":0}
00166  * \endenglish
00167  */
00168
00169 int fcEnable();
00170
00171  /**
00172  * @ingroup ForceControl
00173  * \chinese
00174  *
00175  * fcDisable
00176  *
00177  * @return 0
00178  * AUBO_BUSY
00179  * AUBO_BAD_STATE
00180  * -AUBO_BAD_STATE
00181  *
00182  * @throws arcs::common_interface::AuboException
00183  *
00184  * @par Python
00185  * fcDisable(self: pyaubo_sdk.ForceControl) -> int
00186  *

```

```

00187 * @par Lua
00188 * fcDisable() -> nil
00189 *
00190 * @par Lua
00191 * fcDisable()
00192 *
00193 * @par JSON-RPC
00194 * {"jsonrpc":"2.0","method":"rob1.ForceControl.fcDisable","params":[],"id":1}
00195 *
00196 * @par JSON-RPC
00197 * {"id":1,"jsonrpc":"2.0","result":0}
00198 * \endchinese
00199 *
00200 * \english
00201 * End force control
00202 *
00203 * fcDisable is used to disable Force Control. After a successful
00204 * deactivation the robot is back in position control.
00205 *
00206 * @return Return 0 if succeeded; return error code if failed
00207 * AUBO_BUSY
00208 * AUBO_BAD_STATE
00209 * -AUBO_BAD_STATE
00210 *
00211 * @throws arcs::common_interface::AuboException
00212 *
00213 * @par Python Function Prototype
00214 * fcDisable(self: pyaubo_sdk.ForceControl) -> int
00215 *
00216 * @par Lua Function Prototype
00217 * fcDisable() -> nil
00218 *
00219 * @par Lua example
00220 * fcDisable()
00221 *
00222 * @par JSON-RPC Request example
00223 * {"jsonrpc":"2.0","method":"rob1.ForceControl.fcDisable","params":[],"id":1}
00224 *
00225 * @par JSON-RPC Response example
00226 * {"id":1,"jsonrpc":"2.0","result":0}
00227 * \endenglish
00228 */
00229
00230 int fcDisable();
00231
00232 /**
00233 * @ingroup ForceControl
00234 * \chinese
00235 *
00236 *
00237 * @return true false
00238 *
00239 * @throws arcs::common_interface::AuboException
00240 *
00241 * @par Python
00242 * isFcEnabled(self: pyaubo_sdk.ForceControl) -> bool
00243 *
00244 * @par Lua
00245 * isFcEnabled() -> boolean
00246 *
00247 * @par Lua
00248 * Fc_status = isFcEnabled()
00249 *
00250 * @par JSON-RPC
00251 * {"jsonrpc":"2.0","method":"rob1.ForceControl.isFcEnabled","params":[],"id":1}
00252 *
00253 * @par JSON-RPC
00254 * {"id":1,"jsonrpc":"2.0","result":false}
00255 * \endchinese
00256 *
00257 * \english
00258 * Check if force control is enabled
00259 *
00260 * @return Returns true if enabled, false if disabled
00261 *
00262 * @throws arcs::common_interface::AuboException
00263 *
00264 * @par Python Function Prototype
00265 * isFcEnabled(self: pyaubo_sdk.ForceControl) -> bool
00266 *
00267 * @par Lua Function Prototype
00268 * isFcEnabled() -> boolean
00269 *
00270 * @par Lua example
00271 * Fc_status = isFcEnabled()
00272 *
00273 * @par JSON-RPC Request example

```

```

00274 * {"jsonrpc":"2.0","method":"rob1.ForceControl.isFcEnabled","params":[],"id":1}
00275 *
00276 * @par JSON-RPC Response example
00277 * {"id":1,"jsonrpc":"2.0","result":false}
00278 * \endenglish
00279 */
00280 bool isFcEnabled();
00281
00282 /**
00283 * @ingroup ForceControl
00284 * \chinese
00285 * ( )
00286 *
00287 * @param feature:
00288 * @param compliance:
00289 * @param wrench: /
00290 * @param limits:
00291 * @param type:
00292 *
00293 *
00294 * 1.
00295 * feature = {0,0,0,0,0,0}
00296 * type = TaskFrameType::NONE
00297 *
00298 * 2.
00299 * feature = {0,0,0,0,0,0}
00300 * type = TaskFrameType::TOOL_FORCE
00301 *
00302 * 3. TCP
00303 * feature = tcp_offset
00304 * type = TaskFrameType::TOOL_FORCE
00305 *
00306 * 4. FRAME_FORCE
00307 * type = TaskFrameType::FRAME_FORCE
00308 * feature getTcpPose() TCP
00309 *
00310 * @return 0
00311 * AUBO_BUSY
00312 * AUBO_BAD_STATE
00313 * -AUBO_INVL_ARGUMENT
00314 * -AUBO_BAD_STATE
00315 *
00316 * @throws arcs::common_interface::AuboException
00317 *
00318 * @par Python
00319 * setTargetForce(self: pyaubo_sdk.ForceControl, arg0: List[float], arg1:
00320 * List[bool], arg2: List[float], arg3: List[float], arg4:
00321 * arcs::common_interface::TaskFrameType) -> int
00322 *
00323 * @par Lua
00324 * setTargetForce(feature: table, compliance: table, wrench: table, limits:
00325 * table, type: number) -> nil
00326 *
00327 * @par Lua
00328 * setTargetForce({0,0,0,0,0,0},{true,false,false,false,false,false},{10,0,0,0,0,0},{0,0,0,0,0,0},4)
00329 *
00330 * \endchinese
00331 * \english
00332 * Set force control reference (target) value
00333 *
00334 * @param feature: Reference geometric feature for generating force control reference frame
00335 * @param compliance: Compliance axis (direction) selection
00336 * @param wrench: Target force/torque
00337 * @param limits: Velocity limits
00338 * @param type: Force control reference frame type
00339 *
00340 * Usage Examples:
00341 * 1. Base Coordinate Frame:
00342 * feature = {0,0,0,0,0,0}
00343 * type = TaskFrameType::NONE
00344 *
00345 * 2. Flange Coordinate Frame:
00346 * feature = {0,0,0,0,0,0}
00347 * type = TaskFrameType::TOOL_FORCE
00348 *
00349 * 3. TCP Coordinate Frame:
00350 * feature = tcp_offset
00351 * type = TaskFrameType::TOOL_FORCE
00352 *
00353 * 4. User Coordinate Frame (FRAME_FORCE):
00354 * type = TaskFrameType::FRAME_FORCE
00355 * feature should be the user-defined reference frame,
00356 * for example, getTcpPose() means setting the force control frame to current TCP pose.
00357 *
00358 * @return Return 0 if succeeded; return error code if failed
00359 * AUBO_BUSY
00360 * AUBO_BAD_STATE

```



```

00361 * -AUBO_INVL_ARGUMENT
00362 * -AUBO_BAD_STATE
00363 *
00364 * @throws arcs::common_interface::AuboException
00365 *
00366 * @par Python Function Prototype
00367 * setTargetForce(self: pyaubo_sdk.ForceControl, arg0: List[float], arg1:
00368 * List[bool], arg2: List[float], arg3: List[float], arg4:
00369 * arcs::common_interface::TaskFrameType) -> int
00370 *
00371 * @par Lua Function Prototype
00372 * setTargetForce(feature: table, compliance: table, wrench: table, limits:
00373 * table, type: number) -> nil
00374 *
00375 * @par Lua example
00376 * setTargetForce({0,0,0,0,0,0},{true,false,false,false,false,false},{10,0,0,0,0,0},{0,0,0,0,0,0},4)
00377 *
00378 * \endenglish
00379 */
00380 int setTargetForce(const std::vector<double> &feature,
00381                   const std::vector<bool> &compliance,
00382                   const std::vector<double> &wrench,
00383                   const std::vector<double> &limits,
00384                   TaskFrameType type = TaskFrameType::FRAME_FORCE);
00385
00386 /**
00387 * @ingroup ForceControl
00388 * \chinese
00389 *
00390 *
00391 * @param env_stiff:          [0, 1] 0
00392 * @param damp_scale:        [0, 1] 0.5
00393 * @param stiff_scale:       [0, 1] 0.5
00394 *
00395 * @return 0
00396 * AUBO_BUSY
00397 * AUBO_BAD_STATE
00398 * -AUBO_INVL_ARGUMENT
00399 * -AUBO_BAD_STATE
00400 *
00401 * @throws arcs::common_interface::AuboException
00402 *
00403 * @par Python
00404 * setDynamicModel1(self: pyaubo_sdk.ForceControl, arg0: List[float],
00405 * arg1: List[float], arg2: List[float]) -> int
00406 *
00407 * @par Lua
00408 * setDynamicModel1(env_stiff: table, damp_scale: table, stiff_scale:
00409 * table) -> nil
00410 *
00411 * @par Lua
00412 * setDynamicModel1({0,0,0,0,0,0},{0.5,0.5,0.5,0.5,0.5,0.5},{0.5,0.5,0.5,0.5,0.5,0.5})
00413 *
00414 * \endchinese
00415 * \english
00416 * Set force control dynamics model
00417 *
00418 * @param env_stiff: Environment stiffness, representing the workpiece stiffness in the contact axis direction, range [0,
00419 1], default 0
00420 * @param damp_scale: Parameter representing damping level, range [0, 1], default 0.5
00421 * @param stiff_scale: Parameter representing stiffness level, range [0, 1], default 0.5
00422 *
00423 * @return Return 0 if succeeded; return error code if failed
00424 * AUBO_BUSY
00425 * AUBO_BAD_STATE
00426 * -AUBO_INVL_ARGUMENT
00427 * -AUBO_BAD_STATE
00428 *
00429 * @throws arcs::common_interface::AuboException
00430 *
00431 * @par Python Function Prototype
00432 * setDynamicModel1(self: pyaubo_sdk.ForceControl, arg0: List[float],
00433 * arg1: List[float], arg2: List[float]) -> int
00434 *
00435 * @par Lua Function Prototype
00436 * setDynamicModel1(env_stiff: table, damp_scale: table, stiff_scale:
00437 * table) -> nil
00438 *
00439 * @par Lua example
00440 * setDynamicModel1({0,0,0,0,0,0},{0.5,0.5,0.5,0.5,0.5,0.5},{0.5,0.5,0.5,0.5,0.5,0.5})
00441 *
00442 * \endenglish
00443 */
00444 int setDynamicModel1(const std::vector<double> &env_stiff,
00445                     const std::vector<double> &damp_scale,
00446                     const std::vector<double> &stiff_scale);
00447

```

```

00447  /**
00448  * @ingroup ForceControl
00449  * \chinese
00450  *
00451  *
00452  * @param env_stiff:          [0, 1]  0
00453  * @param damp_scale:        [0, 1]  0.5
00454  * @param stiff_scale:       [0, 1]  0.5
00455  *
00456  * @return MDK
00457  * AUBO_BUSY
00458  * AUBO_BAD_STATE
00459  * -AUBO_INVL_ARGUMENT
00460  * -AUBO_BAD_STATE
00461  *
00462  * @throws arcs::common_interface::AuboException
00463  *
00464  * @par Python
00465  * fcCalDynamicModel(self: pyaubo_sdk.ForceControl, arg0: List[float],
00466  * arg1: List[float], arg2: List[float]) -> Tuple[List[float], List[float], List[float]]
00467  *
00468  * @par Lua
00469  * fcCalDynamicModel(env_stiff: table, damp_scale: table, stiff_scale: table) -> table
00470  *
00471  * @par Lua
00472  * fc_table = fcCalDynamicModel({0,0,0,0,0},{0.5,0.5,0.5,0.5,0.5,0.5},{0.5,0.5,0.5,0.5,0.5,0.5})
00473  *
00474  * \endchinese
00475  * \english
00476  * Calculate force control dynamics model
00477  *
00478  * @param env_stiff: Environment stiffness, representing the workpiece stiffness in the contact axis direction, range [0,
00479 1], default 0
00480  * @param damp_scale: Parameter representing damping level, range [0, 1], default 0.5
00481  * @param stiff_scale: Parameter representing stiffness level, range [0, 1], default 0.5
00482  *
00483  * @return Force control dynamics model MDK
00484  * AUBO_BUSY
00485  * AUBO_BAD_STATE
00486  * -AUBO_INVL_ARGUMENT
00487  * -AUBO_BAD_STATE
00488  *
00489  * @throws arcs::common_interface::AuboException
00490  *
00491  * @par Python Function Prototype
00492  * fcCalDynamicModel(self: pyaubo_sdk.ForceControl, arg0: List[float],
00493  * arg1: List[float], arg2: List[float]) -> Tuple[List[float], List[float], List[float]]
00494  *
00495  * @par Lua Function Prototype
00496  * fcCalDynamicModel(env_stiff: table, damp_scale: table, stiff_scale: table) -> table
00497  *
00498  * @par Lua example
00499  * fc_table = fcCalDynamicModel({0,0,0,0,0},{0.5,0.5,0.5,0.5,0.5,0.5},{0.5,0.5,0.5,0.5,0.5,0.5})
00500  * \endenglish
00501  */
00502 DynamicsModel fcCalDynamicModel(const std::vector<double> &env_stiff,
00503                                const std::vector<double> &damp_scale,
00504                                const std::vector<double> &stiff_scale);
00505
00506  /**
00507  * @ingroup ForceControl
00508  * \chinese
00509  *
00510  *
00511  * @param damp_scale:        [0, 1]  0.5
00512  * @param stiff_scale:       [0, 1]  0.5
00513  *
00514  * @return 0
00515  * AUBO_BUSY
00516  * AUBO_BAD_STATE
00517  * -AUBO_INVL_ARGUMENT
00518  * -AUBO_BAD_STATE
00519  *
00520  * @throws arcs::common_interface::AuboException
00521  *
00522  * @par Python
00523  * setDynamicModelSearch(self: pyaubo_sdk.ForceControl, arg0: List[float],
00524  * arg1: List[float]) -> int
00525  *
00526  * @par Lua
00527  * setDynamicModelSearch(damp_scale: table, stiff_scale: table) -> nil
00528  *
00529  * @par Lua
00530  * setDynamicModelSearch({0.5,0.5,0.5,0.5,0.5,0.5},{0.5,0.5,0.5,0.5,0.5,0.5})
00531  *
00532  * \endchinese

```

```

00533 * \english
00534 * Set force control dynamics model for hole searching scenario
00535 *
00536 * @param damp_scale: Parameter representing damping level, range [0, 1], default 0.5
00537 * @param stiff_scale: Parameter representing stiffness level, range [0, 1], default 0.5
00538 *
00539 * @return Return 0 if succeeded; return error code if failed
00540 * AUBO_BUSY
00541 * AUBO_BAD_STATE
00542 * -AUBO_INVL_ARGUMENT
00543 * -AUBO_BAD_STATE
00544 *
00545 * @throws arcs::common_interface::AuboException
00546 *
00547 * @par Python Function Prototype
00548 * setDynamicModelSearch(self: pyaubo_sdk.ForceControl, arg0: List[float],
00549 * arg1: List[float]) -> int
00550 *
00551 * @par Lua Function Prototype
00552 * setDynamicModelSearch(damp_scale: table, stiff_scale: table) -> nil
00553 *
00554 * @par Lua example
00555 * setDynamicModelSearch({0.5,0.5,0.5,0.5,0.5,0.5},{0.5,0.5,0.5,0.5,0.5,0.5})
00556 *
00557 * \endenglish
00558 */
00559 int setDynamicModelSearch(const std::vector<double> &damp_scale,
00560                          const std::vector<double> &stiff_scale);
00561
00562 /**
00563 * @ingroup ForceControl
00564 * \chinese
00565 * /
00566 *
00567 * @param damp_scale:      [0, 1]  0.5
00568 * @param stiff_scale:     [0, 1]  0.5
00569 *
00570 * @return 0
00571 * AUBO_BUSY
00572 * AUBO_BAD_STATE
00573 * -AUBO_INVL_ARGUMENT
00574 * -AUBO_BAD_STATE
00575 *
00576 * @throws arcs::common_interface::AuboException
00577 *
00578 * @par Python
00579 * setDynamicModelInsert(self: pyaubo_sdk.ForceControl, arg0: List[float],
00580 * arg1: List[float]) -> int
00581 *
00582 * @par Lua
00583 * setDynamicModelInsert(damp_scale: table, stiff_scale: table) -> nil
00584 *
00585 * @par Lua
00586 * setDynamicModelInsert({0.5,0.5,0.5,0.5,0.5,0.5},{0.5,0.5,0.5,0.5,0.5,0.5})
00587 *
00588 * \endchinese
00589 * \english
00590 * Set force control dynamics model for insertion/extraction scenario
00591 *
00592 * @param damp_scale: Parameter representing damping level, range [0, 1], default 0.5
00593 * @param stiff_scale: Parameter representing stiffness level, range [0, 1], default 0.5
00594 *
00595 * @return Return 0 if succeeded; return error code if failed
00596 * AUBO_BUSY
00597 * AUBO_BAD_STATE
00598 * -AUBO_INVL_ARGUMENT
00599 * -AUBO_BAD_STATE
00600 *
00601 * @throws arcs::common_interface::AuboException
00602 *
00603 * @par Python Function Prototype
00604 * setDynamicModelInsert(self: pyaubo_sdk.ForceControl, arg0: List[float],
00605 * arg1: List[float]) -> int
00606 *
00607 * @par Lua Function Prototype
00608 * setDynamicModelInsert(damp_scale: table, stiff_scale: table) -> nil
00609 *
00610 * @par Lua example
00611 * setDynamicModelInsert({0.5,0.5,0.5,0.5,0.5,0.5},{0.5,0.5,0.5,0.5,0.5,0.5})
00612 *
00613 * \endenglish
00614 */
00615 int setDynamicModelInsert(const std::vector<double> &damp_scale,
00616                          const std::vector<double> &stiff_scale);
00617
00618 /**
00619 * @ingroup ForceControl
00619 * \chinese

```

```

00620 *
00621 *
00622 * @param env_stiff:      [0, 1]  0
00623 * @param damp_scale:    [0, 1]  0.5
00624 * @param stiff_scale:    [0, 1]  0.5
00625 *
00626 * @return 0
00627 * AUBO_BUSY
00628 * AUBO_BAD_STATE
00629 * -AUBO_INVL_ARGUMENT
00630 * -AUBO_BAD_STATE
00631 *
00632 * @throws arcs::common_interface::AuboException
00633 *
00634 * @par Python
00635 * setDynamicModelContact(self: pyaubo_sdk.ForceControl, arg0: List[float],
00636 * arg1: List[float], arg2: List[float]) -> int
00637 *
00638 * @par Lua
00639 * setDynamicModelContact(env_stiff: table, damp_scale: table, stiff_scale:
00640 * table) -> nil
00641 *
00642 * @par Lua
00643 * setDynamicModelContact({0,0,0,0,0,0},{0.5,0.5,0.5,0.5,0.5,0.5},{0.5,0.5,0.5,0.5,0.5,0.5})
00644 *
00645 * \endchinese
00646 * \english
00647 * Set force control dynamics model for contact scenario
00648 *
00649 * @param env_stiff: Parameter representing environment stiffness, range [0, 1], default 0
00650 * @param damp_scale: Parameter representing damping level, range [0, 1], default 0.5
00651 * @param stiff_scale: Parameter representing stiffness level, range [0, 1], default 0.5
00652 *
00653 * @return Return 0 if succeeded; return error code if failed
00654 * AUBO_BUSY
00655 * AUBO_BAD_STATE
00656 * -AUBO_INVL_ARGUMENT
00657 * -AUBO_BAD_STATE
00658 *
00659 * @throws arcs::common_interface::AuboException
00660 *
00661 * @par Python Function Prototype
00662 * setDynamicModelContact(self: pyaubo_sdk.ForceControl, arg0: List[float],
00663 * arg1: List[float], arg2: List[float]) -> int
00664 *
00665 * @par Lua Function Prototype
00666 * setDynamicModelContact(env_stiff: table, damp_scale: table, stiff_scale:
00667 * table) -> nil
00668 *
00669 * @par Lua example
00670 * setDynamicModelContact({0,0,0,0,0,0},{0.5,0.5,0.5,0.5,0.5,0.5},{0.5,0.5,0.5,0.5,0.5,0.5})
00671 *
00672 * \endenglish
00673 */
00674 int setDynamicModelContact(const std::vector<double> &env_stiff,
00675                             const std::vector<double> &damp_scale,
00676                             const std::vector<double> &stiff_scale);
00677
00678 /**
00679 * @ingroup ForceControl
00680 * \chinese
00681 *
00682 *
00683 * @param m
00684 * @param d
00685 * @param k
00686 *
00687 *
00688 * -
00689 * -      kg      6      [x, y, z, Rx, Ry, Rz]
00690 * -      kg · m2  6      [J1, J2, J3, J4, J5, J6]
00691 * -
00692 * -      N · s/m   [x, y, z, Rx, Ry, Rz]
00693 * -      N · m · s/rad [J1, J2, J3, J4, J5, J6]
00694 * -
00695 * -      N/m      [x, y, z, Rx, Ry, Rz]
00696 * -      N · m/rad [J1, J2, J3, J4, J5, J6]
00697 *
00698 * @return 0
00699 * AUBO_BUSY
00700 * AUBO_BAD_STATE
00701 * -AUBO_INVL_ARGUMENT
00702 * -AUBO_BAD_STATE
00703 *
00704 * @throws arcs::common_interface::AuboException
00705 *
00706 * @par Python

```

```

00707 * setDynamicModel(self: pyaubo_sdk.ForceControl, arg0: List[float], arg1:
00708 * List[float], arg2: List[float]) -> int
00709 *
00710 * @par Lua
00711 * setDynamicModel(m: table, d: table, k: table) -> nil
00712 *
00713 * @par Lua
00714 * setDynamicModel({20.0, 20.0, 20.0, 10.0, 10.0, 10.0},{2000, 0, 0, 0, 0, 0},{0, 0, 0, 0, 0, 0})
00715 *
00716 * \endchinese
00717 * \english
00718 * Set force control dynamics model
00719 *
00720 * @param m Mass parameters
00721 * @param d Damping parameters
00722 * @param k Stiffness parameters
00723 *
00724 * Parameter unit description:
00725 * - Mass:
00726 *   - Cartesian space: unit is kg, length is 6, order is [x, y, z, Rx, Ry, Rz]
00727 *   - Joint space: unit is kg · m2, length is 6, order is [J1, J2, J3, J4, J5, J6]
00728 * - Damping:
00729 *   - Cartesian space: unit is N · s/m, order is [x, y, z, Rx, Ry, Rz]
00730 *   - Joint space: unit is N · m · s/rad, order is [J1, J2, J3, J4, J5, J6]
00731 * - Stiffness:
00732 *   - Cartesian space: unit is N/m, order is [x, y, z, Rx, Ry, Rz]
00733 *   - Joint space: unit is N · m/rad, order is [J1, J2, J3, J4, J5, J6]
00734 *
00735 * @return Return 0 if succeeded; return error code if failed
00736 * AUBO_BUSY
00737 * AUBO_BAD_STATE
00738 * -AUBO_INVL_ARGUMENT
00739 * -AUBO_BAD_STATE
00740 *
00741 * @throws arcs::common_interface::AuboException
00742 *
00743 * @par Python Function Prototype
00744 * setDynamicModel(self: pyaubo_sdk.ForceControl, arg0: List[float], arg1:
00745 * List[float], arg2: List[float]) -> int
00746 *
00747 * @par Lua Function Prototype
00748 * setDynamicModel(m: table, d: table, k: table) -> nil
00749 *
00750 * @par Lua example
00751 * setDynamicModel({20.0, 20.0, 20.0, 10.0, 10.0, 10.0},{2000, 0, 0, 0, 0, 0},{0, 0, 0, 0, 0, 0})
00752 *
00753 * \endenglish
00754 */
00755 int setDynamicModel(const std::vector<double> &m,
00756                    const std::vector<double> &d,
00757                    const std::vector<double> &k);
00758
00759 /**
00760 * @ingroup ForceControl
00761 * \chinese
00762 *
00763 *
00764 * @param thresholds:
00765 *
00766 * @return 0
00767 * AUBO_BUSY
00768 * AUBO_BAD_STATE
00769 * -AUBO_INVL_ARGUMENT
00770 * -AUBO_BAD_STATE
00771 *
00772 * @throws arcs::common_interface::AuboException
00773 *
00774 * @par Python
00775 * fcSetSensorThresholds(self: pyaubo_sdk.ForceControl, arg0:
00776 * List[float]) -> int
00777 *
00778 * @par Lua
00779 * fcSetSensorThresholds(thresholds: table) -> nil
00780 *
00781 * @par Lua
00782 * fcSetSensorThresholds({1.0,1.0,1.0,0.5,0.5,0.5})
00783 *
00784 * \endch
00785 * \endchinese
00786 * \english
00787 * Set force control thresholds
00788 *
00789 * @param thresholds: Force control thresholds
00790 *
00791 * @return Return 0 if succeeded; return error code if failed
00792 * AUBO_BUSY
00793 * AUBO_BAD_STATE

```

```

00794 * -AUBO_INVL_ARGUMENT
00795 * -AUBO_BAD_STATE
00796 *
00797 * @throws arcs::common_interface::AuboException
00798 *
00799 * @par Python Function Prototype
00800 * fcSetSensorThresholds(self: pyaubo_sdk.ForceControl, arg0:
00801 * List[float]) -> int
00802 *
00803 * @par Lua Function Prototype
00804 * fcSetSensorThresholds(thresholds: table) -> nil
00805 *
00806 * @par Lua example
00807 * fcSetSensorThresholds({1.0,1.0,1.0,0.5,0.5,0.5})
00808 *
00809 * \endenglish
00810 */
00811 int fcSetSensorThresholds(const std::vector<double> &thresholds);
00812
00813 /**
00814 * @ingroup ForceControl
00815 * \chinese
00816 *
00817 *
00818 * @param limits:
00819 *
00820 * @return 0
00821 * AUBO_BUSY
00822 * AUBO_BAD_STATE
00823 * -AUBO_INVL_ARGUMENT
00824 * -AUBO_BAD_STATE
00825 *
00826 * @throws arcs::common_interface::AuboException
00827 *
00828 * @par Python
00829 * fcSetSensorLimits(self: pyaubo_sdk.ForceControl, arg0:
00830 * List[float]) -> int
00831 *
00832 * @par Lua
00833 * fcSetSensorLimits(limits: table) -> nil
00834 *
00835 * @par Lua
00836 * fcSetSensorLimits({200.0,200.0,200.0,50.0,50.0,50.0})
00837 *
00838 * \endchinese
00839 * \english
00840 * Set force control maximum force limits
00841 *
00842 * @param limits: Force limits
00843 *
00844 * @return Return 0 if succeeded; return error code if failed
00845 * AUBO_BUSY
00846 * AUBO_BAD_STATE
00847 * -AUBO_INVL_ARGUMENT
00848 * -AUBO_BAD_STATE
00849 *
00850 * @throws arcs::common_interface::AuboException
00851 *
00852 * @par Python Function Prototype
00853 * fcSetSensorLimits(self: pyaubo_sdk.ForceControl, arg0:
00854 * List[float]) -> int
00855 *
00856 * @par Lua Function Prototype
00857 * fcSetSensorLimits(limits: table) -> nil
00858 *
00859 * @par Lua example
00860 * fcSetSensorLimits({200.0,200.0,200.0,50.0,50.0,50.0})
00861 *
00862 * \endenglish
00863 */
00864 int fcSetSensorLimits(const std::vector<double> &limits);
00865
00866 /**
00867 * @ingroup ForceControl
00868 * \chinese
00869 *
00870 *
00871 * @return
00872 *
00873 * @throws arcs::common_interface::AuboException
00874 *
00875 * @par Python
00876 * getFcSensorThresholds(self: pyaubo_sdk.ForceControl) -> list
00877 *
00878 * @par Lua
00879 * getFcSensorThresholds() -> table
00880 *

```

```

00881 * @par Lua
00882 * Fc_Thresholds = getFcSensorThresholds()
00883 *
00884 * \endchinese
00885 * \english
00886 * Get force control thresholds
00887 *
00888 * @return Minimum force control thresholds
00889 *
00890 * @throws arcs::common_interface::AuboException
00891 *
00892 * @par Python Function Prototype
00893 * getFcSensorThresholds(self: pyaubo_sdk.ForceControl) -> list
00894 *
00895 * @par Lua Function Prototype
00896 * getFcSensorThresholds() -> table
00897 *
00898 * @par Lua example
00899 * Fc_Thresholds = getFcSensorThresholds()
00900 *
00901 * \endenglish
00902 */
00903 std::vector<double> getFcSensorThresholds();
00904
00905 /**
00906 * @ingroup ForceControl
00907 * \chinese
00908 *
00909 *
00910 * @return
00911 *
00912 * @throws arcs::common_interface::AuboException
00913 *
00914 * @par Python
00915 * getFcSensorLimits(self: pyaubo_sdk.ForceControl) -> list
00916 *
00917 * @par Lua
00918 * getFcSensorLimits() -> table
00919 *
00920 * @par Lua
00921 * Fc_Limits = getFcSensorLimits()
00922 *
00923 * \endchinese
00924 * \english
00925 * Get maximum force limits
00926 *
00927 * @return Force control maximum force limits
00928 *
00929 * @throws arcs::common_interface::AuboException
00930 *
00931 * @par Python Function Prototype
00932 * getFcSensorLimits(self: pyaubo_sdk.ForceControl) -> list
00933 *
00934 * @par Lua Function Prototype
00935 * getFcSensorLimits() -> table
00936 *
00937 * @par Lua example
00938 * Fc_Limits = getFcSensorLimits()
00939 *
00940 * \endenglish
00941 */
00942 std::vector<double> getFcSensorLimits();
00943
00944 /**
00945 * @ingroup ForceControl
00946 * \chinese
00947 *
00948 *
00949 * @return
00950 *
00951 * @throws arcs::common_interface::AuboException
00952 *
00953 * @par Python
00954 * getDynamicModel(self: pyaubo_sdk.ForceControl) -> Tuple[List[float],
00955 * List[float], List[float]]
00956 *
00957 * @par Lua
00958 * getDynamicModel() -> table
00959 *
00960 * @par Lua
00961 * Fc_Model = getDynamicModel()
00962 *
00963 * @par JSON-RPC
00964 * {"jsonrpc":"2.0","method":"rob1.ForceControl.getDynamicModel","params":[],"id":1}
00965 *
00966 * @par JSON-RPC
00967 * {"id":1,"jsonrpc":"2.0","result":[[],[20.0,20.0,20.0,5.0,5.0,5.0],[]]}

```

```

00968 * \endchinese
00969 * \english
00970 * Get force control dynamics model
00971 *
00972 * @return Force control dynamics model
00973 *
00974 * @throws arcs::common_interface::AuboException
00975 *
00976 * @par Python Function Prototype
00977 * getDynamicModel(self: pyaubo_sdk.ForceControl) -> Tuple[List[float],
00978 * List[float], List[float]]
00979 *
00980 * @par Lua Function Prototype
00981 * getDynamicModel() -> table
00982 *
00983 * @par Lua example
00984 * Fc_Model = getDynamicModel()
00985 *
00986 * @par JSON-RPC Request example
00987 * {"jsonrpc": "2.0", "method": "rob1.ForceControl.getDynamicModel", "params": [], "id": 1}
00988 *
00989 * @par JSON-RPC Response example
00990 * {"id": 1, "jsonrpc": "2.0", "result": [[], [20.0, 20.0, 20.0, 5.0, 5.0, 5.0], []]}
00991 * \endenglish
00992 */
00993 DynamicsModel getDynamicModel();
00994
00995 /**
00996 * @ingroup ForceControl
00997 * \chinese
00998 *
00999 *
01000 * The condition is lateractivated by calling the instruction
01001 * FCCondWaitWhile, which will wait and hold the program execution while the
01002 * specified condition is true. This allows the reference force, torque and
01003 * movement to continue until the force is outside the specified limits.
01004 *
01005 * A force condition is set up by defining minimum and maximum limits for
01006 * the force in the directions of the force control coordinate system. Once
01007 * activated with FCCondWaitWhile, the program execution will continue to
01008 * wait while the measured force is within its specified limits.
01009 *
01010 * It is possible to specify that the condition is fulfilled when the force
01011 * is outside the specified limits instead. This is done by using the switch
01012 * argument Outside. The condition on force is specified in the force
01013 * control coordinate system. This coordinate system is setup by the user in
01014 * the instruction FCAct.
01015 *
01016 * @param min /
01017 * @param max /
01018 * @param outside false
01019 * true
01020 * @param timeout
01021 * s()
01022 *
01023 * @return 0
01024 * AUBO_BUSY
01025 * AUBO_BAD_STATE
01026 * -AUBO_INVL_ARGUMENT
01027 * -AUBO_BAD_STATE
01028 *
01029 * @throws arcs::common_interface::AuboException
01030 *
01031 * @par Python
01032 * setCondForce(self: pyaubo_sdk.ForceControl, arg0: List[float], arg1:
01033 * List[float], arg2: bool, arg3: float) -> int
01034 *
01035 * @par Lua
01036 * setCondForce(min: table, max: table, outside: boolean, timeout: number)
01037 * -> nil
01038 * @par Lua
01039 * setCondForce({0.1,0.1,0.1,0.1,0.1,0.1},{50,50,50,5,5,5},{true,true,true,true,true},10)
01040 *
01041 * \endchinese
01042 * \english
01043 * Set force control termination condition: Force. When the measured force is within
01044 * the specified range, the force control algorithm will continue to run until the set
01045 * condition is not met, at which point force control will exit.
01046 *
01047 * The condition is lateractivated by calling the instruction
01048 * FCCondWaitWhile, which will wait and hold the program execution while the
01049 * specified condition is true. This allows the reference force, torque and
01050 * movement to continue until the force is outside the specified limits.
01051 *
01052 * A force condition is set up by defining minimum and maximum limits for
01053 * the force in the directions of the force control coordinate system. Once
01054 * activated with FCCondWaitWhile, the program execution will continue to

```



```

01055 * wait while the measured force is within its specified limits.
01056 *
01057 * It is possible to specify that the condition is fulfilled when the force
01058 * is outside the specified limits instead. This is done by using the switch
01059 * argument Outside. The condition on force is specified in the force
01060 * control coordinate system. This coordinate system is setup by the user in
01061 * the instruction FCAct.
01062 *
01063 * @param min Minimum force/torque in each direction
01064 * @param max Maximum force/torque in each direction
01065 * @param outside false: valid within the specified range
01066 *               true: valid outside the specified range
01067 * @param timeout
01068 * Time limit in seconds; when this time is reached from the start of force control,
01069 * force control will terminate regardless of whether the end condition is met
01070 *
01071 * @return Return 0 if succeeded; return error code if failed
01072 * AUBO_BUSY
01073 * AUBO_BAD_STATE
01074 * -AUBO_INVL_ARGUMENT
01075 * -AUBO_BAD_STATE
01076 *
01077 * @throws arcs::common_interface::AuboException
01078 *
01079 * @par Python Function Prototype
01080 * setCondForce(self: pyaubo_sdk.ForceControl, arg0: List[float], arg1:
01081 * List[float], arg2: bool, arg3: float) -> int
01082 *
01083 * @par Lua Function Prototype
01084 * setCondForce(min: table, max: table, outside: boolean, timeout: number)
01085 * -> nil
01086 * @par Lua example
01087 * setCondForce({0.1,0.1,0.1,0.1,0.1,0.1},{50,50,50,5,5,5},{true,true,true,true,true,true},10)
01088 *
01089 * \endenglish
01090 */
01091 int setCondForce(const std::vector<double> &min,
01092                 const std::vector<double> &max, bool outside,
01093                 double timeout);
01094
01095 /**
01096 * @ingroup ForceControl
01097 * \chinese
01098 *
01099 *
01100 * setCondOrient is used to set up an end condition for the tool orientation.
01101 * The condition is later activated by calling the instruction
01102 * FCCondWaitWhile, which will wait and hold the program execution while the
01103 * specified condition is true. This allows the reference force, torque and
01104 * movement to continue until the orientation is outside the specified
01105 * limits.
01106 *
01107 * An orientation condition is set up by defining a maximum angle and a
01108 * maximum rotation from a reference orientation. The reference orientation
01109 * is either defined by the current z direction of the tool, or by
01110 * specifying an orientation in relation to the z direction of the work
01111 * object.
01112 *
01113 * Once activated, the tool orientation must be within the limits (or
01114 * outside, if the argument Outside is used).
01115 *
01116 * @param frame
01117 * @param max_angle
01118 * @param max_rot
01119 * @param outside
01120 * @param timeout
01121 *
01122 * @return 0
01123 * AUBO_BUSY
01124 * AUBO_BAD_STATE
01125 * -AUBO_INVL_ARGUMENT
01126 * -AUBO_BAD_STATE
01127 *
01128 * @throws arcs::common_interface::AuboException
01129 *
01130 * @par Python
01131 * setCondOrient(self: pyaubo_sdk.ForceControl, arg0: List[float], arg1:
01132 * float, arg2: float, arg3: bool, arg4: float) -> int
01133 *
01134 * @par Lua
01135 * setCondOrient(frame: table, max_angle: number, max_rot: number, outside:
01136 * boolean, timeout: number) -> nil
01137 *
01138 * @par Lua
01139 * setCondOrient({0.1, 0.3, 0.1, 0.3142, 0.0, 1.571},30.0,20.0,true,10.0)
01140 *
01141 * \endchinese

```

```

01142 *
01143 * \english
01144 * setCondOrient is used to set up an end condition for the tool orientation.
01145 * The condition will wait and hold the program execution while the
01146 * specified condition is true. This allows the reference force, torque and
01147 * movement to continue until the orientation is outside the specified
01148 * limits.
01149 *
01150 * An orientation condition is set up by defining a maximum angle and a
01151 * maximum rotation from a reference orientation. The reference orientation
01152 * is either defined by the current z direction of the tool, or by
01153 * specifying an orientation in relation to the z direction of the work
01154 * object.
01155 *
01156 * Once activated, the tool orientation must be within the limits (or
01157 * outside, if the argument Outside is used).
01158 *
01159 * @param frame
01160 * @param max_angle
01161 * @param max_rot
01162 * @param outside
01163 * @param timeout
01164 *
01165 * @return Return 0 if succeeded; return error code if failed
01166 * AUBO_BUSY
01167 * AUBO_BAD_STATE
01168 * -AUBO_INVL_ARGUMENT
01169 * -AUBO_BAD_STATE
01170 *
01171 * @throws arcs::common_interface::AuboException
01172 *
01173 * @par Python Function Prototype
01174 * setCondOrient(self: pyaubo_sdk.ForceControl, arg0: List[float], arg1:
01175 * float, arg2: float, arg3: bool, arg4: float) -> int
01176 *
01177 * @par Lua Function Prototype
01178 * setCondOrient(frame: table, max_angle: number, max_rot: number, outside:
01179 * boolean, timeout: number) -> nil
01180 *
01181 * @par Lua example
01182 * setCondOrient({0.1, 0.3, 0.1, 0.3142, 0.0, 1.571},30.0,20.0,true,10.0)
01183 *
01184 * \endenglish
01185 */
01186 int setCondOrient(const std::vector<double> &frame, double max_angle,
01187                  double max_rot, bool outside, double timeout);
01188
01189 /**
01190 * @ingroup ForceControl
01191 * \chinese
01192 * x-y z
01193 *
01194 * @param plane={A,B,C,D}
01195 * Ax +By +Cz + D = 0
01196 * ,n = (A, B, C)
01197 * D D=0
01198 * @param timeout
01199 *
01200 * @return 0
01201 * AUBO_BUSY
01202 * AUBO_BAD_STATE
01203 * -AUBO_INVL_ARGUMENT
01204 * -AUBO_BAD_STATE
01205 *
01206 * @throws arcs::common_interface::AuboException
01207 *
01208 * @par Python
01209 * setCondPlane(self: pyaubo_sdk.ForceControl, arg0: List[float], arg1:
01210 * float) -> int
01211 *
01212 * @par Lua
01213 * setCondPlane(plane: table, timeout: number) -> nil
01214 *
01215 * @par Lua
01216 * setCondPlane({0.1, 0.3, 0.1, 0.3},10.0)
01217 *
01218 * \endchinese
01219 * \english
01220 * Specify a valid force control plane, x-y plane, z direction is valid
01221 *
01222 * @param plane={A,B,C,D}
01223 * Plane equation: Ax + By + Cz + D = 0
01224 * where n = (A, B, C) is the normal vector of the plane,
01225 * D is the distance required to move the plane to the origin (so D=0 means the plane passes through the origin)
01226 * @param timeout
01227 *
01228 * @return Return 0 if succeeded; return error code if failed

```

```

01229 * AUBO_BUSY
01230 * AUBO_BAD_STATE
01231 * -AUBO_INVL_ARGUMENT
01232 * -AUBO_BAD_STATE
01233 *
01234 * @throws arcs::common_interface::AuboException
01235 *
01236 * @par Python Function Prototype
01237 * setCondPlane(self: pyaubo_sdk.ForceControl, arg0: List[float], arg1:
01238 * float) -> int
01239 *
01240 * @par Lua Function Prototype
01241 * setCondPlane(plane: table, timeout: number) -> nil
01242 *
01243 * @par Lua example
01244 * setCondPlane({0.1, 0.3, 0.1, 0.3},10.0)
01245 *
01246 * \endenglish
01247 */
01248 int setCondPlane(const std::vector<double> &plane, double timeout);
01249
01250 /**
01251 * @ingroup ForceControl
01252 * \chinese
01253 *
01254 *
01255 * @param axis
01256 * @param radius
01257 * @param outside
01258 * @param timeout
01259 *
01260 * @return 0
01261 * AUBO_BUSY
01262 * AUBO_BAD_STATE
01263 * -AUBO_INVL_ARGUMENT
01264 * -AUBO_BAD_STATE
01265 *
01266 * @throws arcs::common_interface::AuboException
01267 *
01268 * @par Python
01269 * setCondCylinder(self: pyaubo_sdk.ForceControl, arg0: List[float], arg1:
01270 * float, arg2: bool, arg3: float) -> int
01271 *
01272 * @par Lua
01273 * setCondCylinder(axis: table, radius: number, outside: boolean, timeout:
01274 * number) -> nil
01275 *
01276 * @par Lua
01277 * setCondCylinder({0,1,0},10.0,true,5,0)
01278 *
01279 * \endchinese
01280 * \english
01281 * Specify a valid force control cylinder by providing the central axis and cylinder radius, and specify whether the inside
or outside of the cylinder is valid.
01282 *
01283 * @param axis
01284 * @param radius
01285 * @param outside
01286 * @param timeout
01287 *
01288 * @return Return 0 if succeeded; return error code if failed
01289 * AUBO_BUSY
01290 * AUBO_BAD_STATE
01291 * -AUBO_INVL_ARGUMENT
01292 * -AUBO_BAD_STATE
01293 *
01294 * @throws arcs::common_interface::AuboException
01295 *
01296 * @par Python Function Prototype
01297 * setCondCylinder(self: pyaubo_sdk.ForceControl, arg0: List[float], arg1:
01298 * float, arg2: bool, arg3: float) -> int
01299 *
01300 * @par Lua Function Prototype
01301 * setCondCylinder(axis: table, radius: number, outside: boolean, timeout:
01302 * number) -> nil
01303 *
01304 * @par Lua example
01305 * setCondCylinder({0,1,0},10.0,true,5,0)
01306 *
01307 * \endenglish
01308 */
01309 int setCondCylinder(const std::vector<double> &axis, double radius,
01310 bool outside, double timeout);
01311
01312 /**
01313 * @ingroup ForceControl
01314 * \chinese

```

```

01315 *
01316 *
01317 * @param center
01318 * @param radius
01319 * @param outside
01320 * @param timeout
01321 *
01322 * @return 0
01323 * AUBO_BUSY
01324 * AUBO_BAD_STATE
01325 * -AUBO_INVL_ARGUMENT
01326 * -AUBO_BAD_STATE
01327 *
01328 * @throws arcs::common_interface::AuboException
01329 *
01330 * @par Python
01331 * setCondSphere(self: pyaubo_sdk.ForceControl, arg0: List[float], arg1:
01332 * float, arg2: bool, arg3: float) -> int
01333 *
01334 * @par Lua
01335 * setCondSphere(center: table, radius: number, outside: boolean, timeout:
01336 * number) -> nil
01337 *
01338 * @par Lua
01339 * setCondSphere({0.2, 0.5, 0.1, 1.57, 0, 0},10.0,true,5,0)
01340 *
01341 * \endchinese
01342 * \english
01343 * Specify a valid force control sphere by providing the center and radius, and specify whether the inside or outside of
the sphere is valid.
01344 *
01345 * @param center
01346 * @param radius
01347 * @param outside
01348 * @param timeout
01349 *
01350 * @return Return 0 if succeeded; return error code if failed
01351 * AUBO_BUSY
01352 * AUBO_BAD_STATE
01353 * -AUBO_INVL_ARGUMENT
01354 * -AUBO_BAD_STATE
01355 *
01356 * @throws arcs::common_interface::AuboException
01357 *
01358 * @par Python Function Prototype
01359 * setCondSphere(self: pyaubo_sdk.ForceControl, arg0: List[float], arg1:
01360 * float, arg2: bool, arg3: float) -> int
01361 *
01362 * @par Lua Function Prototype
01363 * setCondSphere(center: table, radius: number, outside: boolean, timeout:
01364 * number) -> nil
01365 *
01366 * @par Lua example
01367 * setCondSphere({0.2, 0.5, 0.1, 1.57, 0, 0},10.0,true,5,0)
01368 *
01369 * \endenglish
01370 */
01371 int setCondSphere(const std::vector<double> &center, double radius,
01372                  bool outside, double timeout);
01373
01374 /**
01375 * @ingroup ForceControl
01376 * \chinese
01377 * TCP          FCCondWaitWhile
01378 *
01379 *
01380 *
01381 * TCP          TCP
01382 * FCCondWaitWhile
01383 *
01384 *          outside
01385 * TCP
01386 *
01387 * @param min
01388 * @param max
01389 * @param outside
01390 * @param timeout
01391 *
01392 * @return 0
01393 * AUBO_BUSY
01394 * AUBO_BAD_STATE
01395 * -AUBO_INVL_ARGUMENT
01396 * -AUBO_BAD_STATE
01397 *
01398 * @throws arcs::common_interface::AuboException
01399 *
01400 * @par Python

```

```

01401 * setCondTcpSpeed(self: pyaubo_sdk.ForceControl, arg0: List[float], arg1:
01402 * List[float], arg2: bool, arg3: float) -> int
01403 *
01404 * @par Lua
01405 * setCondTcpSpeed(min: table, max: table, outside: boolean, timeout:
01406 * number) -> nil
01407 *
01408 * @par Lua
01409 * setCondTcpSpeed({0.2, 0.2, 0.2, 0.2, 0.2, 0.2},{1.2, 1.2, 1.2, 1.2, 1.2, 1.2},true,5,0)
01410 *
01411 * \endchinese
01412 * \english
01413 * setCondTcpSpeed is used to setup an end condition for the TCP speed. The
01414 * condition is later activated by calling the instruction FCCondWaitWhile,
01415 * which will wait and hold the program execution while the specified
01416 * condition is true. This allows the reference force, torque and movement
01417 * to continue until the speed is outside the specified limits.
01418 *
01419 * A TCP speed condition is set up by defining minimum and maximum limits
01420 * for the TCP speed in all directions of the work object. Once activated
01421 * with FCCondWaitWhile, the program execution will continue to wait while
01422 * the measured speed is within its specified limits.
01423 *
01424 * It is possible to specify that the condition is fulfilled when the speed
01425 * is outside the specified limits instead. This is done by using the
01426 * switch argument Outside. The condition on TCP speed is specified in the
01427 * work object coordinate system.
01428 *
01429 * @param min
01430 * @param max
01431 * @param outside
01432 * @param timeout
01433 *
01434 * @return Return 0 if succeeded; return error code if failed
01435 * AUBO_BUSY
01436 * AUBO_BAD_STATE
01437 * -AUBO_INVL_ARGUMENT
01438 * -AUBO_BAD_STATE
01439 *
01440 * @throws arcs::common_interface::AuboException
01441 *
01442 * @par Python Function Prototype
01443 * setCondTcpSpeed(self: pyaubo_sdk.ForceControl, arg0: List[float], arg1:
01444 * List[float], arg2: bool, arg3: float) -> int
01445 *
01446 * @par Lua Function Prototype
01447 * setCondTcpSpeed(min: table, max: table, outside: boolean, timeout:
01448 * number) -> nil
01449 *
01450 * @par Lua example
01451 * setCondTcpSpeed({0.2, 0.2, 0.2, 0.2, 0.2, 0.2},{1.2, 1.2, 1.2, 1.2, 1.2, 1.2},true,5,0)
01452 *
01453 * \endenglish
01454 */
01455 int setCondTcpSpeed(const std::vector<double> &min,
01456                    const std::vector<double> &max, bool outside,
01457                    double timeout);
01458
01459 /**
01460 * @ingroup ForceControl
01461 * \chinese
01462 * -
01463 *
01464 * @param distance
01465 * @param timeout
01466 *
01467 * @return 0
01468 * AUBO_BUSY
01469 * AUBO_BAD_STATE
01470 * -AUBO_INVL_ARGUMENT
01471 * -AUBO_BAD_STATE
01472 *
01473 * @throws arcs::common_interface::AuboException
01474 *
01475 * @par Lua
01476 * setCondDistance(distance: number, timeout: number)
01477 *
01478 * @par Lua
01479 * setCondDistance(0.2, 10.0)
01480 *
01481 * \endchinese
01482 * \english
01483 * Force control termination condition - distance
01484 *
01485 * @param distance Distance
01486 * @param timeout Timeout
01487 *
01488

```

```

01488 * @return Return 0 if succeeded; return error code if failed
01489 * AUBO_BUSY
01490 * AUBO_BAD_STATE
01491 * -AUBO_INVL_ARGUMENT
01492 * -AUBO_BAD_STATE
01493 *
01494 * @throws arcs::common_interface::AuboException
01495 *
01496 * @par Lua Function Prototype
01497 * setCondDistance(distance: number, timeout: number)
01498 *
01499 * @par Lua example
01500 * setCondDistance(0.2, 10.0)
01501 *
01502 * \endenglish
01503 */
01504 int setCondDistance(double distance, double timeout);
01505
01506 /**
01507 * @ingroup ForceControl
01508 * \chinese
01509 *
01510 *
01511 * @param type
01512 * @param args
01513 * @param timeout
01514 *
01515 * @return 0
01516 * AUBO_BUSY
01517 * AUBO_BAD_STATE
01518 * -AUBO_INVL_ARGUMENT
01519 * -AUBO_BAD_STATE
01520 *
01521 * @throws arcs::common_interface::AuboException
01522 *
01523 * @par Lua
01524 * setCondAdvanced(type: number, args: table, timeout: number)
01525 *
01526 * @par Lua
01527 * setCondAdvanced(1, {0,1,0,0,0,0},10)
01528 *
01529 * \endchinese
01530 * \english
01531 * Advanced force control termination condition
01532 *
01533 * @param type Type
01534 * @param args Arguments
01535 * @param timeout Timeout
01536 *
01537 * @return Return 0 if succeeded; return error code if failed
01538 * AUBO_BUSY
01539 * AUBO_BAD_STATE
01540 * -AUBO_INVL_ARGUMENT
01541 * -AUBO_BAD_STATE
01542 *
01543 * @throws arcs::common_interface::AuboException
01544 *
01545 * @par Lua Function Prototype
01546 * setCondAdvanced(type: number, args: table, timeout: number)
01547 *
01548 * @par Lua example
01549 * setCondAdvanced(1, {0,1,0,0,0,0},10)
01550 *
01551 * \endenglish
01552 */
01553 int setCondAdvanced(const std::string &type,
01554                    const std::vector<double> &args, double timeout);
01555
01556 /**
01557 * @ingroup ForceControl
01558 * \chinese
01559 *
01560 *
01561 * @return 0
01562 * AUBO_BUSY
01563 * AUBO_BAD_STATE
01564 * -AUBO_BAD_STATE
01565 *
01566 * @throws arcs::common_interface::AuboException
01567 *
01568 * @par Python
01569 * setCondActive(self: pyaubo_sdk.ForceControl) -> int
01570 *
01571 * @par Lua
01572 * setCondActive() -> nil
01573 *
01574 * @par Lua

```

```

01575     * setCondActive()
01576     *
01577     * @par JSON-RPC
01578     * {"jsonrpc":"2.0","method":"rob1.ForceControl.setCondActive","params":[],"id":1}
01579     *
01580     * @par JSON-RPC
01581     * {"id":1,"jsonrpc":"2.0","result":0}
01582     * \endchinese
01583     * \english
01584     * Activate force control termination condition
01585     *
01586     * @return Return 0 if succeeded; return error code if failed
01587     * AUBO_BUSY
01588     * AUBO_BAD_STATE
01589     * -AUBO_BAD_STATE
01590     *
01591     * @throws arcs::common_interface::AuboException
01592     *
01593     * @par Python Function Prototype
01594     * setCondActive(self: pyaubo_sdk.ForceControl) -> int
01595     *
01596     * @par Lua Function Prototype
01597     * setCondActive() -> nil
01598     *
01599     * @par Lua example
01600     * setCondActive()
01601     *
01602     * @par JSON-RPC Request example
01603     * {"jsonrpc":"2.0","method":"rob1.ForceControl.setCondActive","params":[],"id":1}
01604     *
01605     * @par JSON-RPC Response example
01606     * {"id":1,"jsonrpc":"2.0","result":0}
01607     * \endenglish
01608     */
01609 int setCondActive();
01610
01611 /**
01612  * @ingroup ForceControl
01613  * \chinese
01614  *
01615  *
01616  * @return
01617  *
01618  * @throws arcs::common_interface::AuboException
01619  *
01620  * @par Python
01621  * isCondFullfiled(self: pyaubo_sdk.ForceControl) -> bool
01622  *
01623  * @par Lua
01624  * isCondFullfiled() -> boolean
01625  *
01626  * @par Lua
01627  * status = isCondFullfiled()
01628  *
01629  * @par JSON-RPC
01630  * {"jsonrpc":"2.0","method":"rob1.ForceControl.isCondFullfiled","params":[],"id":1}
01631  *
01632  * @par JSON-RPC
01633  * {"id":1,"jsonrpc":"2.0","result":false}
01634  * \endchinese
01635  * \english
01636  * Check if the force control termination condition has been fulfilled
01637  *
01638  * @return
01639  *
01640  * @throws arcs::common_interface::AuboException
01641  *
01642  * @par Python Function Prototype
01643  * isCondFullfiled(self: pyaubo_sdk.ForceControl) -> bool
01644  *
01645  * @par Lua Function Prototype
01646  * isCondFullfiled() -> boolean
01647  *
01648  * @par Lua example
01649  * status = isCondFullfiled()
01650  *
01651  * @par JSON-RPC Request example
01652  * {"jsonrpc":"2.0","method":"rob1.ForceControl.isCondFullfiled","params":[],"id":1}
01653  *
01654  * @par JSON-RPC Response example
01655  * {"id":1,"jsonrpc":"2.0","result":false}
01656  * \endenglish
01657  */
01658 bool isCondFullfiled();
01659
01660 /**
01661  * @ingroup ForceControl

```

```

01662 * \chinese
01663 * setSupvForce          FCAct
01664 *
01665 *
01666 *
01667 *          FCAct
01668 *
01669 * @param min
01670 * @param max
01671 *
01672 * @return 0
01673 * AUBO_BUSY
01674 * AUBO_BAD_STATE
01675 * -AUBO_INVL_ARGUMENT
01676 * -AUBO_BAD_STATE
01677 *
01678 * @throws arcs::common_interface::AuboException
01679 *
01680 * @par Python
01681 * setSupvForce(self: pyaubo_sdk.ForceControl, arg0: List[float], arg1:
01682 * List[float]) -> int
01683 *
01684 * @par Lua
01685 * setSupvForce(min: table, max: table) -> nil
01686 *
01687 * @par Lua
01688 * setSupvForce({0.1 ,0.1 ,0.1 ,0.1 ,0.1 ,0.1}, {10.0 ,10.0 ,10.0 ,10.0 ,10.0 ,10.0})
01689 *
01690 * \endchinese
01691 * \english
01692 * setSupvForce is used to set up force supervision in Force Control. The
01693 * supervision is activated when Force Control is activated with the
01694 * instruction FCAct.
01695 *
01696 * The force supervision is set up by defining minimum and maximum limits
01697 * for the force in the directions of the force control coordinate system.
01698 * Once activated, the supervision will stop the execution if the force is
01699 * outside the allowed values. The force supervision is specified in the
01700 * force control coordinate system. This coordinate system is setup by the
01701 * user with the instruction FCAct.
01702 *
01703 * @param min
01704 * @param max
01705 *
01706 * @return Return 0 if succeeded; return error code if failed
01707 * AUBO_BUSY
01708 * AUBO_BAD_STATE
01709 * -AUBO_INVL_ARGUMENT
01710 * -AUBO_BAD_STATE
01711 *
01712 * @throws arcs::common_interface::AuboException
01713 *
01714 * @par Python Function Prototype
01715 * setSupvForce(self: pyaubo_sdk.ForceControl, arg0: List[float], arg1:
01716 * List[float]) -> int
01717 *
01718 * @par Lua Function Prototype
01719 * setSupvForce(min: table, max: table) -> nil
01720 *
01721 * @par Lua example
01722 * setSupvForce({0.1 ,0.1 ,0.1 ,0.1 ,0.1 ,0.1}, {10.0 ,10.0 ,10.0 ,10.0 ,10.0 ,10.0})
01723 *
01724 * \endenglish
01725 */
01726 int setSupvForce(const std::vector<double> &min,
01727                 const std::vector<double> &max);
01728
01729 /**
01730 * @ingroup ForceControl
01731 * \chinese
01732 * setSupvOrient
01733 *          FCAct
01734 *
01735 *
01736 *          z          z
01737 *
01738 *
01739 *
01740 * @param frame
01741 * @param max_angle
01742 * @param max_rot
01743 * @param outside
01744 *
01745 * @return 0
01746 * AUBO_BUSY
01747 * AUBO_BAD_STATE
01748 * -AUBO_INVL_ARGUMENT

```



```

01749 * -AUBO_BAD_STATE
01750 *
01751 * @throws arcs::common_interface::AuboException
01752 *
01753 * @par Python
01754 * setSupvOrient(self: pyaubo_sdk.ForceControl, arg0: List[float], arg1:
01755 * float, arg2: float, arg3: bool) -> int
01756 *
01757 * @par Lua
01758 * setSupvOrient(frame: table, max_angle: number, max_rot: number,
01759 * outside: boolean) -> nil
01760 * \endchinese
01761 * \english
01762 * setSupvOrient is used to set up an supervision for the tool orientation.
01763 * The supervision is activated when Force Control is activated with the
01764 * instruction FCAct.
01765 *
01766 * An orientation supervision is set up by defining a maximum angle and a
01767 * maximum rotation from a reference orientation. The reference orientation
01768 * is either defined by the current z direction of the tool, or by
01769 * specifying an orientation in relation to the z direction of the work
01770 * object.
01771 *
01772 * Once activated, the tool orientation must be within the limits otherwise
01773 * the supervision will stop the execution.
01774 *
01775 * @param frame
01776 * @param max_angle
01777 * @param max_rot
01778 * @param outside
01779 *
01780 * @return Return 0 if succeeded; return error code if failed
01781 * AUBO_BUSY
01782 * AUBO_BAD_STATE
01783 * -AUBO_INVL_ARGUMENT
01784 * -AUBO_BAD_STATE
01785 *
01786 * @throws arcs::common_interface::AuboException
01787 *
01788 * @par Python Function Prototype
01789 * setSupvOrient(self: pyaubo_sdk.ForceControl, arg0: List[float], arg1:
01790 * float, arg2: float, arg3: bool) -> int
01791 *
01792 * @par Lua Function Prototype
01793 * setSupvOrient(frame: table, max_angle: number, max_rot: number,
01794 * outside: boolean) -> nil
01795 * \endenglish
01796 */
01797 int setSupvOrient(const std::vector<double> &frame, double max_angle,
01798                  double max_rot, bool outside);
01799
01800 /**
01801 * @ingroup ForceControl
01802 * \chinese
01803 * setSupvPosBox TCP FCAct
01804 * TCP TCP
01805 *
01806 * @param frame
01807 * @param box
01808 *
01809 * @return 0
01810 * AUBO_BUSY
01811 * AUBO_BAD_STATE
01812 * -AUBO_INVL_ARGUMENT
01813 * -AUBO_BAD_STATE
01814 *
01815 * @throws arcs::common_interface::AuboException
01816 *
01817 * @par Python
01818 * setSupvPosBox(self: pyaubo_sdk.ForceControl, arg0: List[float], arg1:
01819 * List[float[6]]) -> int
01820 *
01821 * @par Lua
01822 * setSupvPosBox(frame: table, box: table) -> nil
01823 * \endchinese
01824 * \english
01825 * setSupvPosBox is used to set up position supervision in Force Control.
01826 * Supervision is activated when Force Control is activated with the
01827 * instruction FCAct. Position supervision is set up by defining a volume in
01828 * space for the TCP. Once activated, the supervision will stop the
01829 * execution if the TCP is outside this volume.
01830 *
01831 * @param frame
01832 * @param box
01833 *
01834 * @return Return 0 if succeeded; return error code if failed
01835 * AUBO_BUSY

```

```

01836 * AUBO_BAD_STATE
01837 * -AUBO_INVL_ARGUMENT
01838 * -AUBO_BAD_STATE
01839 *
01840 * @throws arcs::common_interface::AuboException
01841 *
01842 * @par Python Function Prototype
01843 * setSupvPosBox(self: pyaubo_sdk.ForceControl, arg0: List[float], arg1:
01844 * List[float[6]]) -> int
01845 *
01846 * @par Lua Function Prototype
01847 * setSupvPosBox(frame: table, box: table) -> nil
01848 * \endenglish
01849 */
01850 int setSupvPosBox(const std::vector<double> &frame, const Box &box);
01851
01852 /**
01853 * @ingroup ForceControl
01854 * \chinese
01855 *
01856 * @param frame
01857 * @param cylinder
01858 *
01859 * @return 0
01860 * AUBO_BUSY
01861 * AUBO_BAD_STATE
01862 * -AUBO_INVL_ARGUMENT
01863 * -AUBO_BAD_STATE
01864 *
01865 * @throws arcs::common_interface::AuboException
01866 *
01867 * @par Python
01868 * setSupvPosCylinder(self: pyaubo_sdk.ForceControl, arg0: List[float],
01869 * arg1: List[float[5]]) -> int
01870 *
01871 * @par Lua
01872 * setSupvPosCylinder(frame: table, cylinder: table) -> nil
01873 *
01874 * \endchinese
01875 * \english
01876 *
01877 * @param frame
01878 * @param cylinder
01879 *
01880 * @return Return 0 if succeeded; return error code if failed
01881 * AUBO_BUSY
01882 * AUBO_BAD_STATE
01883 * -AUBO_INVL_ARGUMENT
01884 * -AUBO_BAD_STATE
01885 *
01886 * @throws arcs::common_interface::AuboException
01887 *
01888 * @par Python Function Prototype
01889 * setSupvPosCylinder(self: pyaubo_sdk.ForceControl, arg0: List[float],
01890 * arg1: List[float[5]]) -> int
01891 *
01892 * @par Lua Function Prototype
01893 * setSupvPosCylinder(frame: table, cylinder: table) -> nil
01894 *
01895 * \endenglish
01896 */
01897 int setSupvPosCylinder(const std::vector<double> &frame,
01898                       const Cylinder &cylinder);
01899
01900 /**
01901 * @ingroup ForceControl
01902 * \chinese
01903 *
01904 * @param frame
01905 * @param sphere
01906 *
01907 * @return 0
01908 * AUBO_BUSY
01909 * AUBO_BAD_STATE
01910 * -AUBO_INVL_ARGUMENT
01911 * -AUBO_BAD_STATE
01912 *
01913 * @throws arcs::common_interface::AuboException
01914 *
01915 * @par Python
01916 * setSupvPosSphere(self: pyaubo_sdk.ForceControl, arg0: List[float], arg1:
01917 * List[float[3]]) -> int
01918 *
01919 * @par Lua
01920 * setSupvPosSphere(frame: table, sphere: table) -> nil
01921 *
01922 * \endchinese

```

```

01923 * \english
01924 *
01925 * @param frame
01926 * @param sphere
01927 *
01928 * @return Return 0 if succeeded; return error code if failed
01929 * AUBO_BUSY
01930 * AUBO_BAD_STATE
01931 * -AUBO_INVL_ARGUMENT
01932 * -AUBO_BAD_STATE
01933 *
01934 * @throws arcs::common_interface::AuboException
01935 *
01936 * @par Python Function Prototype
01937 * setSupvPosSphere(self: pyaubo_sdk.ForceControl, arg0: List[float], arg1:
01938 * List[float[3]]) -> int
01939 *
01940 * @par Lua Function Prototype
01941 * setSupvPosSphere(frame: table, sphere: table) -> nil
01942 *
01943 * \endenglish
01944 */
01945 int setSupvPosSphere(const std::vector<double> &frame,
01946                     const Sphere &sphere);
01947
01948 /**
01949 * @ingroup ForceControl
01950 * \chinese
01951 * setSupvReoriSpeed          FCAct
01952 *
01953 *
01954 *
01955 *
01956 *      FCSupvReoriSpeed  FCSupvTCPSpeed  199  FCSupvTCPSpeed
01957 *
01958 * -      TCP
01959 * -  TCP          TCP
01960 *
01961 * @param speed_limit
01962 * @param outside
01963 * @param timeout
01964 *
01965 * @return 0
01966 * AUBO_BUSY
01967 * AUBO_BAD_STATE
01968 * -AUBO_INVL_ARGUMENT
01969 * -AUBO_BAD_STATE
01970 *
01971 * @throws arcs::common_interface::AuboException
01972 *
01973 * @par Python
01974 * setSupvReoriSpeed(self: pyaubo_sdk.ForceControl, arg0: List[float], arg1: bool, arg2: float) -> int
01975 *
01976 * @par Lua
01977 * setSupvReoriSpeed(speed_limit: table, outside: boolean, timeout: number) -> nil
01978 * \endchinese
01979 * \english
01980 * setSupvReoriSpeed is used to set up reorientation speed supervision in Force Control. The supervision is activated
when Force Control is activated with the instruction FCAct.
01981 *
01982 * The reorientation speed supervision is set up by defining minimum and maximum limits for the reorientation speed
around the axis of the work object coordinate system. Once activated, the supervision will stop the execution if the values
of the reorientation speed are too high.
01983 *
01984 * There are two speed supervisions: FCSupvReoriSpeed and FCSupvTCPSpeed, which is described in section
FCSupvTCPSpeed on page 199.
01985 * Both supervisions may be required because:
01986 * - A robot axis can rotate with high speed while the TCP is stationary.
01987 * - The TCP can be far from the rotating axis and a small axis rotation may result in a high speed movement of the
TCP.
01988 *
01989 * @param speed_limit
01990 * @param outside
01991 * @param timeout
01992 *
01993 * @return Return 0 if succeeded; return error code if failed
01994 * AUBO_BUSY
01995 * AUBO_BAD_STATE
01996 * -AUBO_INVL_ARGUMENT
01997 * -AUBO_BAD_STATE
01998 *
01999 * @throws arcs::common_interface::AuboException
02000 *
02001 * @par Python Function Prototype
02002 * setSupvReoriSpeed(self: pyaubo_sdk.ForceControl, arg0: List[float], arg1: bool, arg2: float) -> int
02003 *
02004 * @par Lua Function Prototype

```

```

02005 * setSupvReoriSpeed(speed_limit: table, outside: boolean, timeout: number) -> nil
02006 * \endenglish
02007 */
02008 int setSupvReoriSpeed(const std::vector<double> &speed_limit, bool outside,
02009                     double timeout);
02010
02011 /**
02012 * @ingroup ForceControl
02013 * \chinese
02014 * setSupvTcpSpeed      TCP      FCAct
02015 * TCP
02016 *      TCP
02017 *
02018 *      FCSupvTCPSpeed  FCSupvReoriSpeed  197  FCSupvReoriSpeed
02019 *
02020 *
02021 * -      TCP
02022 * - TCP      TCP
02023 *
02024 * @param speed_limit
02025 * @param outside
02026 * @param timeout
02027 *
02028 * @return 0
02029 * AUBO_BUSY
02030 * AUBO_BAD_STATE
02031 * -AUBO_INVL_ARGUMENT
02032 * -AUBO_BAD_STATE
02033 *
02034 * @throws arcs::common_interface::AuboException
02035 *
02036 * @par Python
02037 * setSupvTcpSpeed(self: pyaubo_sdk.ForceControl, arg0: List[float], arg1: bool, arg2: float) -> int
02038 *
02039 * @par Lua
02040 * setSupvTcpSpeed(speed_limit: table, outside: boolean, timeout: number) -> nil
02041 * \endchinese
02042 * \english
02043 * setSupvTcpSpeed is used to set up TCP speed supervision in Force Control.
02044 * The supervision is activated when Force Control is activated with the
02045 * instruction FCAct. The TCP speed supervision is set up by defining
02046 * minimum and maximum limits for the TCP speed in the directions of the
02047 * work object coordinate system. Once activated, the supervision will stop
02048 * the execution if too high TCP speed values are detected.
02049 *
02050 * There are two speed supervisions: FCSupvTCPSpeed and FCSupvReoriSpeed,
02051 * which is described in section FCSupvReoriSpeed on page 197.
02052 *
02053 * Both supervisions may be required because:
02054 * - A robot axis can rotate with high speed while the TCP is stationary.
02055 * - The TCP can be far from the rotating axis and a small axis rotation may
02056 * result in a high speed movement of the TCP.
02057 *
02058 * @param speed_limit
02059 * @param outside
02060 * @param timeout
02061 *
02062 * @return Return 0 if succeeded; return error code if failed
02063 * AUBO_BUSY
02064 * AUBO_BAD_STATE
02065 * -AUBO_INVL_ARGUMENT
02066 * -AUBO_BAD_STATE
02067 *
02068 * @throws arcs::common_interface::AuboException
02069 *
02070 * @par Python Function Prototype
02071 * setSupvTcpSpeed(self: pyaubo_sdk.ForceControl, arg0: List[float], arg1: bool, arg2: float) -> int
02072 *
02073 * @par Lua Function Prototype
02074 * setSupvTcpSpeed(speed_limit: table, outside: boolean, timeout: number) -> nil
02075 * \endenglish
02076 */
02077 int setSupvTcpSpeed(const std::vector<double> &speed_limit, bool outside,
02078                   double timeout);
02079
02080 //
02081 // --- force frame filter:      /
02082 // +++ force loop filter:
02083
02084 /**
02085 * @ingroup ForceControl
02086 * \chinese
02087 *
02088 *
02089 * --- force frame filter:      /
02090 * +++ force loop filter:
02091 *

```

```

02092 * @param cutoff_freq
02093 *
02094 * @return 0
02095 * AUBO_BUSY
02096 * AUBO_BAD_STATE
02097 * -AUBO_INVL_ARGUMENT
02098 * -AUBO_BAD_STATE
02099 *
02100 * @throws arcs::common_interface::AuboException
02101 *
02102 * @par Python
02103 * setLpFilter(self: pyaubo_sdk.ForceControl, arg0: List[float]) -> int
02104 *
02105 * @par Lua
02106 * setLpFilter(cutoff_freq: table) -> nil
02107 * \endchinese
02108 * \english
02109 * Set low-pass filter
02110 *
02111 * --- force frame filter: filter measured force/torque
02112 * +++ force loop filter: filter for force control output reference speed
02113 *
02114 * FCSetLPFilterTune is used to change the response of force loop according to
02115 * the description in Damping and LP-filter on page 103.
02116 *
02117 * @param cutoff_freq Cutoff frequency
02118 *
02119 * @return Return 0 if succeeded; return error code if failed
02120 * AUBO_BUSY
02121 * AUBO_BAD_STATE
02122 * -AUBO_INVL_ARGUMENT
02123 * -AUBO_BAD_STATE
02124 *
02125 * @throws arcs::common_interface::AuboException
02126 *
02127 * @par Python Function Prototype
02128 * setLpFilter(self: pyaubo_sdk.ForceControl, arg0: List[float]) -> int
02129 *
02130 * @par Lua Function Prototype
02131 * setLpFilter(cutoff_freq: table) -> nil
02132 * \endenglish
02133 */
02134 int setLpFilter(const std::vector<double> &cutoff_freq);
02135
02136 /**
02137 * @ingroup ForceControl
02138 * \chinese
02139 *
02140 *
02141 * @return 0
02142 * AUBO_BUSY
02143 * AUBO_BAD_STATE
02144 * -AUBO_BAD_STATE
02145 *
02146 * @throws arcs::common_interface::AuboException
02147 *
02148 * @par Python
02149 * resetLpFilter(self: pyaubo_sdk.ForceControl) -> int
02150 *
02151 * @par Lua
02152 * resetLpFilter() -> nil
02153 *
02154 * @par JSON-RPC
02155 * {"jsonrpc": "2.0", "method": "rob1.ForceControl.resetLpFilter", "params": [], "id": 1}
02156 *
02157 * @par JSON-RPC
02158 * {"id": 1, "jsonrpc": "2.0", "result": 0}
02159 * \endchinese
02160 * \english
02161 * Reset low-pass filter
02162 *
02163 * @return Return 0 if succeeded; return error code if failed
02164 * AUBO_BUSY
02165 * AUBO_BAD_STATE
02166 * -AUBO_BAD_STATE
02167 *
02168 * @throws arcs::common_interface::AuboException
02169 *
02170 * @par Python Function Prototype
02171 * resetLpFilter(self: pyaubo_sdk.ForceControl) -> int
02172 *
02173 * @par Lua Function Prototype
02174 * resetLpFilter() -> nil
02175 *
02176 * @par JSON-RPC Request example
02177 * {"jsonrpc": "2.0", "method": "rob1.ForceControl.resetLpFilter", "params": [], "id": 1}
02178 *

```

```

02179     * @par JSON-RPC Response example
02180     * {"id":1,"jsonrpc":"2.0","result":0}
02181     * \endenglish
02182     */
02183     int resetLpFilter();
02184
02185     /**
02186     * @ingroup ForceControl
02187     * \chinese
02188     * speedChangeEnable    FC SpeedChange
02189     *    FC SpeedChange
02190     *
02191     * @param ref_force
02192     *
02193     * @return    0
02194     * AUBO_BUSY
02195     * AUBO_BAD_STATE
02196     * -AUBO_INVL_ARGUMENT
02197     * -AUBO_BAD_STATE
02198     *
02199     * @throws arcs::common_interface::AuboException
02200     *
02201     * @par Python
02202     * speedChangeEnable(self: pyaubo_sdk.ForceControl, arg0: float) -> int
02203     *
02204     * @par Lua
02205     * speedChangeEnable(ref_force: number) -> nil
02206     * \endchinese
02207     * \english
02208     * The speedChangeEnable is used to activate FC SpeedChange function with desired
02209     * reference and recover behavior. When FC SpeedChange function is active,
02210     * the robot speed will be reduced/increased in order to keep the measured
02211     * signal close to the reference.
02212     *
02213     * @param ref_force
02214     *
02215     * @return Return 0 if succeeded; return error code if failed
02216     * AUBO_BUSY
02217     * AUBO_BAD_STATE
02218     * -AUBO_INVL_ARGUMENT
02219     * -AUBO_BAD_STATE
02220     *
02221     * @throws arcs::common_interface::AuboException
02222     *
02223     * @par Python Function Prototype
02224     * speedChangeEnable(self: pyaubo_sdk.ForceControl, arg0: float) -> int
02225     *
02226     * @par Lua Function Prototype
02227     * speedChangeEnable(ref_force: number) -> nil
02228     * \endenglish
02229     */
02230     int speedChangeEnable(double ref_force);
02231
02232     /**
02233     * @ingroup ForceControl
02234     * \chinese
02235     *    FC SpeedChange
02236     *
02237     * @return    0
02238     * AUBO_BUSY
02239     * AUBO_BAD_STATE
02240     * -AUBO_BAD_STATE
02241     *
02242     * @throws arcs::common_interface::AuboException
02243     *
02244     * @par Python
02245     * speedChangeDisable(self: pyaubo_sdk.ForceControl) -> int
02246     *
02247     * @par Lua
02248     * speedChangeDisable() -> nil
02249     *
02250     * @par JSON-RPC
02251     * {"jsonrpc":"2.0","method":"rob1.ForceControl.speedChangeDisable","params":[],"id":1}
02252     *
02253     * @par JSON-RPC
02254     * {"id":1,"jsonrpc":"2.0","result":0}
02255     * \endchinese
02256     * \english
02257     * Deactivate FC SpeedChange function.
02258     *
02259     * @return Return 0 if succeeded; return error code if failed
02260     * AUBO_BUSY
02261     * AUBO_BAD_STATE
02262     * -AUBO_BAD_STATE
02263     *
02264     * @throws arcs::common_interface::AuboException
02265     *

```

```

02266 * @par Python Function Prototype
02267 * speedChangeDisable(self: pyaubo_sdk.ForceControl) -> int
02268 *
02269 * @par Lua Function Prototype
02270 * speedChangeDisable() -> nil
02271 *
02272 * @par JSON-RPC Request example
02273 * {"jsonrpc":"2.0","method":"rob1.ForceControl.speedChangeDisable","params":[],"id":1}
02274 *
02275 * @par JSON-RPC Response example
02276 * {"id":1,"jsonrpc":"2.0","result":0}
02277 * \endenglish
02278 */
02279 int speedChangeDisable();
02280
02281 /**
02282 * @ingroup ForceControl
02283 * \chinese
02284 * speedChangeTune    FC SpeedChange
02285 *
02286 * @param speed_levels
02287 * @param speed_ratio_min
02288 *
02289 * @return 0
02290 * AUBO_BUSY
02291 * AUBO_BAD_STATE
02292 * -AUBO_INVL_ARGUMENT
02293 * -AUBO_BAD_STATE
02294 *
02295 * @throws arcs::common_interface::AuboException
02296 *
02297 * @par Python
02298 * speedChangeTune(self: pyaubo_sdk.ForceControl, arg0: int, arg1: float) -> int
02299 *
02300 * @par Lua
02301 * speedChangeTune(speed_levels: number, speed_ratio_min: number) -> nil
02302 * \endchinese
02303 * \english
02304 * speedChangeTune is used to set FC SpeedChange system parameters to a new value.
02305 *
02306 * @param speed_levels
02307 * @param speed_ratio_min
02308 *
02309 * @return Return 0 if succeeded; return error code if failed
02310 * AUBO_BUSY
02311 * AUBO_BAD_STATE
02312 * -AUBO_INVL_ARGUMENT
02313 * -AUBO_BAD_STATE
02314 *
02315 * @throws arcs::common_interface::AuboException
02316 *
02317 * @par Python Function Prototype
02318 * speedChangeTune(self: pyaubo_sdk.ForceControl, arg0: int, arg1: float) -> int
02319 *
02320 * @par Lua Function Prototype
02321 * speedChangeTune(speed_levels: number, speed_ratio_min: number) -> nil
02322 * \endenglish
02323 */
02324 int speedChangeTune(int speed_levels, double speed_ratio_min);
02325
02326 /* Defines how many Newtons are required to make the robot move 1 m/s. The
02327    higher the value, the less responsive the robot gets.
02328    The damping can be tuned (as a percentage of the system parameter values)
02329    by the RAPID instruction FCAct. */
02330 //
02331 // [damping_fx, damping_fy, damping_fz, damping_tx, damping_ty, damping_tz]
02332 // A value between min and 10,000,000 Ns/m.
02333 // A value between minimum and 10,000,000 Nms/rad.
02334
02335 /**
02336 * @ingroup ForceControl
02337 * \chinese
02338 * setDamping          x z    255 x z    254
02339 *
02340 *          FCAct          FCSetDampingTune  FCAct
02341 *
02342 * @param damping
02343 * @param ramp_time
02344 *
02345 * @return 0
02346 * AUBO_BUSY
02347 * AUBO_BAD_STATE
02348 * -AUBO_INVL_ARGUMENT
02349 * -AUBO_BAD_STATE
02350 *
02351 * @throws arcs::common_interface::AuboException
02352 *

```

```

02353 * @par Python
02354 * setDamping(self: pyaubo_sdk.ForceControl, arg0: List[float], arg1: float) -> int
02355 *
02356 * @par Lua
02357 * setDamping(damping: table, ramp_time: number) -> nil
02358 * \endchinese
02359 * \english
02360 * setDamping is used to tune the damping in the force control coordinate systems. The parameters tuned are those
described in Damping in Torque x Direction - Damping in Torque z Direction on page 255 and Damping in Force x
Direction - Damping in Force z Direction on page 254.
02361 *
02362 * Damping can be set in the configuration file or by the instruction FCAct. The difference is that this instruction can
be used when force control is active. FCSetDampingTune tunes the actual values set by the instruction FCAct, not the
value in the configuration file.
02363 *
02364 * @param damping
02365 * @param ramp_time
02366 *
02367 * @return Return 0 if succeeded; return error code if failed
02368 * AUBO_BUSY
02369 * AUBO_BAD_STATE
02370 * -AUBO_INVL_ARGUMENT
02371 * -AUBO_BAD_STATE
02372 *
02373 * @throws arcs::common_interface::AuboException
02374 *
02375 * @par Python Function Prototype
02376 * setDamping(self: pyaubo_sdk.ForceControl, arg0: List[float], arg1: float) -> int
02377 *
02378 * @par Lua Function Prototype
02379 * setDamping(damping: table, ramp_time: number) -> nil
02380 * \endenglish
02381 */
02382 int setDamping(const std::vector<double> &damping, double ramp_time);
02383
02384 /**
02385 * @ingroup ForceControl
02386 * \chinese
02387 *
02388 *
02389 * @return 0
02390 * AUBO_BUSY
02391 * AUBO_BAD_STATE
02392 * -AUBO_BAD_STATE
02393 *
02394 * @throws arcs::common_interface::AuboException
02395 *
02396 * @par Python
02397 * resetDamping(self: pyaubo_sdk.ForceControl) -> int
02398 *
02399 * @par Lua
02400 * resetDamping() -> nil
02401 *
02402 * @par JSON-RPC
02403 * {"jsonrpc":"2.0","method":"rob1.ForceControl.resetDamping","params":[],"id":1}
02404 *
02405 * @par JSON-RPC
02406 * {"id":1,"jsonrpc":"2.0","result":0}
02407 * \endchinese
02408 * \english
02409 * Reset damping parameters
02410 *
02411 * @return Return 0 if succeeded; return error code if failed
02412 * AUBO_BUSY
02413 * AUBO_BAD_STATE
02414 * -AUBO_BAD_STATE
02415 *
02416 * @throws arcs::common_interface::AuboException
02417 *
02418 * @par Python Function Prototype
02419 * resetDamping(self: pyaubo_sdk.ForceControl) -> int
02420 *
02421 * @par Lua Function Prototype
02422 * resetDamping() -> nil
02423 *
02424 * @par JSON-RPC Request example
02425 * {"jsonrpc":"2.0","method":"rob1.ForceControl.resetDamping","params":[],"id":1}
02426 *
02427 * @par JSON-RPC Response example
02428 * {"id":1,"jsonrpc":"2.0","result":0}
02429 * \endenglish
02430 */
02431 int resetDamping();
02432
02433 /**
02434 * @ingroup ForceControl
02435 * \chinese

```



```

02436 *
02437 *
02438 * @return 0
02439 * AUBO_BUSY
02440 * AUBO_BAD_STATE
02441 * -AUBO_BAD_STATE
02442 *
02443 * @throws arcs::common_interface::AuboException
02444 *
02445 * @par JSON-RPC
02446 * {"jsonrpc":"2.0","method":"rob1.ForceControl.softFloatEnable","params":[],"id":1}
02447 *
02448 * @par JSON-RPC
02449 * {"id":1,"jsonrpc":"2.0","result":0}
02450 * \endchinese
02451 * \english
02452 * Enable soft float function.
02453 *
02454 * @return Return 0 if succeeded; return error code if failed
02455 * AUBO_BUSY
02456 * AUBO_BAD_STATE
02457 * -AUBO_BAD_STATE
02458 *
02459 * @throws arcs::common_interface::AuboException
02460 *
02461 * @par JSON-RPC Request example
02462 * {"jsonrpc":"2.0","method":"rob1.ForceControl.softFloatEnable","params":[],"id":1}
02463 *
02464 * @par JSON-RPC Response example
02465 * {"id":1,"jsonrpc":"2.0","result":0}
02466 * \endenglish
02467 */
02468 int softFloatEnable();
02469
02470 /**
02471 * @ingroup ForceControl
02472 * \chinese
02473 *
02474 *
02475 * @return 0
02476 * AUBO_BUSY
02477 * AUBO_BAD_STATE
02478 * -AUBO_BAD_STATE
02479 *
02480 * @throws arcs::common_interface::AuboException
02481 *
02482 * @par JSON-RPC
02483 * {"jsonrpc":"2.0","method":"rob1.ForceControl.softFloatDisable","params":[],"id":1}
02484 *
02485 * @par JSON-RPC
02486 * {"id":1,"jsonrpc":"2.0","result":-1}
02487 * \endchinese
02488 * \english
02489 * Disable soft float function.
02490 *
02491 * @return Return 0 if succeeded; return error code if failed
02492 * AUBO_BUSY
02493 * AUBO_BAD_STATE
02494 * -AUBO_BAD_STATE
02495 *
02496 * @throws arcs::common_interface::AuboException
02497 *
02498 * @par JSON-RPC Request example
02499 * {"jsonrpc":"2.0","method":"rob1.ForceControl.softFloatDisable","params":[],"id":1}
02500 *
02501 * @par JSON-RPC Response example
02502 * {"id":1,"jsonrpc":"2.0","result":-1}
02503 * \endenglish
02504 */
02505 int softFloatDisable();
02506
02507 /**
02508 * @ingroup ForceControl
02509 * \chinese
02510 *
02511 *
02512 * @return
02513 *
02514 * @throws arcs::common_interface::AuboException
02515 *
02516 * @par JSON-RPC
02517 * {"jsonrpc":"2.0","method":"rob1.ForceControl.isSoftFloatEnabled","params":[],"id":1}
02518 *
02519 * @par JSON-RPC
02520 * {"id":1,"jsonrpc":"2.0","result":false}
02521 * \endchinese
02522 * \english

```

```

02523 * Returns whether soft float is enabled
02524 *
02525 * @return
02526 *
02527 * @throws arcs::common_interface::AuboException
02528 *
02529 * @par JSON-RPC Request example
02530 * {"jsonrpc":"2.0","method":"rob1.ForceControl.isSoftFloatEnabled","params":[],"id":1}
02531 *
02532 * @par JSON-RPC Response example
02533 * {"id":1,"jsonrpc":"2.0","result":false}
02534 * \endenglish
02535 */
02536 bool isSoftFloatEnabled();
02537
02538 /**
02539 * @ingroup ForceControl
02540 * \chinese
02541 *
02542 *
02543 * @param joint_softfloat
02544 * @param select
02545 * @param stiff_percent
02546 * @param stiff_damp_ratio
02547 * @param force_threshold
02548 * @param force_limit
02549 * @return 0
02550 * \endchinese
02551 * \english
02552 * Set soft float parameters
02553 *
02554 * @param joint_softfloat Whether to enable soft float in joint space
02555 * @param select Select which degrees of freedom to enable soft float
02556 * @param stiff_percent Stiffness percentage
02557 * @param stiff_damp_ratio Stiffness damping ratio
02558 * @param force_threshold Force threshold
02559 * @param force_limit Force limit
02560 * @return Return 0 if succeeded, otherwise error code
02561 * \endenglish
02562 */
02563 int setSoftFloatParams(bool joint_space, const std::vector<bool> &select,
02564                       const std::vector<double> &stiff_percent,
02565                       const std::vector<double> &stiff_damp_ratio,
02566                       const std::vector<double> &force_threshold,
02567                       const std::vector<double> &force_limit);
02568
02569 /**
02570 * @ingroup ForceControl
02571 * \chinese
02572 *
02573 *
02574 * @param direction
02575 * 0
02576 * @return
02577 * 0
02578 * \endchinese
02579 * \english
02580 * Detect contact between the tool and external objects
02581 *
02582 * @param direction
02583 * Expected contact direction. If all elements are 0, detect contact in all directions.
02584 * @return
02585 * Returns the number of cycle steps back from the current point to the collision start point. If the return value is 0, no
02586 contact is detected.
02587 * \endenglish
02588 */
02589 int toolContact(const std::vector<bool> &direction);
02590
02591 /**
02592 * @ingroup ForceControl
02593 * \chinese
02594 * @brief
02595 *
02596 *
02597 * @param steps
02598 *
02599 * @return std::vector<double>
02600 *
02601 * \endchinese
02602 * \english
02603 * @brief Get historical joint positions
02604 *
02605 * According to the given number of cycle steps, go back the specified number of cycles from the joint state history to
02606 obtain the joint position data at that time.
02607 *
02608 * @param steps

```

```

02608  * Number of cycles to go back (unit: control cycles), the larger the value, the earlier the historical data is obtained
02609  * @return std::vector<double>
02610  * Joint positions (unit: radians) at the corresponding time point
02611  * \endenglish
02612  */
02613  std::vector<double> getActualJointPositionsHistory(int steps);
02614
02615  protected:
02616  void *d_{ nullptr };
02617 };
02618 using ForceControlPtr = std::shared_ptr<ForceControl>;
02619 } // namespace common_interface
02620 } // namespace arcs
02621
02622 #endif // AUBO_SDK_FORCE_CONTROL_INTERFACE_H

```

12.22 include/aubo/robot/io_control.h File Reference

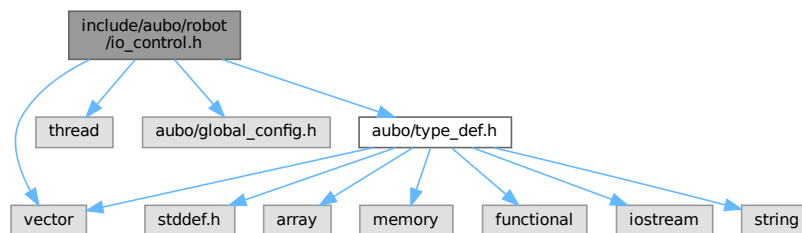
IO

```

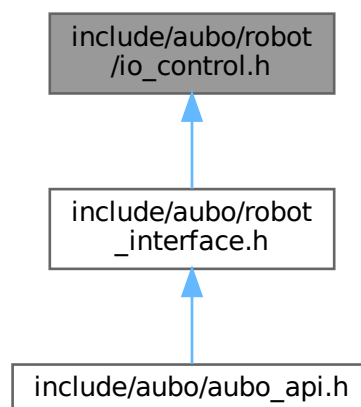
#include <vector>
#include <thread>
#include <aubo/global_config.h>
#include <aubo/type_def.h>

```

Include dependency graph for io_control.h:



This graph shows which files directly or indirectly include this file:



Classes

- class [arcs::common_interface::IoControl](#)

Namespaces

- namespace [arcs](#)
- namespace [arcs::common_interface](#)

Typedefs

- using [arcs::common_interface::IoControlPtr](#) = std::shared_ptr<[IoControl](#)>

12.22.1 Detailed Description

IO

Definition in file [io_control.h](#).

12.23 io_control.h

[Go to the documentation of this file.](#)

```

00001 /** @file io_control.h
00002  * @brief IO
00003  */
00004 #ifndef AUBO_SDK_IO_CONTROL_INTERFACE_H
00005 #define AUBO_SDK_IO_CONTROL_INTERFACE_H
00006
00007 #include <vector>
00008 #include <thread>
00009
00010 #include <aubo/global_config.h>
00011 #include <aubo/type_def.h>
00012
00013 namespace arcs {
00014 namespace common_interface {
00015
00016 /**
00017  * @defgroup IoControl IoControl (IO )
00018  * @ingroup RobotInterface
00019  * \chinese
00020  * IoControl          IO
00021  *
00022  * 1. IO  \n
00023  * 2. IO  \n
00024  * 3. IO  \n
00025  * 4. IO
00026  *
00027  *      IO  IO \n
00028  *      IO  IO \n
00029  *      IO  IO \n
00030  * \endchinese
00031  * \english
00032  * The IoControl class provides a series of interfaces for configuring and
00033  * reading the robot's standard digital and analog IO, as well as setting output
00034  * states.
00035  *
00036  * 1. Get the number of various IOs \n
00037  * 2. Configure IO input/output functions \n
00038  * 3. Configuration of configurable IOs \n
00039  * 4. Set and read the input/output range of analog IOs
00040  *
00041  * Standard digital input/output: Standard IOs on the control cabinet IO panel
00042  * \n Tool digital input/output: Digital IOs exposed via the tool-end connector
00043  * \n Configurable input/output: Can be configured as safety IO or general
00044  * digital IO \n \endenglish
00045  */

```

```

00046 class ARCS_ABI_EXPORT IoControl
00047 {
00048 public:
00049     IoControl();
00050     virtual ~IoControl();
00051
00052     /**
00053      * @ingroup IoControl
00054      * \chinese
00055      *
00056      *
00057      * @return
00058      *
00059      * @throws arcs::common_interface::AuboException
00060      *
00061      * @par Python
00062      * getStandardDigitalInputNum(self: pyaubo_sdk.IoControl) -> int
00063      *
00064      * @par Lua
00065      * getStandardDigitalInputNum() -> number
00066      *
00067      * @par Lua
00068      * num = getStandardDigitalInputNum()
00069      *
00070      * @par JSON-RPC
00071      * {"jsonrpc":"2.0","method":"rob1.IoControl.getStandardDigitalInputNum","params":[],"id":1}
00072      *
00073      * @par JSON-RPC
00074      * {"id":1,"jsonrpc":"2.0","result":16}
00075      *
00076      * \endchinese
00077      * \english
00078      * Get the number of standard digital inputs.
00079      *
00080      * @return Number of standard digital inputs.
00081      *
00082      * @throws arcs::common_interface::AuboException
00083      *
00084      * @par Python function prototype
00085      * getStandardDigitalInputNum(self: pyaubo_sdk.IoControl) -> int
00086      *
00087      * @par Lua function prototype
00088      * getStandardDigitalInputNum() -> number
00089      *
00090      * @par Lua example
00091      * num = getStandardDigitalInputNum()
00092      *
00093      * @par JSON-RPC request example
00094      * {"jsonrpc":"2.0","method":"rob1.IoControl.getStandardDigitalInputNum","params":[],"id":1}
00095      *
00096      * @par JSON-RPC response example
00097      * {"id":1,"jsonrpc":"2.0","result":16}
00098      *
00099      * \endenglish
00100      */
00101     int getStandardDigitalInputNum();
00102
00103     /**
00104      * @ingroup IoControl
00105      * \chinese
00106      * IO (      )
00107      *
00108      * @return IO (      )
00109      *
00110      * @throws arcs::common_interface::AuboException
00111      *
00112      * @par Python
00113      * getToolDigitalInputNum(self: pyaubo_sdk.IoControl) -> int
00114      *
00115      * @par Lua
00116      * getToolDigitalInputNum() -> number
00117      *
00118      * @par Lua
00119      * num = getStandardDigitalInputNum()
00120      *
00121      * @par JSON-RPC
00122      * {"jsonrpc":"2.0","method":"rob1.IoControl.getToolDigitalInputNum","params":[],"id":1}
00123      *
00124      * @par JSON-RPC
00125      * {"id":1,"jsonrpc":"2.0","result":4}
00126      *
00127      * \endchinese
00128      * \english
00129      * Get the number of tool digital IOs (including digital inputs and
00130      * outputs).
00131      *
00132      * @return Number of tool digital IOs (including digital inputs and

```

```

00133     * outputs).
00134     *
00135     * @throws arcs::common_interface::AuboException
00136     *
00137     * @par Python function prototype
00138     * getToolDigitalInputNum(self: pyaubo_sdk.IoControl) -> int
00139     *
00140     * @par Lua function prototype
00141     * getToolDigitalInputNum() -> number
00142     *
00143     * @par Lua example
00144     * num = getStandardDigitalInputNum()
00145     *
00146     * @par JSON-RPC request example
00147     * {"jsonrpc":"2.0","method":"rob1.IoControl.getToolDigitalInputNum","params":[],"id":1}
00148     *
00149     * @par JSON-RPC response example
00150     * {"id":1,"jsonrpc":"2.0","result":4}
00151     *
00152     * \endenglish
00153     */
00154 int getToolDigitalInputNum();
00155
00156 /**
00157  * @ingroup IoControl
00158  * \chinese
00159  *
00160  *
00161  * @return
00162  *
00163  * @throws arcs::common_interface::AuboException
00164  *
00165  * @par Python
00166  * getConfigurableDigitalInputNum(self: pyaubo_sdk.IoControl) -> int
00167  *
00168  * @par Lua
00169  * getConfigurableDigitalInputNum() -> number
00170  *
00171  * @par Lua
00172  * num = getConfigurableDigitalInputNum()
00173  *
00174  * @par JSON-RPC
00175  * {"jsonrpc":"2.0","method":"rob1.IoControl.getConfigurableDigitalInputNum","params":[],"id":1}
00176  *
00177  * @par JSON-RPC
00178  * {"id":1,"jsonrpc":"2.0","result":16}
00179  *
00180  * \endchinese
00181  * \english
00182  * Get the number of configurable digital inputs.
00183  *
00184  * @return Number of configurable digital inputs.
00185  *
00186  * @throws arcs::common_interface::AuboException
00187  *
00188  * @par Python function prototype
00189  * getConfigurableDigitalInputNum(self: pyaubo_sdk.IoControl) -> int
00190  *
00191  * @par Lua function prototype
00192  * getConfigurableDigitalInputNum() -> number
00193  *
00194  * @par Lua example
00195  * num = getConfigurableDigitalInputNum()
00196  *
00197  * @par JSON-RPC request example
00198  * {"jsonrpc":"2.0","method":"rob1.IoControl.getConfigurableDigitalInputNum","params":[],"id":1}
00199  *
00200  * @par JSON-RPC response example
00201  * {"id":1,"jsonrpc":"2.0","result":16}
00202  *
00203  * \endenglish
00204  */
00205 int getConfigurableDigitalInputNum();
00206
00207 /**
00208  * @ingroup IoControl
00209  * \chinese
00210  *
00211  *
00212  * @return
00213  *
00214  * @throws arcs::common_interface::AuboException
00215  *
00216  * @par Python
00217  * getStandardDigitalOutputNum(self: pyaubo_sdk.IoControl) -> int
00218  *
00219  * @par Lua

```

```

00220 * getStandardDigitalOutputNum() -> number
00221 *
00222 * @par Lua
00223 * num = getStandardDigitalOutputNum()
00224 *
00225 * @par JSON-RPC
00226 * {"jsonrpc":"2.0","method":"rob1.IoControl.getStandardDigitalOutputNum","params":[],"id":1}
00227 *
00228 * @par JSON-RPC
00229 * {"id":1,"jsonrpc":"2.0","result":8}
00230 *
00231 * \endchinese
00232 * \english
00233 * Get the number of standard digital outputs.
00234 *
00235 * @return Number of standard digital outputs.
00236 *
00237 * @throws arcs::common_interface::AuboException
00238 *
00239 * @par Python function prototype
00240 * getStandardDigitalOutputNum(self: pyaubo_sdk.IoControl) -> int
00241 *
00242 * @par Lua function prototype
00243 * getStandardDigitalOutputNum() -> number
00244 *
00245 * @par Lua example
00246 * num = getStandardDigitalOutputNum()
00247 *
00248 * @par JSON-RPC request example
00249 * {"jsonrpc":"2.0","method":"rob1.IoControl.getStandardDigitalOutputNum","params":[],"id":1}
00250 *
00251 * @par JSON-RPC response example
00252 * {"id":1,"jsonrpc":"2.0","result":8}
00253 *
00254 * \endenglish
00255 */
00256 int getStandardDigitalOutputNum();
00257
00258 /**
00259 * @ingroup IoControl
00260 * \chinese
00261 * IO (      )
00262 *
00263 * @return IO (      )
00264 *
00265 * @throws arcs::common_interface::AuboException
00266 *
00267 * @par Python
00268 * getToolDigitalOutputNum(self: pyaubo_sdk.IoControl) -> int
00269 *
00270 * @par Lua
00271 * getToolDigitalOutputNum() -> number
00272 *
00273 * @par Lua
00274 * num = getToolDigitalOutputNum()
00275 *
00276 * @par JSON-RPC
00277 * {"jsonrpc":"2.0","method":"rob1.IoControl.getToolDigitalOutputNum","params":[],"id":1}
00278 *
00279 * @par JSON-RPC
00280 * {"id":1,"jsonrpc":"2.0","result":4}
00281 *
00282 * \endchinese
00283 * \english
00284 * Get the number of tool digital IOs (including digital inputs and
00285 * outputs).
00286 *
00287 * @return Number of tool digital IOs (including digital inputs and
00288 * outputs).
00289 *
00290 * @throws arcs::common_interface::AuboException
00291 *
00292 * @par Python function prototype
00293 * getToolDigitalOutputNum(self: pyaubo_sdk.IoControl) -> int
00294 *
00295 * @par Lua function prototype
00296 * getToolDigitalOutputNum() -> number
00297 *
00298 * @par Lua example
00299 * num = getToolDigitalOutputNum()
00300 *
00301 * @par JSON-RPC request example
00302 * {"jsonrpc":"2.0","method":"rob1.IoControl.getToolDigitalOutputNum","params":[],"id":1}
00303 *
00304 * @par JSON-RPC response example
00305 * {"id":1,"jsonrpc":"2.0","result":4}
00306 *

```

```

00307     * \endenglish
00308     */
00309 int getToolDigitalOutputNum();
00310
00311 /**
00312  * @ingroup IoControl
00313  * \chinese
00314  *     IO
00315  *
00316  *     IO   IO
00317  *
00318  * @param index:  IO      0
00319  *               0
00320  * @param input:  IO
00321  *               input true  IO
00322  * @return  0
00323  * AUBO_BUSY
00324  * AUBO_BAD_STATE
00325  * -AUBO_INVL_ARGUMENT
00326  * -AUBO_BAD_STATE
00327  *
00328  * @throws arcs::common_interface::AuboException
00329  *
00330  * @par Python
00331  * setToolIoInput(self: pyaubo_sdk.IoControl, arg0: int, arg1: bool) -> int
00332  *
00333  * @par Lua
00334  * setToolIoInput(index: number, input: boolean) -> nil
00335  *
00336  * @par Lua
00337  * setToolIoInput(0,true)
00338  *
00339  * @par JSON-RPC
00340  * {"jsonrpc":"2.0","method":"rob1.IoControl.setToolIoInput","params":[0,true],"id":1}
00341  *
00342  * @par JSON-RPC
00343  * {"id":1,"jsonrpc":"2.0","result":0}
00344  *
00345  * \endchinese
00346  * \english
00347  * Set the specified tool digital IO as input or output.
00348  *
00349  * Tool digital IOs are special and can be configured as input or output.
00350  *
00351  * @param index: Indicates the IO pin, starting from 0.
00352  * For example, 0 means the first pin.
00353  * @param input: Indicates whether the specified IO is input.
00354  * If input is true, set the IO as input; otherwise, set as output.
00355  * @return Returns 0 on success; error code on failure.
00356  * AUBO_BUSY
00357  * AUBO_BAD_STATE
00358  * -AUBO_INVL_ARGUMENT
00359  * -AUBO_BAD_STATE
00360  *
00361  * @throws arcs::common_interface::AuboException
00362  *
00363  * @par Python function prototype
00364  * setToolIoInput(self: pyaubo_sdk.IoControl, arg0: int, arg1: bool) -> int
00365  *
00366  * @par Lua function prototype
00367  * setToolIoInput(index: number, input: boolean) -> nil
00368  *
00369  * @par Lua example
00370  * setToolIoInput(0,true)
00371  *
00372  * @par JSON-RPC request example
00373  * {"jsonrpc":"2.0","method":"rob1.IoControl.setToolIoInput","params":[0,true],"id":1}
00374  *
00375  * @par JSON-RPC response example
00376  * {"id":1,"jsonrpc":"2.0","result":0}
00377  *
00378  * \endenglish
00379  */
00380 int setToolIoInput(int index, bool input);
00381
00382 /**
00383  * @ingroup IoControl
00384  * \chinese
00385  *     IO
00386  *
00387  * @param index:  IO      0
00388  *               0
00389  * @return  IO      true,   false
00390  *
00391  * @throws arcs::common_interface::AuboException
00392  *
00393  * @par Python

```



```

00394 * isToolIoInput(self: pyaubo_sdk.IoControl, arg0: int) -> bool
00395 *
00396 * @par Lua
00397 * isToolIoInput(index: number) -> boolean
00398 *
00399 * @par Lua
00400 * status = isToolIoInput(0)
00401 *
00402 * @par JSON-RPC
00403 * {"jsonrpc":"2.0","method":"rob1.IoControl.isToolIoInput","params":[0],"id":1}
00404 *
00405 * @par JSON-RPC
00406 * {"id":1,"jsonrpc":"2.0","result":true}
00407 *
00408 * \endchinese
00409 * \english
00410 * Determine whether the specified tool digital IO is configured as input.
00411 *
00412 * @param index: Indicates the IO pin, starting from 0.
00413 * For example, 0 means the first pin.
00414 * @return Returns true if the specified IO is input, otherwise false.
00415 *
00416 * @throws arcs::common_interface::AuboException
00417 *
00418 * @par Python function prototype
00419 * isToolIoInput(self: pyaubo_sdk.IoControl, arg0: int) -> bool
00420 *
00421 * @par Lua function prototype
00422 * isToolIoInput(index: number) -> boolean
00423 *
00424 * @par Lua example
00425 * status = isToolIoInput(0)
00426 *
00427 * @par JSON-RPC request example
00428 * {"jsonrpc":"2.0","method":"rob1.IoControl.isToolIoInput","params":[0],"id":1}
00429 *
00430 * @par JSON-RPC response example
00431 * {"id":1,"jsonrpc":"2.0","result":true}
00432 *
00433 * \endenglish
00434 */
00435 bool isToolIoInput(int index);
00436
00437 /**
00438 * @ingroup IoControl
00439 * \chinese
00440 *
00441 *
00442 * @return
00443 *
00444 * @throws arcs::common_interface::AuboException
00445 *
00446 * @par Python
00447 * getConfigurableDigitalOutputNum(self: pyaubo_sdk.IoControl) -> int
00448 *
00449 * @par Lua
00450 * getConfigurableDigitalOutputNum() -> number
00451 *
00452 * @par Lua
00453 * num = getConfigurableDigitalOutputNum()
00454 *
00455 * @par JSON-RPC
00456 * {"jsonrpc":"2.0","method":"rob1.IoControl.getConfigurableDigitalOutputNum","params":[],"id":1}
00457 *
00458 * @par JSON-RPC
00459 * {"id":1,"jsonrpc":"2.0","result":16}
00460 *
00461 * \endchinese
00462 * \english
00463 * Get the number of configurable digital outputs.
00464 *
00465 * @return Number of configurable digital outputs.
00466 *
00467 * @throws arcs::common_interface::AuboException
00468 *
00469 * @par Python function prototype
00470 * getConfigurableDigitalOutputNum(self: pyaubo_sdk.IoControl) -> int
00471 *
00472 * @par Lua function prototype
00473 * getConfigurableDigitalOutputNum() -> number
00474 *
00475 * @par Lua example
00476 * num = getConfigurableDigitalOutputNum()
00477 *
00478 * @par JSON-RPC request example
00479 * {"jsonrpc":"2.0","method":"rob1.IoControl.getConfigurableDigitalOutputNum","params":[],"id":1}
00480 *

```

```

00481 * @par JSON-RPC response example
00482 * {"id":1,"jsonrpc":"2.0","result":16}
00483 *
00484 * \endenglish
00485 */
00486 int getConfigurableDigitalOutputNum();
00487
00488 /**
00489 * @ingroup IoControl
00490 * \chinese
00491 *
00492 *
00493 * @return
00494 *
00495 * @throws arcs::common_interface::AuboException
00496 *
00497 * @par Python
00498 * getStandardAnalogInputNum(self: pyaubo_sdk.IoControl) -> int
00499 *
00500 * @par Lua
00501 * getStandardAnalogInputNum() -> number
00502 *
00503 * @par Lua
00504 * num = getStandardAnalogInputNum()
00505 *
00506 * @par JSON-RPC
00507 * {"jsonrpc":"2.0","method":"rob1.IoControl.getStandardAnalogInputNum","params":[],"id":1}
00508 *
00509 * @par JSON-RPC
00510 * {"id":1,"jsonrpc":"2.0","result":2}
00511 *
00512 * \endchinese
00513 * \english
00514 * Get the number of standard analog inputs.
00515 *
00516 * @return Number of standard analog inputs.
00517 *
00518 * @throws arcs::common_interface::AuboException
00519 *
00520 * @par Python function prototype
00521 * getStandardAnalogInputNum(self: pyaubo_sdk.IoControl) -> int
00522 *
00523 * @par Lua function prototype
00524 * getStandardAnalogInputNum() -> number
00525 *
00526 * @par Lua example
00527 * num = getStandardAnalogInputNum()
00528 *
00529 * @par JSON-RPC request example
00530 * {"jsonrpc":"2.0","method":"rob1.IoControl.getStandardAnalogInputNum","params":[],"id":1}
00531 *
00532 * @par JSON-RPC response example
00533 * {"id":1,"jsonrpc":"2.0","result":2}
00534 *
00535 * \endenglish
00536 */
00537 int getStandardAnalogInputNum();
00538
00539 /**
00540 * @ingroup IoControl
00541 * \chinese
00542 *
00543 *
00544 * @return
00545 *
00546 * @throws arcs::common_interface::AuboException
00547 *
00548 * @par Python
00549 * getToolAnalogInputNum(self: pyaubo_sdk.IoControl) -> int
00550 *
00551 * @par Lua
00552 * getToolAnalogInputNum() -> number
00553 *
00554 * @par Lua
00555 * num = getToolAnalogInputNum()
00556 *
00557 * @par JSON-RPC
00558 * {"jsonrpc":"2.0","method":"rob1.IoControl.getToolAnalogInputNum","params":[],"id":1}
00559 *
00560 * @par JSON-RPC
00561 * {"id":1,"jsonrpc":"2.0","result":2}
00562 *
00563 * \endchinese
00564 * \english
00565 * Get the number of tool analog inputs.
00566 *
00567 * @return Number of tool analog inputs.

```

```

00568      *
00569      * @throws arcs::common_interface::AuboException
00570      *
00571      * @par Python function prototype
00572      * getToolAnalogInputNum(self: pyaubo_sdk.IoControl) -> int
00573      *
00574      * @par Lua function prototype
00575      * getToolAnalogInputNum() -> number
00576      *
00577      * @par Lua example
00578      * num = getToolAnalogInputNum()
00579      *
00580      * @par JSON-RPC request example
00581      * {"jsonrpc":"2.0","method":"rob1.IoControl.getToolAnalogInputNum","params":[],"id":1}
00582      *
00583      * @par JSON-RPC response example
00584      * {"id":1,"jsonrpc":"2.0","result":2}
00585      *
00586      * \endenglish
00587      */
00588 int getToolAnalogInputNum();
00589
00590 /**
00591  * @ingroup IoControl
00592  * \chinese
00593  *
00594  *
00595  * @return
00596  *
00597  * @throws arcs::common_interface::AuboException
00598  *
00599  * @par Python
00600  * getStandardAnalogOutputNum(self: pyaubo_sdk.IoControl) -> int
00601  *
00602  * @par Lua
00603  * getStandardAnalogOutputNum() -> number
00604  *
00605  * @par Lua
00606  * num = getStandardAnalogOutputNum()
00607  *
00608  * @par JSON-RPC
00609  * {"jsonrpc":"2.0","method":"rob1.IoControl.getStandardAnalogOutputNum","params":[],"id":1}
00610  *
00611  * @par JSON-RPC
00612  * {"id":1,"jsonrpc":"2.0","result":2}
00613  *
00614  * \endchinese
00615  * \english
00616  * Get the number of standard analog outputs.
00617  *
00618  * @return Number of standard analog outputs.
00619  *
00620  * @throws arcs::common_interface::AuboException
00621  *
00622  * @par Python function prototype
00623  * getStandardAnalogOutputNum(self: pyaubo_sdk.IoControl) -> int
00624  *
00625  * @par Lua function prototype
00626  * getStandardAnalogOutputNum() -> number
00627  *
00628  * @par Lua example
00629  * num = getStandardAnalogOutputNum()
00630  *
00631  * @par JSON-RPC request example
00632  * {"jsonrpc":"2.0","method":"rob1.IoControl.getStandardAnalogOutputNum","params":[],"id":1}
00633  *
00634  * @par JSON-RPC response example
00635  * {"id":1,"jsonrpc":"2.0","result":2}
00636  *
00637  * \endenglish
00638  */
00639 int getStandardAnalogOutputNum();
00640
00641 /**
00642  * @ingroup IoControl
00643  * \chinese
00644  *
00645  *
00646  * @return
00647  *
00648  * @throws arcs::common_interface::AuboException
00649  *
00650  * @par Python
00651  * getToolAnalogOutputNum(self: pyaubo_sdk.IoControl) -> int
00652  *
00653  * @par Lua
00654  * getToolAnalogOutputNum() -> number

```

```

00655 *
00656 * @par Lua
00657 * num = getToolAnalogOutputNum()
00658 *
00659 * @par JSON-RPC
00660 * {"jsonrpc":"2.0","method":"rob1.IoControl.getToolAnalogOutputNum","params":[],"id":1}
00661 *
00662 * @par JSON-RPC
00663 * {"id":1,"jsonrpc":"2.0","result":0}
00664 *
00665 * \endchinese
00666 * \english
00667 * Get the number of tool analog outputs.
00668 *
00669 * @return Number of tool analog outputs.
00670 *
00671 * @throws arcs::common_interface::AuboException
00672 *
00673 * @par Python function prototype
00674 * getToolAnalogOutputNum(self: pyaubo_sdk.IoControl) -> int
00675 *
00676 * @par Lua function prototype
00677 * getToolAnalogOutputNum() -> number
00678 *
00679 * @par Lua example
00680 * num = getToolAnalogOutputNum()
00681 *
00682 * @par JSON-RPC request example
00683 * {"jsonrpc":"2.0","method":"rob1.IoControl.getToolAnalogOutputNum","params":[],"id":1}
00684 *
00685 * @par JSON-RPC response example
00686 * {"id":1,"jsonrpc":"2.0","result":0}
00687 *
00688 * \endenglish
00689 */
00690 int getToolAnalogOutputNum();
00691
00692 /**
00693 * @ingroup IoControl
00694 * \chinese
00695 *
00696 *
00697 * @note
00698 *
00699 *
00700 * @return 0
00701 * AUBO_BUSY
00702 * AUBO_BAD_STATE
00703 * -AUBO_BAD_STATE
00704 *
00705 * @throws arcs::common_interface::AuboException
00706 *
00707 * @par Python
00708 * setDigitalInputActionDefault(self: pyaubo_sdk.IoControl) -> int
00709 *
00710 * @par Lua
00711 * setDigitalInputActionDefault() -> nil
00712 *
00713 * @par Lua
00714 * setDigitalInputActionDefault()
00715 *
00716 * @par JSON-RPC
00717 * {"jsonrpc":"2.0","method":"rob1.IoControl.setDigitalInputActionDefault","params":[],"id":1}
00718 *
00719 * @par JSON-RPC
00720 * {"id":1,"jsonrpc":"2.0","result":0}
00721 *
00722 * \endchinese
00723 * \english
00724 * Set all digital input actions to no trigger.
00725 *
00726 * @note
00727 * When the input action is set to no trigger, setting the digital input
00728 * value to high will not trigger any robot action.
00729 *
00730 * @return Returns 0 on success; error code on failure.
00731 * AUBO_BUSY
00732 * AUBO_BAD_STATE
00733 * -AUBO_BAD_STATE
00734 *
00735 * @throws arcs::common_interface::AuboException
00736 *
00737 * @par Python function prototype
00738 * setDigitalInputActionDefault(self: pyaubo_sdk.IoControl) -> int
00739 *
00740 * @par Lua function prototype
00741 * setDigitalInputActionDefault() -> nil

```

```

00742 *
00743 * @par Lua example
00744 * setDigitalInputActionDefault()
00745 *
00746 * @par JSON-RPC request example
00747 * {"jsonrpc":"2.0","method":"rob1.IoControl.setDigitalInputActionDefault","params":[],"id":1}
00748 *
00749 * @par JSON-RPC response example
00750 * {"id":1,"jsonrpc":"2.0","result":0}
00751 *
00752 * \endenglish
00753 */
00754 int setDigitalInputActionDefault();
00755
00756 /**
00757 * @ingroup IoControl
00758 * \chinese
00759 *
00760 *
00761 * @note
00762 * (StandardInputAction::Default)
00763 * \n
00764 * \n
00765 * DI0 (StandardInputAction::Handguide)
00766 * DI0
00767 * DI0
00768 *
00769 * @param index: IO 0
00770 * 0
00771 * @param action:
00772 *
00773 * @return 0
00774 * AUBO_REQUEST_IGNORE
00775 * AUBO_BUSY
00776 * AUBO_BAD_STATE
00777 * -AUBO_INVL_ARGUMENT
00778 * -AUBO_BAD_STATE
00779 *
00780 * @throws arcs::common_interface::AuboException
00781 *
00782 * @par Python
00783 * setStandardDigitalInputAction(self: pyaubo_sdk.IoControl, arg0: int,
00784 * arg1: arcs::common_interface::StandardInputAction) -> int
00785 *
00786 * @par Lua
00787 * setStandardDigitalInputAction(index: number, action: number) -> nil
00788 *
00789 * @par Lua
00790 * setStandardDigitalInputAction(0,1)
00791 *
00792 * @par JSON-RPC
00793 * {"jsonrpc":"2.0","method":"rob1.IoControl.setStandardDigitalInputAction","params":[0,"Handguide"],"id":1}
00794 *
00795 * @par JSON-RPC
00796 * {"id":1,"jsonrpc":"2.0","result":0}
00797 *
00798 * \endchinese
00799 * \english
00800 * Set the trigger action for standard digital input.
00801 *
00802 * @note
00803 * When the input is set to no trigger action
00804 * (StandardInputAction::Default), setting the digital input value to high
00805 * will not trigger any robot action.\n When a trigger action is set,
00806 * setting the digital input value to high will trigger the corresponding
00807 * robot action.\n For example, if DI0 is set to trigger Handguide
00808 * (StandardInputAction::Handguide), setting DI0 to high will enable
00809 * hand-guiding mode. Setting DI0 to low will exit hand-guiding mode.
00810 *
00811 * @param index: Indicates the IO pin, starting from 0.
00812 * For example, 0 means the first pin.
00813 * @param action: Trigger action
00814 *
00815 * @return Returns 0 on success; error code on failure.
00816 * AUBO_REQUEST_IGNORE
00817 * AUBO_BUSY
00818 * AUBO_BAD_STATE
00819 * -AUBO_INVL_ARGUMENT
00820 * -AUBO_BAD_STATE
00821 *
00822 * @throws arcs::common_interface::AuboException
00823 *
00824 * @par Python function prototype
00825 * setStandardDigitalInputAction(self: pyaubo_sdk.IoControl, arg0: int,
00826 * arg1: arcs::common_interface::StandardInputAction) -> int
00827 *
00828 * @par Lua function prototype

```

```

00829 * setStandardDigitalInputAction(index: number, action: number) -> nil
00830 *
00831 * @par Lua example
00832 * setStandardDigitalInputAction(0,1)
00833 *
00834 * @par JSON-RPC request example
00835 * {"jsonrpc":"2.0","method":"rob1.IoControl.setStandardDigitalInputAction","params":[0,"Handguide"],"id":1}
00836 *
00837 * @par JSON-RPC response example
00838 * {"id":1,"jsonrpc":"2.0","result":0}
00839 *
00840 * \endenglish
00841 */
00842 int setStandardDigitalInputAction(int index, StandardInputAction action);
00843
00844 /**
00845 * @ingroup IoControl
00846 * \chinese
00847 *
00848 *
00849 * @note
00850 * (StandardInputAction::Default)
00851 * \n
00852 * \n
00853 * TOOL_IO[0] (StandardInputAction::Handguide)
00854 * TOOL_IO[0]
00855 * TOOL_IO[0]
00856 *
00857 * @param index: IO 0
00858 * 0
00859 * @param action:
00860 *
00861 * @return 0
00862 * AUBO_REQUEST_IGNORE
00863 * AUBO_BUSY
00864 * AUBO_BAD_STATE
00865 * -AUBO_INVL_ARGUMENT
00866 * -AUBO_BAD_STATE
00867 *
00868 * @throws arcs::common_interface::AuboException
00869 *
00870 * @par Python
00871 * setToolDigitalInputAction(self: pyaubo_sdk.IoControl, arg0: int, arg1:
00872 * arcs::common_interface::StandardInputAction) -> int
00873 *
00874 * @par Lua
00875 * setToolDigitalInputAction(index: number, action: number) -> nil
00876 *
00877 * @par Lua
00878 * setToolDigitalInputAction(0,1)
00879 *
00880 * @par JSON-RPC
00881 * {"jsonrpc":"2.0","method":"rob1.IoControl.setToolDigitalInputAction","params":[0,"Handguide"],"id":1}
00882 *
00883 * @par JSON-RPC
00884 * {"id":1,"jsonrpc":"2.0","result":0}
00885 *
00886 * \endchinese
00887 * \english
00888 * Set the trigger action for tool digital input.
00889 *
00890 * @note
00891 * When the input is set to no trigger action
00892 * (StandardInputAction::Default), setting the tool digital input value to
00893 * high will not trigger any robot action.\n When a trigger action is set,
00894 * setting the tool digital input value to high will trigger the
00895 * corresponding robot action.\n For example, if TOOL_IO[0] is set as input
00896 * and its trigger action is Handguide (StandardInputAction::Handguide),
00897 * setting TOOL_IO[0] to high will enable hand-guiding mode.
00898 * Setting TOOL_IO[0] to low will exit hand-guiding mode.
00899 *
00900 * @param index: Indicates the IO pin, starting from 0.
00901 * For example, 0 means the first pin.
00902 * @param action: Trigger action
00903 *
00904 * @return Returns 0 on success; error code on failure.
00905 * AUBO_REQUEST_IGNORE
00906 * AUBO_BUSY
00907 * AUBO_BAD_STATE
00908 * -AUBO_INVL_ARGUMENT
00909 * -AUBO_BAD_STATE
00910 *
00911 * @throws arcs::common_interface::AuboException
00912 *
00913 * @par Python function prototype
00914 * setToolDigitalInputAction(self: pyaubo_sdk.IoControl, arg0: int, arg1:
00915 * arcs::common_interface::StandardInputAction) -> int

```

```

00916  *
00917  * @par Lua function prototype
00918  * setToolDigitalInputAction(index: number, action: number) -> nil
00919  *
00920  * @par Lua example
00921  * setToolDigitalInputAction(0,1)
00922  *
00923  * @par JSON-RPC request example
00924  * {"jsonrpc":"2.0","method":"rob1.IoControl.setToolDigitalInputAction","params":[0,"Handguide"],"id":1}
00925  *
00926  * @par JSON-RPC response example
00927  * {"id":1,"jsonrpc":"2.0","result":0}
00928  *
00929  * \endenglish
00930  */
00931 int setToolDigitalInputAction(int index, StandardInputAction action);
00932
00933 /**
00934  * @ingroup IoControl
00935  * \chinese
00936  *
00937  *
00938  * @param index: IO      0
00939  *      0
00940  * @param action:
00941  *
00942  * @return 0
00943  * AUBO_REQUEST_IGNORE
00944  * AUBO_BUSY
00945  * AUBO_BAD_STATE
00946  * -AUBO_INVL_ARGUMENT
00947  * -AUBO_BAD_STATE
00948  *
00949  * @throws arcs::common_interface::AuboException
00950  *
00951  * @note
00952  * SafetyInputAction::Unassigned
00953  *
00954  * @par Python
00955  * setConfigurableDigitalInputAction(self: pyaubo_sdk.IoControl, arg0: int,
00956  * arg1: arcs::common_interface::StandardInputAction) -> int
00957  *
00958  * @par Lua
00959  * setConfigurableDigitalInputAction(index: number, action: number) -> nil
00960  *
00961  * @par Lua
00962  * setConfigurableDigitalInputAction(0,1)
00963  *
00964  * @par JSON-RPC
00965  * {"jsonrpc":"2.0","method":"rob1.IoControl.setConfigurableDigitalInputAction","params":[0,"Handguide"],"id":1}
00966  *
00967  * @par JSON-RPC
00968  * {"id":1,"jsonrpc":"2.0","result":0}
00969  *
00970  * \endchinese
00971  * \english
00972  * Set the trigger action for configurable digital input.
00973  *
00974  * @param index: Indicates the IO pin, starting from 0.
00975  * For example, 0 means the first pin.
00976  * @param action: Trigger action
00977  *
00978  * @return Returns 0 on success; error code on failure.
00979  * AUBO_REQUEST_IGNORE
00980  * AUBO_BUSY
00981  * AUBO_BAD_STATE
00982  * -AUBO_INVL_ARGUMENT
00983  * -AUBO_BAD_STATE
00984  *
00985  * @throws arcs::common_interface::AuboException
00986  *
00987  * @note This function takes effect only when the safety input action of the
00988  * configurable input is set to SafetyInputAction::Unassigned.
00989  *
00990  * @par Python function prototype
00991  * setConfigurableDigitalInputAction(self: pyaubo_sdk.IoControl, arg0: int,
00992  * arg1: arcs::common_interface::StandardInputAction) -> int
00993  *
00994  * @par Lua function prototype
00995  * setConfigurableDigitalInputAction(index: number, action: number) -> nil
00996  *
00997  * @par Lua example
00998  * setConfigurableDigitalInputAction(0,1)
00999  *
01000  * @par JSON-RPC request example
01001  * {"jsonrpc":"2.0","method":"rob1.IoControl.setConfigurableDigitalInputAction","params":[0,"Handguide"],"id":1}
01002  *

```

```

01003 * @par JSON-RPC response example
01004 * {"id":1,"jsonrpc":"2.0","result":0}
01005 *
01006 * \endenglish
01007 */
01008 int setConfigurableDigitalInputAction(int index,
01009                                     StandardInputAction action);
01010
01011 /**
01012 * @ingroup IoControl
01013 * \chinese
01014 *
01015 *
01016 * @param index: IO      0
01017 * 0
01018 * @return
01019 *
01020 * @par Python
01021 * getStandardDigitalInputAction(self: pyaubo_sdk.IoControl, arg0: int) ->
01022 * arcs::common__interface::StandardInputAction
01023 *
01024 * @par Lua
01025 * getStandardDigitalInputAction(index: number) -> number
01026 *
01027 * @par Lua
01028 * num = getStandardDigitalInputAction(0)
01029 *
01030 * @par JSON-RPC
01031 * {"jsonrpc":"2.0","method":"rob1.IoControl.getStandardDigitalInputAction","params":[0],"id":1}
01032 *
01033 * @par JSON-RPC
01034 * {"id":1,"jsonrpc":"2.0","result":"Default"}
01035 *
01036 * \endchinese
01037 * \english
01038 * Get the trigger action for standard digital input.
01039 *
01040 * @param index: Indicates the IO pin, starting from 0.
01041 * For example, 0 means the first pin.
01042 * @return Standard digital input trigger action
01043 *
01044 * @par Python function prototype
01045 * getStandardDigitalInputAction(self: pyaubo_sdk.IoControl, arg0: int) ->
01046 * arcs::common__interface::StandardInputAction
01047 *
01048 * @par Lua function prototype
01049 * getStandardDigitalInputAction(index: number) -> number
01050 *
01051 * @par Lua example
01052 * num = getStandardDigitalInputAction(0)
01053 *
01054 * @par JSON-RPC request example
01055 * {"jsonrpc":"2.0","method":"rob1.IoControl.getStandardDigitalInputAction","params":[0],"id":1}
01056 *
01057 * @par JSON-RPC response example
01058 * {"id":1,"jsonrpc":"2.0","result":"Default"}
01059 *
01060 * \endenglish
01061 */
01062 StandardInputAction getStandardDigitalInputAction(int index);
01063
01064 /**
01065 * @ingroup IoControl
01066 * \chinese
01067 *
01068 *
01069 * @param index: IO      0
01070 * 0
01071 * @return
01072 *
01073 * @par Python
01074 * getToolDigitalInputAction(self: pyaubo_sdk.IoControl, arg0: int) ->
01075 * arcs::common__interface::StandardInputAction
01076 *
01077 * @par Lua
01078 * getToolDigitalInputAction(index: number) -> number
01079 *
01080 * @par Lua
01081 * getToolDigitalInputAction(0)
01082 *
01083 * @par JSON-RPC
01084 * {"jsonrpc":"2.0","method":"rob1.IoControl.getToolDigitalInputAction","params":[0],"id":1}
01085 *
01086 * @par JSON-RPC
01087 * {"id":1,"jsonrpc":"2.0","result":"Default"}
01088 *
01089 * \endchinese

```



```

01090 * \english
01091 * Get the trigger action for tool digital input.
01092 *
01093 * @param index: Indicates the IO pin, starting from 0.
01094 * For example, 0 means the first pin.
01095 * @return Tool digital input trigger action
01096 *
01097 * @par Python function prototype
01098 * getToolDigitalInputAction(self: pyaubo_sdk.IoControl, arg0: int) ->
01099 * arcs::common_interface::StandardInputAction
01100 *
01101 * @par Lua function prototype
01102 * getToolDigitalInputAction(index: number) -> number
01103 *
01104 * @par Lua example
01105 * getToolDigitalInputAction(0)
01106 *
01107 * @par JSON-RPC request example
01108 * {"jsonrpc":"2.0","method":"rob1.IoControl.getToolDigitalInputAction","params":[0],"id":1}
01109 *
01110 * @par JSON-RPC response example
01111 * {"id":1,"jsonrpc":"2.0","result":"Default"}
01112 *
01113 * \endenglish
01114 */
01115 StandardInputAction getToolDigitalInputAction(int index);
01116
01117 /**
01118 * @ingroup IoControl
01119 * \chinese
01120 *
01121 *
01122 * @param index: IO      0
01123 * 0
01124 * @return
01125 *
01126 * @par Python
01127 * getConfigurableDigitalInputAction(self: pyaubo_sdk.IoControl, arg0: int)
01128 * -> arcs::common_interface::StandardInputAction
01129 *
01130 * @par Lua
01131 * getConfigurableDigitalInputAction(index: number) -> number
01132 *
01133 * @par Lua
01134 * num = getConfigurableDigitalInputAction(0)
01135 *
01136 * @par JSON-RPC
01137 * {"jsonrpc":"2.0","method":"rob1.IoControl.getConfigurableDigitalInputAction","params":[0],"id":1}
01138 *
01139 * @par JSON-RPC
01140 * {"id":1,"jsonrpc":"2.0","result":"Default"}
01141 *
01142 * \endchinese
01143 * \english
01144 * Get the trigger action for configurable digital input.
01145 *
01146 * @param index: Indicates the IO pin, starting from 0.
01147 * For example, 0 means the first pin.
01148 * @return Returns the input trigger action.
01149 *
01150 * @par Python function prototype
01151 * getConfigurableDigitalInputAction(self: pyaubo_sdk.IoControl, arg0: int)
01152 * -> arcs::common_interface::StandardInputAction
01153 *
01154 * @par Lua function prototype
01155 * getConfigurableDigitalInputAction(index: number) -> number
01156 *
01157 * @par Lua example
01158 * num = getConfigurableDigitalInputAction(0)
01159 *
01160 * @par JSON-RPC request example
01161 * {"jsonrpc":"2.0","method":"rob1.IoControl.getConfigurableDigitalInputAction","params":[0],"id":1}
01162 *
01163 * @par JSON-RPC response example
01164 * {"id":1,"jsonrpc":"2.0","result":"Default"}
01165 *
01166 * \endenglish
01167 */
01168 StandardInputAction getConfigurableDigitalInputAction(int index);
01169
01170 /**
01171 * @ingroup IoControl
01172 * \chinese
01173 *
01174 *
01175 * @note
01176 * (StandardOutputRunState::None)

```

```

01177 *      \n
01178 *      \n
01179 *      DO0      (StandardOutputRunState::Handguiding)
01180 *      DO0
01181 *      DO0
01182 *
01183 * @return 0
01184 * AUBO_BUSY
01185 * AUBO_BAD_STATE
01186 * -AUBO_BAD_STATE
01187 *
01188 * @throws arcs::common_interface::AuboException
01189 *
01190 * @par Python
01191 * setDigitalOutputRunstateDefault(self: pyaubo_sdk.IoControl) -> int
01192 *
01193 * @par Lua
01194 * setDigitalOutputRunstateDefault() -> nil
01195 *
01196 * @par Lua
01197 * setDigitalOutputRunstateDefault()
01198 *
01199 * @par JSON-RPC
01200 * {"jsonrpc":"2.0","method":"rob1.IoControl.setDigitalOutputRunstateDefault","params":[],"id":1}
01201 *
01202 * @par JSON-RPC
01203 * {"id":1,"jsonrpc":"2.0","result":0}
01204 *
01205 * \endchinese
01206 * \english
01207 * Set all digital output runstates to None.
01208 *
01209 * @note
01210 * When the output runstate is set to None (StandardOutputRunState::None),
01211 * users can set the digital output value.\n
01212 * When the output runstate is set, users cannot set the digital output
01213 * value, and the controller will set it automatically.\n For example, when
01214 * DO0's output runstate is set to indicate hand-guiding
01215 * (StandardOutputRunState::Handguiding), the robot enters hand-guiding mode
01216 * and DO0 will automatically become high. When the robot exits
01217 * hand-guiding, DO0 will automatically become low.
01218 *
01219 * @return Returns 0 on success; error code on failure.
01220 * AUBO_BUSY
01221 * AUBO_BAD_STATE
01222 * -AUBO_BAD_STATE
01223 *
01224 * @throws arcs::common_interface::AuboException
01225 *
01226 * @par Python function prototype
01227 * setDigitalOutputRunstateDefault(self: pyaubo_sdk.IoControl) -> int
01228 *
01229 * @par Lua function prototype
01230 * setDigitalOutputRunstateDefault() -> nil
01231 *
01232 * @par Lua example
01233 * setDigitalOutputRunstateDefault()
01234 *
01235 * @par JSON-RPC request example
01236 * {"jsonrpc":"2.0","method":"rob1.IoControl.setDigitalOutputRunstateDefault","params":[],"id":1}
01237 *
01238 * @par JSON-RPC response example
01239 * {"id":1,"jsonrpc":"2.0","result":0}
01240 *
01241 * \endenglish
01242 */
01243 int setDigitalOutputRunstateDefault();
01244
01245 /**
01246 * @ingroup IoControl
01247 * \chinese
01248 *
01249 *
01250 * @note
01251 * (StandardOutputRunState::None)
01252 * \n
01253 * \n
01254 * DO0      (StandardOutputRunState::Handguiding)
01255 * DO0
01256 * DO0
01257 *
01258 * @param index: IO      0
01259 * 0
01260 * @param runstate:
01261 *
01262 * @return 0
01263 * AUBO_REQUEST_IGNORE

```

```

01264 * AUBO_BUSY
01265 * AUBO_BAD_STATE
01266 * -AUBO_INVL_ARGUMENT
01267 * -AUBO_BAD_STATE
01268 *
01269 * @throws arcs::common_interface::AuboException
01270 *
01271 * @par Python
01272 * setStandardDigitalOutputRunstate(self: pyaubo_sdk.IoControl, arg0: int,
01273 * arg1: arcs::common_interface::StandardOutputRunState) -> int
01274 *
01275 * @par Lua
01276 * setStandardDigitalOutputRunstate(index: number, runstate: number) -> nil
01277 *
01278 * @par Lua
01279 * setStandardDigitalOutputRunstate(0,1)
01280 *
01281 * @par JSON-RPC
01282 * {"jsonrpc":"2.0","method":"rob1.IoControl.setStandardDigitalOutputRunstate","params":[0,"PowerOn"],"id":1}
01283 *
01284 * @par JSON-RPC
01285 * {"id":1,"jsonrpc":"2.0","result":0}
01286 *
01287 * \endchinese
01288 * \english
01289 * Set the runstate for standard digital output.
01290 *
01291 * @note
01292 * When the output runstate is set to None (StandardOutputRunState::None),
01293 * users can set the digital output value.\n
01294 * When the output runstate is set, users cannot set the digital output
01295 * value, and the controller will set it automatically.\n For example, when
01296 * DO0's output runstate is set to indicate hand-guiding
01297 * (StandardOutputRunState::Handguiding), the robot enters hand-guiding mode
01298 * and DO0 will automatically become high. When the robot exits
01299 * hand-guiding, DO0 will automatically become low.
01300 *
01301 * @param index: Indicates the IO pin, starting from 0.
01302 * For example, 0 means the first pin.
01303 * @param runstate: Output runstate selection
01304 *
01305 * @return Returns 0 on success; error code on failure.
01306 * AUBO_REQUEST_IGNORE
01307 * AUBO_BUSY
01308 * AUBO_BAD_STATE
01309 * -AUBO_INVL_ARGUMENT
01310 * -AUBO_BAD_STATE
01311 *
01312 * @throws arcs::common_interface::AuboException
01313 *
01314 * @par Python function prototype
01315 * setStandardDigitalOutputRunstate(self: pyaubo_sdk.IoControl, arg0: int,
01316 * arg1: arcs::common_interface::StandardOutputRunState) -> int
01317 *
01318 * @par Lua function prototype
01319 * setStandardDigitalOutputRunstate(index: number, runstate: number) -> nil
01320 *
01321 * @par Lua example
01322 * setStandardDigitalOutputRunstate(0,1)
01323 *
01324 * @par JSON-RPC request example
01325 * {"jsonrpc":"2.0","method":"rob1.IoControl.setStandardDigitalOutputRunstate","params":[0,"PowerOn"],"id":1}
01326 *
01327 * @par JSON-RPC response example
01328 * {"id":1,"jsonrpc":"2.0","result":0}
01329 *
01330 * \endenglish
01331 */
01332 int setStandardDigitalOutputRunstate(int index,
01333                                     StandardOutputRunState runstate);
01334
01335 /**
01336 * @ingroup IoControl
01337 * \chinese
01338 *
01339 *
01340 * @note
01341 * (StandardOutputRunState::None)
01342 * \n
01343 * \n
01344 * TOOL_IO[0] (StandardOutputRunState::Handguiding)
01345 * TOOL_IO[0]
01346 * TOOL_IO[0]
01347 *
01348 * @param index: IO 0
01349 * 0
01350 * @param runstate:

```

```

01351 *
01352 * @return 0
01353 * AUBO_REQUEST_IGNORE
01354 * AUBO_BUSY
01355 * AUBO_BAD_STATE
01356 * -AUBO_INVL_ARGUMENT
01357 * -AUBO_BAD_STATE
01358 *
01359 * @throws arcs::common_interface::AuboException
01360 *
01361 * @par Python
01362 * setToolDigitalOutputRunstate(self: pyaubo_sdk.IoControl, arg0: int, arg1:
01363 * arcs::common_interface::StandardOutputRunState) -> int
01364 *
01365 * @par Lua
01366 * setToolDigitalOutputRunstate(index: number, runstate: number) -> nil
01367 *
01368 * @par Lua
01369 * setToolDigitalOutputRunstate(0,1)
01370 *
01371 * @par JSON-RPC
01372 * {"jsonrpc":"2.0","method":"rob1.IoControl.setToolDigitalOutputRunstate","params":[0,"None"],"id":1}
01373 *
01374 * @par JSON-RPC
01375 * {"id":1,"jsonrpc":"2.0","result":0}
01376 *
01377 * \endchinese
01378 * \english
01379 * Set the runstate for tool digital output.
01380 *
01381 * @note
01382 * When the output runstate is set to None (StandardOutputRunState::None),
01383 * users can set the digital output value.\n
01384 * When the output runstate is set, users cannot set the digital output
01385 * value, and the controller will set it automatically.\n For example, when
01386 * TOOL_IO[0] is set as output and its runstate is set to indicate
01387 * hand-guiding (StandardOutputRunState::Handguiding), the robot enters
01388 * hand-guiding mode and TOOL_IO[0] will automatically become high. When the
01389 * robot exits hand-guiding, TOOL_IO[0] will automatically become low.
01390 *
01391 * @param index: Indicates the IO pin, starting from 0.
01392 * For example, 0 means the first pin.
01393 * @param runstate: Output runstate selection
01394 *
01395 * @return Returns 0 on success; error code on failure.
01396 * AUBO_REQUEST_IGNORE
01397 * AUBO_BUSY
01398 * AUBO_BAD_STATE
01399 * -AUBO_INVL_ARGUMENT
01400 * -AUBO_BAD_STATE
01401 *
01402 * @throws arcs::common_interface::AuboException
01403 *
01404 * @par Python function prototype
01405 * setToolDigitalOutputRunstate(self: pyaubo_sdk.IoControl, arg0: int, arg1:
01406 * arcs::common_interface::StandardOutputRunState) -> int
01407 *
01408 * @par Lua function prototype
01409 * setToolDigitalOutputRunstate(index: number, runstate: number) -> nil
01410 *
01411 * @par Lua example
01412 * setToolDigitalOutputRunstate(0,1)
01413 *
01414 * @par JSON-RPC request example
01415 * {"jsonrpc":"2.0","method":"rob1.IoControl.setToolDigitalOutputRunstate","params":[0,"None"],"id":1}
01416 *
01417 * @par JSON-RPC response example
01418 * {"id":1,"jsonrpc":"2.0","result":0}
01419 *
01420 * \endenglish
01421 */
01422 int setToolDigitalOutputRunstate(int index,
01423                                 StandardOutputRunState runstate);
01424
01425 /**
01426 * @ingroup IoControl
01427 * \chinese
01428 *
01429 *
01430 * @param index: IO      0
01431 *              0
01432 * @param runstate:
01433 *
01434 * @return 0
01435 * AUBO_REQUEST_IGNORE
01436 * AUBO_BUSY
01437 * AUBO_BAD_STATE

```

```

01438 * -AUBO_INVL_ARGUMENT
01439 * -AUBO_BAD_STATE
01440 *
01441 * @throws arcs::common_interface::AuboException
01442 *
01443 * @par Python
01444 * setConfigurableDigitalOutputRunstate(self: pyaubo_sdk.IoControl, arg0:
01445 * int, arg1: arcs::common_interface::StandardOutputRunState) -> int
01446 *
01447 * @par Lua
01448 * setConfigurableDigitalOutputRunstate(index: number, runstate: number) ->
01449 * nil
01450 *
01451 * @par Lua
01452 * setConfigurableDigitalOutputRunstate(0,1)
01453 *
01454 * @par JSON-RPC
01455 * {"jsonrpc":"2.0","method":"rob1.IoControl.setConfigurableDigitalOutputRunstate","params":[0,"None"],"id":1}
01456 *
01457 * @par JSON-RPC
01458 * {"id":1,"jsonrpc":"2.0","result":0}
01459 *
01460 * \endchinese
01461 * \english
01462 * Set the runstate for configurable digital output.
01463 *
01464 * @param index: Indicates the IO pin, starting from 0.
01465 * For example, 0 means the first pin.
01466 * @param runstate: Output runstate selection
01467 *
01468 * @return Returns 0 on success; error code on failure.
01469 * AUBO_REQUEST_IGNORE
01470 * AUBO_BUSY
01471 * AUBO_BAD_STATE
01472 * -AUBO_INVL_ARGUMENT
01473 * -AUBO_BAD_STATE
01474 *
01475 * @throws arcs::common_interface::AuboException
01476 *
01477 * @par Python function prototype
01478 * setConfigurableDigitalOutputRunstate(self: pyaubo_sdk.IoControl, arg0:
01479 * int, arg1: arcs::common_interface::StandardOutputRunState) -> int
01480 *
01481 * @par Lua function prototype
01482 * setConfigurableDigitalOutputRunstate(index: number, runstate: number) ->
01483 * nil
01484 *
01485 * @par Lua example
01486 * setConfigurableDigitalOutputRunstate(0,1)
01487 *
01488 * @par JSON-RPC request example
01489 * {"jsonrpc":"2.0","method":"rob1.IoControl.setConfigurableDigitalOutputRunstate","params":[0,"None"],"id":1}
01490 *
01491 * @par JSON-RPC response example
01492 * {"id":1,"jsonrpc":"2.0","result":0}
01493 *
01494 * \endenglish
01495 */
01496 int setConfigurableDigitalOutputRunstate(int index,
01497                                         StandardOutputRunState runstate);
01498
01499 /**
01500 * @ingroup IoControl
01501 * \chinese
01502 *
01503 *
01504 * @param index: IO      0
01505 * 0
01506 * @return
01507 *
01508 * @throws arcs::common_interface::AuboException
01509 *
01510 * @par Python
01511 * getStandardDigitalOutputRunstate(self: pyaubo_sdk.IoControl, arg0: int)
01512 * -> arcs::common_interface::StandardOutputRunState
01513 *
01514 * @par Lua
01515 * getStandardDigitalOutputRunstate(index: number) -> number
01516 *
01517 * @par Lua
01518 * num = getStandardDigitalOutputRunstate(0)
01519 *
01520 * @par JSON-RPC
01521 * {"jsonrpc":"2.0","method":"rob1.IoControl.getStandardDigitalOutputRunstate","params":[0],"id":1}
01522 *
01523 * @par JSON-RPC
01524 * {"id":1,"jsonrpc":"2.0","result":None}

```

```

01525 *
01526 * \endchinese
01527 * \english
01528 * Get the runstate for standard digital output.
01529 *
01530 * @param index: Indicates the IO pin, starting from 0.
01531 * For example, 0 means the first pin.
01532 * @return Output runstate selection
01533 *
01534 * @throws arcs::common_interface::AuboException
01535 *
01536 * @par Python function prototype
01537 * getStandardDigitalOutputRunstate(self: pyaubo_sdk.IoControl, arg0: int)
01538 * -> arcs::common_interface::StandardOutputRunState
01539 *
01540 * @par Lua function prototype
01541 * getStandardDigitalOutputRunstate(index: number) -> number
01542 *
01543 * @par Lua example
01544 * num = getStandardDigitalOutputRunstate(0)
01545 *
01546 * @par JSON-RPC request example
01547 * {"jsonrpc": "2.0", "method": "rob1.IoControl.getStandardDigitalOutputRunstate", "params": [0], "id": 1}
01548 *
01549 * @par JSON-RPC response example
01550 * {"id": 1, "jsonrpc": "2.0", "result": "None"}
01551 *
01552 * \endenglish
01553 */
01554 StandardOutputRunState getStandardDigitalOutputRunstate(int index);
01555
01556 /**
01557 * @ingroup IoControl
01558 * \chinese
01559 *
01560 *
01561 * @param index: IO      0
01562 * 0
01563 * @return
01564 *
01565 * @throws arcs::common_interface::AuboException
01566 *
01567 * @par Python
01568 * getToolDigitalOutputRunstate(self: pyaubo_sdk.IoControl, arg0: int) ->
01569 * arcs::common_interface::StandardOutputRunState
01570 *
01571 * @par Lua
01572 * getToolDigitalOutputRunstate(index: number) -> number
01573 *
01574 * @par Lua
01575 * num = getToolDigitalOutputRunstate(0)
01576 *
01577 * @par JSON-RPC
01578 * {"jsonrpc": "2.0", "method": "rob1.IoControl.getToolDigitalOutputRunstate", "params": [0], "id": 1}
01579 *
01580 * @par JSON-RPC
01581 * {"id": 1, "jsonrpc": "2.0", "result": "None"}
01582 *
01583 * \endchinese
01584 * \english
01585 * Get the runstate for tool digital output.
01586 *
01587 * @param index: Indicates the IO pin, starting from 0.
01588 * For example, 0 means the first pin.
01589 * @return Output runstate selection
01590 *
01591 * @throws arcs::common_interface::AuboException
01592 *
01593 * @par Python function prototype
01594 * getToolDigitalOutputRunstate(self: pyaubo_sdk.IoControl, arg0: int) ->
01595 * arcs::common_interface::StandardOutputRunState
01596 *
01597 * @par Lua function prototype
01598 * getToolDigitalOutputRunstate(index: number) -> number
01599 *
01600 * @par Lua example
01601 * num = getToolDigitalOutputRunstate(0)
01602 *
01603 * @par JSON-RPC request example
01604 * {"jsonrpc": "2.0", "method": "rob1.IoControl.getToolDigitalOutputRunstate", "params": [0], "id": 1}
01605 *
01606 * @par JSON-RPC response example
01607 * {"id": 1, "jsonrpc": "2.0", "result": "None"}
01608 *
01609 * \endenglish
01610 */
01611 StandardOutputRunState getToolDigitalOutputRunstate(int index);

```

```

01612
01613 /**
01614  * @ingroup IoControl
01615  * \chinese
01616  *
01617  *
01618  * @param index: IO      0
01619  *      0
01620  * @return
01621  *
01622  * @throws arcs::common_interface::AuboException
01623  *
01624  * @par Python
01625  * getConfigurableDigitalOutputRunstate(self: pyaubo_sdk.IoControl, arg0:
01626  * int)
01627  * -> arcs::common_interface::StandardOutputRunState
01628  *
01629  * @par Lua
01630  * getConfigurableDigitalOutputRunstate(index: number) -> number
01631  *
01632  * @par Lua
01633  * num = getConfigurableDigitalOutputRunstate(0)
01634  *
01635  * @par JSON-RPC
01636  * {"jsonrpc":"2.0","method":"rob1.IoControl.getConfigurableDigitalOutputRunstate","params":[0],"id":1}
01637  *
01638  * @par JSON-RPC
01639  * {"id":1,"jsonrpc":"2.0","result":None}
01640  *
01641  * \endchinese
01642  * \english
01643  * Get the runstate for configurable digital output.
01644  *
01645  * @param index: Indicates the IO pin, starting from 0.
01646  * For example, 0 means the first pin.
01647  * @return Output runstate selection
01648  *
01649  * @throws arcs::common_interface::AuboException
01650  *
01651  * @par Python function prototype
01652  * getConfigurableDigitalOutputRunstate(self: pyaubo_sdk.IoControl, arg0:
01653  * int)
01654  * -> arcs::common_interface::StandardOutputRunState
01655  *
01656  * @par Lua function prototype
01657  * getConfigurableDigitalOutputRunstate(index: number) -> number
01658  *
01659  * @par Lua example
01660  * num = getConfigurableDigitalOutputRunstate(0)
01661  *
01662  * @par JSON-RPC request example
01663  * {"jsonrpc":"2.0","method":"rob1.IoControl.getConfigurableDigitalOutputRunstate","params":[0],"id":1}
01664  *
01665  * @par JSON-RPC response example
01666  * {"id":1,"jsonrpc":"2.0","result":None}
01667  *
01668  * \endenglish
01669  */
01670 StandardOutputRunState getConfigurableDigitalOutputRunstate(int index);
01671
01672 /**
01673  * @ingroup IoControl
01674  * \chinese
01675  *
01676  *
01677  * @note
01678  *      (StandardOutputRunState::None)
01679  *      \n
01680  *      \n
01681  *      AO0      (StandardOutputRunState::Handguiding)
01682  *      AO0
01683  *      AO0      0
01684  *
01685  * @param index: IO      0
01686  *      0
01687  * @param runstate:
01688  * @return      0
01689  * AUBO_REQUEST_IGNORE
01690  * AUBO_BUSY
01691  * AUBO_BAD_STATE
01692  * -AUBO_INVL_ARGUMENT
01693  * -AUBO_BAD_STATE
01694  *
01695  * @throws arcs::common_interface::AuboException
01696  *
01697  * @par Python
01698  * setStandardAnalogOutputRunstate(self: pyaubo_sdk.IoControl, arg0: int,

```

```

01699 * arg1: arcs::common_interface::StandardOutputRunState) -> int
01700 *
01701 * @par Lua
01702 * setStandardAnalogOutputRunstate(index: number, runstate: number) -> nil
01703 *
01704 * @par Lua
01705 * setStandardAnalogOutputRunstate(0,6)
01706 *
01707 * @par JSON-RPC
01708 * {"jsonrpc":"2.0","method":"rob1.IoControl.setStandardAnalogOutputRunstate","params":[0,"None"],"id":1}
01709 *
01710 * @par JSON-RPC
01711 * {"id":1,"jsonrpc":"2.0","result":0}
01712 *
01713 * \endchinese
01714 * \english
01715 * Set the runstate for standard analog output.
01716 *
01717 * @note
01718 * When the output runstate is set to None (StandardOutputRunState::None),
01719 * users can set the analog output value.\n
01720 * When the output runstate is set, users cannot set the analog output
01721 * value, and the controller will set it automatically.\n For example, when
01722 * AO0's output runstate is set to indicate hand-guiding
01723 * (StandardOutputRunState::Handguiding), the robot enters hand-guiding mode
01724 * and AO0 will automatically become the maximum value. When the robot exits
01725 * hand-guiding, AO0 will automatically become 0.
01726 *
01727 * @param index: Indicates the IO pin, starting from 0.
01728 * For example, 0 means the first pin.
01729 * @param runstate: Output runstate selection
01730 * @return Returns 0 on success; error code on failure.
01731 * AUBO_REQUEST_IGNORE
01732 * AUBO_BUSY
01733 * AUBO_BAD_STATE
01734 * -AUBO_INVL_ARGUMENT
01735 * -AUBO_BAD_STATE
01736 *
01737 * @throws arcs::common_interface::AuboException
01738 *
01739 * @par Python function prototype
01740 * setStandardAnalogOutputRunstate(self: pyaubo_sdk.IoControl, arg0: int,
01741 * arg1: arcs::common_interface::StandardOutputRunState) -> int
01742 *
01743 * @par Lua function prototype
01744 * setStandardAnalogOutputRunstate(index: number, runstate: number) -> nil
01745 *
01746 * @par Lua example
01747 * setStandardAnalogOutputRunstate(0,6)
01748 *
01749 * @par JSON-RPC request example
01750 * {"jsonrpc":"2.0","method":"rob1.IoControl.setStandardAnalogOutputRunstate","params":[0,"None"],"id":1}
01751 *
01752 * @par JSON-RPC response example
01753 * {"id":1,"jsonrpc":"2.0","result":0}
01754 *
01755 * \endenglish
01756 */
01757 int setStandardAnalogOutputRunstate(int index,
01758                                     StandardOutputRunState runstate);
01759
01760 /**
01761 * @ingroup IoControl
01762 * \chinese
01763 *
01764 *
01765 * @param index: IO      0
01766 * 0
01767 * @param runstate:
01768 * @return 0
01769 * AUBO_REQUEST_IGNORE
01770 * AUBO_BUSY
01771 * AUBO_BAD_STATE
01772 * -AUBO_INVL_ARGUMENT
01773 * -AUBO_BAD_STATE
01774 *
01775 * @throws arcs::common_interface::AuboException
01776 *
01777 * @par Python
01778 * setToolAnalogOutputRunstate(self: pyaubo_sdk.IoControl, arg0: int, arg1:
01779 * arcs::common_interface::StandardOutputRunState) -> int
01780 *
01781 * @par Lua
01782 * setToolAnalogOutputRunstate(index: number, runstate: number) -> nil
01783 *
01784 * @par Lua
01785 * setToolAnalogOutputRunstate(0,1)

```



```

01786 *
01787 * @par JSON-RPC
01788 * {"jsonrpc":"2.0","method":"rob1.IoControl.setToolAnalogOutputRunstate","params":[0,"None"],"id":1}
01789 *
01790 * @par JSON-RPC
01791 * {"id":1,"jsonrpc":"2.0","result":0}
01792 *
01793 * \endchinese
01794 * \english
01795 * Set the runstate for tool analog output.
01796 *
01797 * @param index: Indicates the IO pin, starting from 0.
01798 * For example, 0 means the first pin.
01799 * @param runstate: Output runstate selection
01800 * @return Returns 0 on success; error code on failure.
01801 * AUBO_REQUEST_IGNORE
01802 * AUBO_BUSY
01803 * AUBO_BAD_STATE
01804 * -AUBO_INVL_ARGUMENT
01805 * -AUBO_BAD_STATE
01806 *
01807 * @throws arcs::common_interface::AuboException
01808 *
01809 * @par Python function prototype
01810 * setToolAnalogOutputRunstate(self: pyaubo_sdk.IoControl, arg0: int, arg1:
01811 * arcs::common_interface::StandardOutputRunState) -> int
01812 *
01813 * @par Lua function prototype
01814 * setToolAnalogOutputRunstate(index: number, runstate: number) -> nil
01815 *
01816 * @par Lua example
01817 * setToolAnalogOutputRunstate(0,1)
01818 *
01819 * @par JSON-RPC request example
01820 * {"jsonrpc":"2.0","method":"rob1.IoControl.setToolAnalogOutputRunstate","params":[0,"None"],"id":1}
01821 *
01822 * @par JSON-RPC response example
01823 * {"id":1,"jsonrpc":"2.0","result":0}
01824 *
01825 * \endenglish
01826 */
01827 int setToolAnalogOutputRunstate(int index, StandardOutputRunState runstate);
01828
01829 /**
01830 * @ingroup IoControl
01831 * \chinese
01832 *
01833 *
01834 * @param index: IO      0
01835 * 0
01836 * @return
01837 *
01838 * @throws arcs::common_interface::AuboException
01839 *
01840 * @par Python
01841 * getStandardAnalogOutputRunstate(self: pyaubo_sdk.IoControl, arg0: int) ->
01842 * arcs::common_interface::StandardOutputRunState
01843 *
01844 * @par Lua
01845 * getStandardAnalogOutputRunstate(index: number) -> number
01846 *
01847 * @par Lua
01848 * num = getStandardAnalogOutputRunstate(0)
01849 *
01850 * @par JSON-RPC
01851 * {"jsonrpc":"2.0","method":"rob1.IoControl.getStandardAnalogOutputRunstate","params":[0],"id":1}
01852 *
01853 * @par JSON-RPC
01854 * {"id":1,"jsonrpc":"2.0","result":"None"}
01855 *
01856 * \endchinese
01857 * \english
01858 * Get the runstate for standard analog output.
01859 *
01860 * @param index: Indicates the IO pin, starting from 0.
01861 * For example, 0 means the first pin.
01862 * @return Standard analog output runstate selection
01863 *
01864 * @throws arcs::common_interface::AuboException
01865 *
01866 * @par Python function prototype
01867 * getStandardAnalogOutputRunstate(self: pyaubo_sdk.IoControl, arg0: int) ->
01868 * arcs::common_interface::StandardOutputRunState
01869 *
01870 * @par Lua function prototype
01871 * getStandardAnalogOutputRunstate(index: number) -> number
01872 *

```

```

01873 * @par Lua example
01874 * num = getStandardAnalogOutputRunstate(0)
01875 *
01876 * @par JSON-RPC request example
01877 * {"jsonrpc":"2.0","method":"rob1.IoControl.getStandardAnalogOutputRunstate","params":[0],"id":1}
01878 *
01879 * @par JSON-RPC response example
01880 * {"id":1,"jsonrpc":"2.0","result":"None"}
01881 *
01882 * \endenglish
01883 */
01884 StandardOutputRunState getStandardAnalogOutputRunstate(int index);
01885
01886 /**
01887 * @ingroup IoControl
01888 * \chinese
01889 *
01890 *
01891 * @param index: IO      0
01892 *      0
01893 * @return
01894 *
01895 * @throws arcs::common_interface::AuboException
01896 *
01897 * @par Python
01898 * getToolAnalogOutputRunstate(self: pyaubo_sdk.IoControl, arg0: int) ->
01899 * arcs::common_interface::StandardOutputRunState
01900 *
01901 * @par Lua
01902 * getToolAnalogOutputRunstate(index: number) -> number
01903 *
01904 * @par Lua
01905 * num = getToolAnalogOutputRunstate(0)
01906 *
01907 * @par JSON-RPC
01908 * {"jsonrpc":"2.0","method":"rob1.IoControl.getToolAnalogOutputRunstate","params":[0],"id":1}
01909 *
01910 * @par JSON-RPC
01911 * {"id":1,"jsonrpc":"2.0","result":"None"}
01912 *
01913 * \endchinese
01914 * \english
01915 * Get the runstate for tool analog output.
01916 *
01917 * @param index: Indicates the IO pin, starting from 0.
01918 * For example, 0 means the first pin.
01919 * @return Tool analog output runstate selection
01920 *
01921 * @throws arcs::common_interface::AuboException
01922 *
01923 * @par Python function prototype
01924 * getToolAnalogOutputRunstate(self: pyaubo_sdk.IoControl, arg0: int) ->
01925 * arcs::common_interface::StandardOutputRunState
01926 *
01927 * @par Lua function prototype
01928 * getToolAnalogOutputRunstate(index: number) -> number
01929 *
01930 * @par Lua example
01931 * num = getToolAnalogOutputRunstate(0)
01932 *
01933 * @par JSON-RPC request example
01934 * {"jsonrpc":"2.0","method":"rob1.IoControl.getToolAnalogOutputRunstate","params":[0],"id":1}
01935 *
01936 * @par JSON-RPC response example
01937 * {"id":1,"jsonrpc":"2.0","result":"None"}
01938 *
01939 * \endenglish
01940 */
01941 StandardOutputRunState getToolAnalogOutputRunstate(int index);
01942
01943 /**
01944 * @ingroup IoControl
01945 * \chinese
01946 *
01947 *
01948 * @return 0
01949 * AUBO_BUSY
01950 * AUBO_BAD_STATE
01951 * -AUBO_BAD_STATE
01952 *
01953 * @throws arcs::common_interface::AuboException
01954 *
01955 * @par Python
01956 * setDigitalOutputAfterEStopDefault(self: pyaubo_sdk.IoControl) -> int
01957 *
01958 * @par Lua
01959 * setDigitalOutputAfterEStopDefault() -> nil

```

```

01960 *
01961 * @par Lua
01962 * setDigitalOutputAfterEStopDefault()
01963 *
01964 * @par JSON-RPC
01965 * {"jsonrpc":"2.0","method":"rob1.IoControl.setDigitalOutputAfterEStopDefault","params":[],"id":1}
01966 *
01967 * @par JSON-RPC
01968 * {"id":1,"jsonrpc":"2.0","result":0}
01969 *
01970 * \endchinese
01971 * \english
01972 * Set all digital output states after emergency stop to default(no change)
01973 *
01974 * @return Returns 0 on success; error code on failure.
01975 * AUBO_BUSY
01976 * AUBO_BAD_STATE
01977 * -AUBO_BAD_STATE
01978 *
01979 * @throws arcs::common_interface::AuboException
01980 *
01981 * @par Python function prototype
01982 * setDigitalOutputAfterEStopDefault(self: pyaubo_sdk.IoControl) -> int
01983 *
01984 * @par Lua function prototype
01985 * setDigitalOutputAfterEStopDefault() -> nil
01986 *
01987 * @par Lua example
01988 * setDigitalOutputAfterEStopDefault()
01989 *
01990 * @par JSON-RPC request example
01991 * {"jsonrpc":"2.0","method":"rob1.IoControl.setDigitalOutputAfterEStopDefault","params":[],"id":1}
01992 *
01993 * @par JSON-RPC response example
01994 * {"id":1,"jsonrpc":"2.0","result":0}
01995 *
01996 * \endenglish
01997 */
01998 int setDigitalOutputAfterEStopDefault();
01999
02000 /**
02001 * @ingroup IoControl
02002 * \chinese
02003 *
02004 *
02005 * @return 0
02006 * AUBO_BUSY
02007 * AUBO_BAD_STATE
02008 * -AUBO_BAD_STATE
02009 *
02010 * @throws arcs::common_interface::AuboException
02011 *
02012 * @par Python
02013 * setAnalogOutputAfterEStopDefault(self: pyaubo_sdk.IoControl) -> int
02014 *
02015 * @par Lua
02016 * setAnalogOutputAfterEStopDefault() -> nil
02017 *
02018 * @par Lua
02019 * setAnalogOutputAfterEStopDefault()
02020 *
02021 * @par JSON-RPC
02022 * {"jsonrpc":"2.0","method":"rob1.IoControl.setAnalogOutputAfterEStopDefault","params":[],"id":1}
02023 *
02024 * @par JSON-RPC
02025 * {"id":1,"jsonrpc":"2.0","result":0}
02026 *
02027 * \endchinese
02028 * \english
02029 * Set all analog output states after emergency stop to default(no change)
02030 *
02031 * @return Returns 0 on success; error code on failure.
02032 * AUBO_BUSY
02033 * AUBO_BAD_STATE
02034 * -AUBO_BAD_STATE
02035 *
02036 * @throws arcs::common_interface::AuboException
02037 *
02038 * @par Python function prototype
02039 * setAnalogOutputAfterEStopDefault(self: pyaubo_sdk.IoControl) -> int
02040 *
02041 * @par Lua function prototype
02042 * setAnalogOutputAfterEStopDefault() -> nil
02043 *
02044 * @par Lua example
02045 * setAnalogOutputAfterEStopDefault()
02046 *

```

```

02047 * @par JSON-RPC request example
02048 * {"jsonrpc":"2.0","method":"rob1.IoControl.setAnalogOutputAfterEStopDefault","params":[],"id":1}
02049 *
02050 * @par JSON-RPC response example
02051 * {"id":1,"jsonrpc":"2.0","result":0}
02052 *
02053 * \endenglish
02054 */
02055 int setAnalogOutputAfterEStopDefault();
02056
02057 /**
02058 * @ingroup IoControl
02059 * \chinese
02060 *
02061 *
02062 * @param index: IO
02063 * @param value:
02064 * @return 0
02065 * AUBO_REQUEST_IGNORE
02066 * AUBO_BUSY
02067 * AUBO_BAD_STATE
02068 * -AUBO_INVL_ARGUMENT
02069 * -AUBO_BAD_STATE
02070 *
02071 * @throws arcs::common_interface::AuboException
02072 *
02073 * @par Python
02074 * setStandardDigitalOutputAfterEStop(self: pyaubo_sdk.IoControl, arg0: int,
02075 * arg1: bool) -> int
02076 *
02077 * @par Lua
02078 * setStandardDigitalOutputAfterEStop(index: number, value: boolean) -> nil
02079 *
02080 * @par Lua
02081 * setStandardDigitalOutputAfterEStop(0,true)
02082 *
02083 * @par JSON-RPC
02084 * {"jsonrpc":"2.0","method":"rob1.IoControl.setStandardDigitalOutputAfterEStop","params":[0,true],"id":1}
02085 *
02086 * @par JSON-RPC
02087 * {"id":1,"jsonrpc":"2.0","result":0}
02088 *
02089 * \endchinese
02090 * \english
02091 * Set the value of a standard digital output after emergency stop
02092 *
02093 * @param index: Indicates the IO pin.
02094 * @param value: Output value.
02095 * @return Returns 0 on success; error code on failure.
02096 * AUBO_REQUEST_IGNORE
02097 * AUBO_BUSY
02098 * AUBO_BAD_STATE
02099 * -AUBO_INVL_ARGUMENT
02100 * -AUBO_BAD_STATE
02101 *
02102 * @throws arcs::common_interface::AuboException
02103 *
02104 * @par Python function prototype
02105 * setStandardDigitalOutputAfterEStop(self: pyaubo_sdk.IoControl, arg0: int,
02106 * arg1: bool) -> int
02107 *
02108 * @par Lua function prototype
02109 * setStandardDigitalOutputAfterEStop(index: number, value: boolean) -> nil
02110 *
02111 * @par Lua example
02112 * setStandardDigitalOutputAfterEStop(0,true)
02113 *
02114 * @par JSON-RPC request example
02115 * {"jsonrpc":"2.0","method":"rob1.IoControl.setStandardDigitalOutputAfterEStop","params":[0,true],"id":1}
02116 *
02117 * @par JSON-RPC response example
02118 * {"id":1,"jsonrpc":"2.0","result":0}
02119 *
02120 * \endenglish
02121 */
02122 int setStandardDigitalOutputAfterEStop(int index, bool value);
02123
02124 /**
02125 * @ingroup IoControl
02126 * \chinese
02127 *
02128 *
02129 * @param index: IO
02130 * @param value:
02131 * @return 0
02132 * AUBO_REQUEST_IGNORE
02133 * AUBO_BUSY

```

```

02134 * AUBO_BAD_STATE
02135 * -AUBO_INVL_ARGUMENT
02136 * -AUBO_BAD_STATE
02137 *
02138 * @throws arcs::common_interface::AuboException
02139 *
02140 * @par Python
02141 * setConfigurableDigitalOutputAfterEStop(self: pyaubo_sdk.IoControl, arg0:
02142 * int, arg1: bool) -> int
02143 *
02144 * @par Lua
02145 * setConfigurableDigitalOutputAfterEStop(index: number, value: boolean) ->
02146 * nil
02147 *
02148 * @par Lua
02149 * setConfigurableDigitalOutputAfterEStop(0,true)
02150 *
02151 * @par JSON-RPC
02152 * {"jsonrpc":"2.0","method":"rob1.IoControl.setConfigurableDigitalOutputAfterEStop","params":[0,true],"id":1}
02153 *
02154 * @par JSON-RPC
02155 * {"id":1,"jsonrpc":"2.0","result":0}
02156 *
02157 * \endchinese
02158 * \english
02159 * Set the value of a configurable digital output after emergency stop
02160 *
02161 * @param index: Indicates the IO pin.
02162 * @param value: Output value.
02163 * @return Returns 0 on success; error code on failure.
02164 * AUBO_REQUEST_IGNORE
02165 * AUBO_BUSY
02166 * AUBO_BAD_STATE
02167 * -AUBO_INVL_ARGUMENT
02168 * -AUBO_BAD_STATE
02169 *
02170 * @throws arcs::common_interface::AuboException
02171 *
02172 * @par Python function prototype
02173 * setConfigurableDigitalOutputAfterEStop(self: pyaubo_sdk.IoControl, arg0:
02174 * int, arg1: bool) -> int
02175 *
02176 * @par Lua function prototype
02177 * setConfigurableDigitalOutputAfterEStop(index: number, value: boolean) ->
02178 * nil
02179 *
02180 * @par Lua example
02181 * setConfigurableDigitalOutputAfterEStop(0,true)
02182 *
02183 * @par JSON-RPC request example
02184 * {"jsonrpc":"2.0","method":"rob1.IoControl.setConfigurableDigitalOutputAfterEStop","params":[0,true],"id":1}
02185 *
02186 * @par JSON-RPC response example
02187 * {"id":1,"jsonrpc":"2.0","result":0}
02188 *
02189 * \endenglish
02190 */
02191 int setConfigurableDigitalOutputAfterEStop(int index, bool value);
02192
02193 /**
02194 * @ingroup IoControl
02195 * \chinese
02196 *
02197 *
02198 * @param index: IO 0
02199 * 0
02200 * @param value:
02201 *
02202 * @return 0
02203 * AUBO_REQUEST_IGNORE
02204 * AUBO_BUSY
02205 * AUBO_BAD_STATE
02206 * -AUBO_INVL_ARGUMENT
02207 * -AUBO_BAD_STATE
02208 *
02209 * @throws arcs::common_interface::AuboException
02210 *
02211 * @par Python
02212 * setStandardAnalogOutputAfterEStop(self: pyaubo_sdk.IoControl, arg0: int,
02213 * arg1: float) -> int
02214 *
02215 * @par Lua
02216 * setStandardAnalogOutputAfterEStop(index: number, value: number) -> nil
02217 *
02218 * @par Lua
02219 * setStandardAnalogOutputAfterEStop(0,5.4)
02220 *

```

```

02221 * \endchinese
02222 * \english
02223 * Set the value of standard analog output after emergency stop
02224 *
02225 * @param index: Indicates the IO pin, starting from 0.
02226 * For example, 0 means the first pin.
02227 * @param value: Output value.
02228 *
02229 * @return Returns 0 on success; error code on failure.
02230 * AUBO_REQUEST_IGNORE
02231 * AUBO_BUSY
02232 * AUBO_BAD_STATE
02233 * -AUBO_INVL_ARGUMENT
02234 * -AUBO_BAD_STATE
02235 *
02236 * @throws arcs::common_interface::AuboException
02237 *
02238 * @par Python function prototype
02239 * setStandardAnalogOutputAfterEStop(self: pyaubo_sdk.IoControl, arg0: int,
02240 * arg1: float) -> int
02241 *
02242 * @par Lua function prototype
02243 * setStandardAnalogOutputAfterEStop(index: number, value: number) -> nil
02244 *
02245 * @par Lua example
02246 * setStandardAnalogOutputAfterEStop(0,5.4)
02247 *
02248 * \endenglish
02249 */
02250 int setStandardAnalogOutputAfterEStop(int index, double value);
02251
02252 /**
02253 * @ingroup IoControl
02254 * \chinese
02255 *
02256 *
02257 * @param index: IO      0
02258 * 0
02259 * @param domain:
02260 *
02261 * @return 0
02262 * AUBO_REQUEST_IGNORE
02263 * AUBO_BUSY
02264 * AUBO_BAD_STATE
02265 * -AUBO_INVL_ARGUMENT
02266 * -AUBO_BAD_STATE
02267 *
02268 * @throws arcs::common_interface::AuboException
02269 *
02270 * @par Python
02271 * setStandardAnalogInputDomain(self: pyaubo_sdk.IoControl, arg0: int, arg1:
02272 * int) -> int
02273 *
02274 * @par Lua
02275 * setStandardAnalogInputDomain(index: number, domain: number) -> nil
02276 *
02277 * @par Lua
02278 * setStandardAnalogInputDomain(0,1)
02279 *
02280 * @par JSON-RPC
02281 * {"jsonrpc":"2.0","method":"rob1.IoControl.setStandardAnalogInputDomain","params":[0,8],"id":1}
02282 *
02283 * @par JSON-RPC
02284 * {"id":1,"jsonrpc":"2.0","result":0}
02285 *
02286 * \endchinese
02287 * \english
02288 * Set the range of standard analog input.
02289 *
02290 * @param index: Indicates the IO pin, starting from 0.
02291 * For example, 0 means the first pin.
02292 * @param domain: Input range
02293 *
02294 * @return Returns 0 on success; error code on failure.
02295 * AUBO_REQUEST_IGNORE
02296 * AUBO_BUSY
02297 * AUBO_BAD_STATE
02298 * -AUBO_INVL_ARGUMENT
02299 * -AUBO_BAD_STATE
02300 *
02301 * @throws arcs::common_interface::AuboException
02302 *
02303 * @par Python function prototype
02304 * setStandardAnalogInputDomain(self: pyaubo_sdk.IoControl, arg0: int, arg1:
02305 * int) -> int
02306 *
02307 * @par Lua function prototype

```

```

02308 * setStandardAnalogInputDomain(index: number, domain: number) -> nil
02309 *
02310 * @par Lua example
02311 * setStandardAnalogInputDomain(0,1)
02312 *
02313 * @par JSON-RPC request example
02314 * {"jsonrpc":"2.0","method":"rob1.IoControl.setStandardAnalogInputDomain","params":[0,8],"id":1}
02315 *
02316 * @par JSON-RPC response example
02317 * {"id":1,"jsonrpc":"2.0","result":0}
02318 *
02319 * \endenglish
02320 */
02321 int setStandardAnalogInputDomain(int index, int domain);
02322
02323 /**
02324 * @ingroup IoControl
02325 * \chinese
02326 *
02327 *
02328 * @param index: IO      0
02329 * 0
02330 * @param domain:
02331 * @return 0
02332 * AUBO_REQUEST_IGNORE
02333 * AUBO_BUSY
02334 * AUBO_BAD_STATE
02335 * -AUBO_INVL_ARGUMENT
02336 * -AUBO_BAD_STATE
02337 *
02338 * @throws arcs::common_interface::AuboException
02339 *
02340 * @par Python
02341 * setToolAnalogInputDomain(self: pyaubo_sdk.IoControl, arg0: int, arg1:
02342 * int) -> int
02343 *
02344 * @par Lua
02345 * setToolAnalogInputDomain(index: number, domain: number) -> nil
02346 *
02347 * @par Lua
02348 * setToolAnalogInputDomain(0,1)
02349 *
02350 * @par JSON-RPC
02351 * {"jsonrpc":"2.0","method":"rob1.IoControl.setToolAnalogInputDomain","params":[0,8],"id":1}
02352 *
02353 * @par JSON-RPC
02354 * {"id":1,"jsonrpc":"2.0","result":0}
02355 *
02356 * \endchinese
02357 * \english
02358 * Set the range of tool analog input.
02359 *
02360 * @param index: Indicates the IO pin, starting from 0.
02361 * For example, 0 means the first pin.
02362 * @param domain: Input range
02363 * @return Returns 0 on success; error code on failure.
02364 * AUBO_REQUEST_IGNORE
02365 * AUBO_BUSY
02366 * AUBO_BAD_STATE
02367 * -AUBO_INVL_ARGUMENT
02368 * -AUBO_BAD_STATE
02369 *
02370 * @throws arcs::common_interface::AuboException
02371 *
02372 * @par Python function prototype
02373 * setToolAnalogInputDomain(self: pyaubo_sdk.IoControl, arg0: int, arg1:
02374 * int) -> int
02375 *
02376 * @par Lua function prototype
02377 * setToolAnalogInputDomain(index: number, domain: number) -> nil
02378 *
02379 * @par Lua example
02380 * setToolAnalogInputDomain(0,1)
02381 *
02382 * @par JSON-RPC request example
02383 * {"jsonrpc":"2.0","method":"rob1.IoControl.setToolAnalogInputDomain","params":[0,8],"id":1}
02384 *
02385 * @par JSON-RPC response example
02386 * {"id":1,"jsonrpc":"2.0","result":0}
02387 *
02388 * \endenglish
02389 */
02390 int setToolAnalogInputDomain(int index, int domain);
02391
02392 /**
02393 * @ingroup IoControl
02394 * \chinese

```

```

02395 *
02396 *
02397 * @param index: IO      0
02398 * 0
02399 * @return
02400 *
02401 * @throws arcs::common_interface::AuboException
02402 *
02403 * @par Python
02404 * getStandardAnalogInputDomain(self: pyaubo_sdk.IoControl, arg0: int) ->
02405 * int
02406 *
02407 * @par Lua
02408 * getStandardAnalogInputDomain(index: number) -> number
02409 *
02410 * @par Lua
02411 * num = getStandardAnalogInputDomain(0)
02412 *
02413 * @par JSON-RPC
02414 * {"jsonrpc":"2.0","method":"rob1.IoControl.getStandardAnalogInputDomain","params":[0],"id":1}
02415 *
02416 * @par JSON-RPC
02417 * {"id":1,"jsonrpc":"2.0","result":0}
02418 *
02419 * \endchinese
02420 * \english
02421 * Get the domain of standard analog input.
02422 *
02423 * @param index: Indicates the IO pin, starting from 0.
02424 * For example, 0 means the first pin.
02425 * @return Standard analog input domain
02426 *
02427 * @throws arcs::common_interface::AuboException
02428 *
02429 * @par Python function prototype
02430 * getStandardAnalogInputDomain(self: pyaubo_sdk.IoControl, arg0: int) ->
02431 * int
02432 *
02433 * @par Lua function prototype
02434 * getStandardAnalogInputDomain(index: number) -> number
02435 *
02436 * @par Lua example
02437 * num = getStandardAnalogInputDomain(0)
02438 *
02439 * @par JSON-RPC request example
02440 * {"jsonrpc":"2.0","method":"rob1.IoControl.getStandardAnalogInputDomain","params":[0],"id":1}
02441 *
02442 * @par JSON-RPC response example
02443 * {"id":1,"jsonrpc":"2.0","result":0}
02444 *
02445 * \endenglish
02446 */
02447 int getStandardAnalogInputDomain(int index);
02448
02449 /**
02450 * @ingroup IoControl
02451 * \chinese
02452 *
02453 *
02454 * @param index: IO      0
02455 * 0
02456 * @return
02457 *
02458 * @throws arcs::common_interface::AuboException
02459 *
02460 * @par Python
02461 * getToolAnalogInputDomain(self: pyaubo_sdk.IoControl, arg0: int) -> int
02462 *
02463 * @par Lua
02464 * getToolAnalogInputDomain(index: number) -> number
02465 *
02466 * @par Lua
02467 * num = getToolAnalogInputDomain(0)
02468 *
02469 * @par JSON-RPC
02470 * {"jsonrpc":"2.0","method":"rob1.IoControl.getToolAnalogInputDomain","params":[0],"id":1}
02471 *
02472 * @par JSON-RPC
02473 * {"id":1,"jsonrpc":"2.0","result":10}
02474 *
02475 * \endchinese
02476 * \english
02477 * Get the domain of tool analog input.
02478 *
02479 * @param index: Indicates the IO pin, starting from 0.
02480 * For example, 0 means the first pin.
02481 * @return Tool analog input domain

```



```

02482 *
02483 * @throws arcs::common_interface::AuboException
02484 *
02485 * @par Python function prototype
02486 * getToolAnalogInputDomain(self: pyaubo_sdk.IoControl, arg0: int) -> int
02487 *
02488 * @par Lua function prototype
02489 * getToolAnalogInputDomain(index: number) -> number
02490 *
02491 * @par Lua example
02492 * num = getToolAnalogInputDomain(0)
02493 *
02494 * @par JSON-RPC request example
02495 * {"jsonrpc":"2.0","method":"rob1.IoControl.getToolAnalogInputDomain","params":[0],"id":1}
02496 *
02497 * @par JSON-RPC response example
02498 * {"id":1,"jsonrpc":"2.0","result":10}
02499 *
02500 * \endenglish
02501 */
02502 int getToolAnalogInputDomain(int index);
02503
02504 /**
02505 * @ingroup IoControl
02506 * \chinese
02507 *
02508 *
02509 * @param index: IO      0
02510 *      0
02511 * @param domain:
02512 *
02513 * @return 0
02514 * AUBO_REQUEST_IGNORE
02515 * AUBO_BUSY
02516 * AUBO_BAD_STATE
02517 * -AUBO_INVL_ARGUMENT
02518 * -AUBO_BAD_STATE
02519 *
02520 * @throws arcs::common_interface::AuboException
02521 *
02522 * @par Python
02523 * setStandardAnalogOutputDomain(self: pyaubo_sdk.IoControl, arg0: int,
02524 * arg1: int) -> int
02525 *
02526 * @par Lua
02527 * setStandardAnalogOutputDomain(index: number, domain: number) -> nil
02528 *
02529 * @par Lua
02530 * setStandardAnalogOutputDomain(0,1)
02531 *
02532 * @par JSON-RPC
02533 * {"jsonrpc":"2.0","method":"rob1.IoControl.setStandardAnalogOutputDomain","params":[0,8],"id":1}
02534 *
02535 * @par JSON-RPC
02536 * {"id":1,"jsonrpc":"2.0","result":0}
02537 *
02538 * \endchinese
02539 * \english
02540 * Set the range of standard analog output.
02541 *
02542 * @param index: Indicates the IO pin, starting from 0.
02543 * For example, 0 means the first pin.
02544 * @param domain: Output range
02545 *
02546 * @return Returns 0 on success; error code on failure.
02547 * AUBO_REQUEST_IGNORE
02548 * AUBO_BUSY
02549 * AUBO_BAD_STATE
02550 * -AUBO_INVL_ARGUMENT
02551 * -AUBO_BAD_STATE
02552 *
02553 * @throws arcs::common_interface::AuboException
02554 *
02555 * @par Python function prototype
02556 * setStandardAnalogOutputDomain(self: pyaubo_sdk.IoControl, arg0: int,
02557 * arg1: int) -> int
02558 *
02559 * @par Lua function prototype
02560 * setStandardAnalogOutputDomain(index: number, domain: number) -> nil
02561 *
02562 * @par Lua example
02563 * setStandardAnalogOutputDomain(0,1)
02564 *
02565 * @par JSON-RPC request example
02566 * {"jsonrpc":"2.0","method":"rob1.IoControl.setStandardAnalogOutputDomain","params":[0,8],"id":1}
02567 *
02568 * @par JSON-RPC response example

```

```

02569     * {"id":1,"jsonrpc":"2.0","result":0}
02570     *
02571     * \endenglish
02572     */
02573     int setStandardAnalogOutputDomain(int index, int domain);
02574
02575     /**
02576     * @ingroup IoControl
02577     * \chinese
02578     *
02579     *
02580     * @param index: IO      0
02581     *      0
02582     * @param domain:
02583     *
02584     * @return 0
02585     * AUBO_REQUEST_IGNORE
02586     * AUBO_BUSY
02587     * AUBO_BAD_STATE
02588     * -AUBO_INVL_ARGUMENT
02589     * -AUBO_BAD_STATE
02590     *
02591     * @throws arcs::common_interface::AuboException
02592     *
02593     * @par Python
02594     * setToolAnalogOutputDomain(self: pyaubo_sdk.IoControl, arg0: int, arg1:
02595     * int) -> int
02596     *
02597     * @par Lua
02598     * setToolAnalogOutputDomain(index: number, domain: number) -> nil
02599     *
02600     * @par Lua
02601     * setToolAnalogOutputDomain(0,1)
02602     *
02603     * \endchinese
02604     * \english
02605     * Set the range of tool analog output.
02606     *
02607     * @param index: Indicates the IO pin, starting from 0.
02608     * For example, 0 means the first pin.
02609     * @param domain: Output range
02610     *
02611     * @return Returns 0 on success; error code on failure.
02612     * AUBO_REQUEST_IGNORE
02613     * AUBO_BUSY
02614     * AUBO_BAD_STATE
02615     * -AUBO_INVL_ARGUMENT
02616     * -AUBO_BAD_STATE
02617     *
02618     * @throws arcs::common_interface::AuboException
02619     *
02620     * @par Python function prototype
02621     * setToolAnalogOutputDomain(self: pyaubo_sdk.IoControl, arg0: int, arg1:
02622     * int) -> int
02623     *
02624     * @par Lua function prototype
02625     * setToolAnalogOutputDomain(index: number, domain: number) -> nil
02626     *
02627     * @par Lua example
02628     * setToolAnalogOutputDomain(0,1)
02629     *
02630     * \endenglish
02631     */
02632     int setToolAnalogOutputDomain(int index, int domain);
02633
02634     /**
02635     * @ingroup IoControl
02636     * \chinese
02637     *
02638     *
02639     * @param index: IO      0
02640     *      0
02641     * @return
02642     *
02643     * @throws arcs::common_interface::AuboException
02644     *
02645     * @par Python
02646     * getStandardAnalogOutputDomain(self: pyaubo_sdk.IoControl, arg0: int) ->
02647     * int
02648     *
02649     * @par Lua
02650     * getStandardAnalogOutputDomain(index: number) -> number
02651     *
02652     * @par Lua
02653     * num = getStandardAnalogOutputDomain(0)
02654     *
02655     * @par JSON-RPC

```

```

02656 * {"jsonrpc":"2.0","method":"rob1.IoControl.getStandardAnalogOutputDomain","params":[0],"id":1}
02657 *
02658 * @par JSON-RPC
02659 * {"id":1,"jsonrpc":"2.0","result":0}
02660 *
02661 * \endchinese
02662 * \english
02663 * Get the domain of standard analog output.
02664 *
02665 * @param index: Indicates the IO pin, starting from 0.
02666 * For example, 0 means the first pin.
02667 * @return Standard analog output domain
02668 *
02669 * @throws arcs::common_interface::AuboException
02670 *
02671 * @par Python function prototype
02672 * getStandardAnalogOutputDomain(self: pyaubo_sdk.IoControl, arg0: int) ->
02673 * int
02674 *
02675 * @par Lua function prototype
02676 * getStandardAnalogOutputDomain(index: number) -> number
02677 *
02678 * @par Lua example
02679 * num = getStandardAnalogOutputDomain(0)
02680 *
02681 * @par JSON-RPC request example
02682 * {"jsonrpc":"2.0","method":"rob1.IoControl.getStandardAnalogOutputDomain","params":[0],"id":1}
02683 *
02684 * @par JSON-RPC response example
02685 * {"id":1,"jsonrpc":"2.0","result":0}
02686 *
02687 * \endenglish
02688 */
02689 int getStandardAnalogOutputDomain(int index);
02690
02691 /**
02692 * @ingroup IoControl
02693 * \chinese
02694 *
02695 *
02696 * @param index: IO      0
02697 * 0
02698 * @return
02699 *
02700 * @throws arcs::common_interface::AuboException
02701 *
02702 * @par Python
02703 * getToolAnalogOutputDomain(self: pyaubo_sdk.IoControl, arg0: int) -> int
02704 *
02705 * @par Lua
02706 * getToolAnalogOutputDomain(index: number) -> number
02707 *
02708 * @par Lua
02709 * num = getToolAnalogOutputDomain(0)
02710 *
02711 * @par JSON-RPC
02712 * {"jsonrpc":"2.0","method":"rob1.IoControl.getToolAnalogOutputDomain","params":[0],"id":1}
02713 *
02714 * @par JSON-RPC
02715 * {"id":1,"jsonrpc":"2.0","result":0}
02716 *
02717 * \endchinese
02718 * \english
02719 * Get the domain of tool analog output.
02720 *
02721 * @param index: Indicates the IO pin, starting from 0.
02722 * For example, 0 means the first pin.
02723 * @return Tool analog output domain
02724 *
02725 * @throws arcs::common_interface::AuboException
02726 *
02727 * @par Python function prototype
02728 * getToolAnalogOutputDomain(self: pyaubo_sdk.IoControl, arg0: int) -> int
02729 *
02730 * @par Lua function prototype
02731 * getToolAnalogOutputDomain(index: number) -> number
02732 *
02733 * @par Lua example
02734 * num = getToolAnalogOutputDomain(0)
02735 *
02736 * @par JSON-RPC request example
02737 * {"jsonrpc":"2.0","method":"rob1.IoControl.getToolAnalogOutputDomain","params":[0],"id":1}
02738 *
02739 * @par JSON-RPC response example
02740 * {"id":1,"jsonrpc":"2.0","result":0}
02741 *
02742 * \endenglish

```

```

02743     */
02744     int getToolAnalogOutputDomain(int index);
02745
02746     /**
02747     * @ingroup IoControl
02748     * \chinese
02749     *     ( V)
02750     *
02751     * @param domain:          0 12 24 \n
02752     *     0 0V, 12 12V, 24 24V
02753     *
02754     * @return 0;
02755     * AUBO_REQUEST_IGNORE
02756     * AUBO_BUSY
02757     * AUBO_BAD_STATE
02758     * -AUBO_INVL_ARGUMENT
02759     * -AUBO_BAD_STATE
02760     *
02761     * @throws arcs::common_interface::AuboException
02762     *
02763     * @par Python
02764     * setToolVoltageOutputDomain(self: pyaubo_sdk.IoControl, arg0: int) -> int
02765     *
02766     * @par Lua
02767     * setToolVoltageOutputDomain(domain: number) -> nil
02768     *
02769     * @par Lua
02770     * setToolVoltageOutputDomain(24)
02771     *
02772     * @par JSON-RPC
02773     * {"jsonrpc":"2.0","method":"rob1.IoControl.setToolVoltageOutputDomain","params":[24],"id":1}
02774     *
02775     * @par JSON-RPC
02776     * {"id":1,"jsonrpc":"2.0","result":0}
02777     *
02778     * \endchinese
02779     * \english
02780     * Set the tool voltage output value (unit: V)
02781     *
02782     * @param domain: Tool voltage output value, can be 0, 12, or 24.\n
02783     *     0 means 0V, 12 means 12V, 24 means 24V.
02784     *
02785     * @return Returns 0 on success; error code on failure.
02786     * AUBO_REQUEST_IGNORE
02787     * AUBO_BUSY
02788     * AUBO_BAD_STATE
02789     * -AUBO_INVL_ARGUMENT
02790     * -AUBO_BAD_STATE
02791     *
02792     * @throws arcs::common_interface::AuboException
02793     *
02794     * @par Python function prototype
02795     * setToolVoltageOutputDomain(self: pyaubo_sdk.IoControl, arg0: int) -> int
02796     *
02797     * @par Lua function prototype
02798     * setToolVoltageOutputDomain(domain: number) -> nil
02799     *
02800     * @par Lua example
02801     * setToolVoltageOutputDomain(24)
02802     *
02803     * @par JSON-RPC request example
02804     * {"jsonrpc":"2.0","method":"rob1.IoControl.setToolVoltageOutputDomain","params":[24],"id":1}
02805     *
02806     * @par JSON-RPC response example
02807     * {"id":1,"jsonrpc":"2.0","result":0}
02808     *
02809     * \endenglish
02810     */
02811     int setToolVoltageOutputDomain(int domain);
02812
02813     /**
02814     * @ingroup IoControl
02815     * \chinese
02816     *     ( V)
02817     *
02818     * @return ( V)
02819     *
02820     * @throws arcs::common_interface::AuboException
02821     *
02822     * @par Python
02823     * getToolVoltageOutputDomain(self: pyaubo_sdk.IoControl) -> int
02824     *
02825     * @par Lua
02826     * getToolVoltageOutputDomain() -> number
02827     *
02828     * @par Lua
02829     * num = getToolVoltageOutputDomain()

```

```

02830 *
02831 * @par JSON-RPC
02832 * {"jsonrpc":"2.0","method":"rob1.IoControl.getToolVoltageOutputDomain","params":[],"id":1}
02833 *
02834 * @par JSON-RPC
02835 * {"id":1,"jsonrpc":"2.0","result":0}
02836 *
02837 * \endchinese
02838 * \english
02839 * Get the tool voltage output value (unit: V)
02840 *
02841 * @return Tool voltage output value (unit: V)
02842 *
02843 * @throws arcs::common_interface::AuboException
02844 *
02845 * @par Python function prototype
02846 * getToolVoltageOutputDomain(self: pyaubo_sdk.IoControl) -> int
02847 *
02848 * @par Lua function prototype
02849 * getToolVoltageOutputDomain() -> number
02850 *
02851 * @par Lua example
02852 * num = getToolVoltageOutputDomain()
02853 *
02854 * @par JSON-RPC request example
02855 * {"jsonrpc":"2.0","method":"rob1.IoControl.getToolVoltageOutputDomain","params":[],"id":1}
02856 *
02857 * @par JSON-RPC response example
02858 * {"id":1,"jsonrpc":"2.0","result":0}
02859 *
02860 * \endenglish
02861 */
02862 int getToolVoltageOutputDomain();
02863
02864 /**
02865 * @ingroup IoControl
02866 * \chinese
02867 *
02868 *
02869 * @param index: IO
02870 * @param value:
02871 * @return 0
02872 * AUBO_REQUEST_IGNORE
02873 * AUBO_BUSY
02874 * AUBO_BAD_STATE
02875 * -AUBO_INVL_ARGUMENT
02876 * -AUBO_BAD_STATE
02877 *
02878 * @throws arcs::common_interface::AuboException
02879 *
02880 * @par Python
02881 * setStandardDigitalOutput(self: pyaubo_sdk.IoControl, arg0: int, arg1:
02882 * bool) -> int
02883 *
02884 * @par Lua
02885 * setStandardDigitalOutput(index: number, value: boolean) -> nil
02886 *
02887 * @par Lua
02888 * setStandardDigitalOutput(0,true)
02889 *
02890 * @par JSON-RPC
02891 * {"jsonrpc":"2.0","method":"rob1.IoControl.setStandardDigitalOutput","params":[0,true],"id":1}
02892 *
02893 * @par JSON-RPC
02894 * {"id":1,"jsonrpc":"2.0","result":0}
02895 *
02896 * \endchinese
02897 * \english
02898 * Set the value of a standard digital output.
02899 *
02900 * @param index: Indicates the IO pin.
02901 * @param value: Output value.
02902 * @return Returns 0 on success; error code on failure.
02903 * AUBO_REQUEST_IGNORE
02904 * AUBO_BUSY
02905 * AUBO_BAD_STATE
02906 * -AUBO_INVL_ARGUMENT
02907 * -AUBO_BAD_STATE
02908 *
02909 * @throws arcs::common_interface::AuboException
02910 *
02911 * @par Python function prototype
02912 * setStandardDigitalOutput(self: pyaubo_sdk.IoControl, arg0: int, arg1:
02913 * bool) -> int
02914 *
02915 * @par Lua function prototype
02916 * setStandardDigitalOutput(index: number, value: boolean) -> nil

```

```

02917 *
02918 * @par Lua example
02919 * setStandardDigitalOutput(0,true)
02920 *
02921 * @par JSON-RPC request example
02922 * {"jsonrpc":"2.0","method":"rob1.IoControl.setStandardDigitalOutput","params":[0,true],"id":1}
02923 *
02924 * @par JSON-RPC response example
02925 * {"id":1,"jsonrpc":"2.0","result":0}
02926 *
02927 * \endenglish
02928 */
02929 int setStandardDigitalOutput(int index, bool value);
02930
02931 /**
02932 * @ingroup IoControl
02933 * \chinese
02934 *
02935 *
02936 * @param index
02937 * @param value
02938 * @param duration
02939 *
02940 * @return 0
02941 * AUBO_REQUEST_IGNORE
02942 * AUBO_BUSY
02943 * AUBO_BAD_STATE
02944 * -AUBO_INVL_ARGUMENT
02945 * -AUBO_BAD_STATE
02946 *
02947 * @throws arcs::common_interface::AuboException
02948 *
02949 * @par Python
02950 * setStandardDigitalOutputPulse(self: pyaubo_sdk.IoControl, arg0: int,
02951 * arg1: bool, arg2: float) -> int
02952 *
02953 * @par Lua
02954 * setStandardDigitalOutputPulse(index: number, value: boolean, duration:
02955 * number) -> nil
02956 *
02957 * @par Lua
02958 * setStandardDigitalOutputPulse(0,true,2.5)
02959 *
02960 * @par JSON-RPC
02961 * {"jsonrpc":"2.0","method":"rob1.IoControl.setStandardDigitalOutputPulse","params":[0,true,0.5],"id":1}
02962 *
02963 * @par JSON-RPC
02964 * {"id":1,"jsonrpc":"2.0","result":0}
02965 *
02966 * \endchinese
02967 * \english
02968 * Set digital output pulse.
02969 *
02970 * @param index
02971 * @param value
02972 * @param duration
02973 *
02974 * @return Returns 0 on success; error code on failure.
02975 * AUBO_REQUEST_IGNORE
02976 * AUBO_BUSY
02977 * AUBO_BAD_STATE
02978 * -AUBO_INVL_ARGUMENT
02979 * -AUBO_BAD_STATE
02980 *
02981 * @throws arcs::common_interface::AuboException
02982 *
02983 * @par Python function prototype
02984 * setStandardDigitalOutputPulse(self: pyaubo_sdk.IoControl, arg0: int,
02985 * arg1: bool, arg2: float) -> int
02986 *
02987 * @par Lua function prototype
02988 * setStandardDigitalOutputPulse(index: number, value: boolean, duration:
02989 * number) -> nil
02990 *
02991 * @par Lua example
02992 * setStandardDigitalOutputPulse(0,true,2.5)
02993 *
02994 * @par JSON-RPC request example
02995 * {"jsonrpc":"2.0","method":"rob1.IoControl.setStandardDigitalOutputPulse","params":[0,true,0.5],"id":1}
02996 *
02997 * @par JSON-RPC response example
02998 * {"id":1,"jsonrpc":"2.0","result":0}
02999 *
03000 * \endenglish
03001 */
03002 int setStandardDigitalOutputPulse(int index, bool value, double duration);
03003

```

```

03004  /**
03005  * @ingroup IoControl
03006  * \chinese
03007  *
03008  *
03009  * @param index: IO      0
03010  * 0
03011  * @param value:
03012  *
03013  * @return 0
03014  * AUBO_REQUEST_IGNORE
03015  * AUBO_BUSY
03016  * AUBO_BAD_STATE
03017  * -AUBO_INVL_ARGUMENT
03018  * -AUBO_BAD_STATE
03019  *
03020  * @throws arcs::common_interface::AuboException
03021  *
03022  * @par Python
03023  * setToolDigitalOutput(self: pyaubo_sdk.IoControl, arg0: int, arg1: bool)
03024  * -> int
03025  *
03026  * @par Lua
03027  * setToolDigitalOutput(index: number, value: boolean) -> nil
03028  *
03029  * @par Lua
03030  * setToolDigitalOutput(0,true)
03031  *
03032  * @par JSON-RPC
03033  * {"jsonrpc":"2.0","method":"rob1.IoControl.setToolDigitalOutput","params":[0,true],"id":1}
03034  *
03035  * @par JSON-RPC
03036  * {"id":1,"jsonrpc":"2.0","result":0}
03037  *
03038  * \endchinese
03039  * \english
03040  * Set the value of tool digital output.
03041  *
03042  * @param index: Indicates the IO pin, starting from 0.
03043  * For example, 0 means the first pin.
03044  * @param value: Output value.
03045  *
03046  * @return Returns 0 on success; error code on failure.
03047  * AUBO_REQUEST_IGNORE
03048  * AUBO_BUSY
03049  * AUBO_BAD_STATE
03050  * -AUBO_INVL_ARGUMENT
03051  * -AUBO_BAD_STATE
03052  *
03053  * @throws arcs::common_interface::AuboException
03054  *
03055  * @par Python function prototype
03056  * setToolDigitalOutput(self: pyaubo_sdk.IoControl, arg0: int, arg1: bool)
03057  * -> int
03058  *
03059  * @par Lua function prototype
03060  * setToolDigitalOutput(index: number, value: boolean) -> nil
03061  *
03062  * @par Lua example
03063  * setToolDigitalOutput(0,true)
03064  *
03065  * @par JSON-RPC request example
03066  * {"jsonrpc":"2.0","method":"rob1.IoControl.setToolDigitalOutput","params":[0,true],"id":1}
03067  *
03068  * @par JSON-RPC response example
03069  * {"id":1,"jsonrpc":"2.0","result":0}
03070  *
03071  * \endenglish
03072  */
03073  int setToolDigitalOutput(int index, bool value);
03074
03075  /**
03076  * @ingroup IoControl
03077  * \chinese
03078  *
03079  *
03080  * @param index
03081  * @param value
03082  * @param duration
03083  *
03084  * @return 0
03085  * AUBO_REQUEST_IGNORE
03086  * AUBO_BUSY
03087  * AUBO_BAD_STATE
03088  * -AUBO_INVL_ARGUMENT
03089  * -AUBO_BAD_STATE
03090  *

```

```

03091 * @throws arcs::common_interface::AuboException
03092 *
03093 * @par Python
03094 * setToolDigitalOutputPulse(self: pyaubo_sdk.IoControl, arg0: int, arg1:
03095 * bool, arg2: float) -> int
03096 *
03097 * @par Lua
03098 * setToolDigitalOutputPulse(index: number, value: boolean, duration:
03099 * number) -> nil
03100 *
03101 * @par Lua
03102 * setToolDigitalOutputPulse(0,true,2)
03103 *
03104 * @par JSON-RPC
03105 * {"jsonrpc":"2.0","method":"rob1.IoControl.setToolDigitalOutputPulse","params":[0,true,0.5],"id":1}
03106 *
03107 * @par JSON-RPC
03108 * {"id":1,"jsonrpc":"2.0","result":0}
03109 *
03110 * \endchinese
03111 * \english
03112 * Set tool digital output pulse.
03113 *
03114 * @param index
03115 * @param value
03116 * @param duration
03117 *
03118 * @return Returns 0 on success; error code on failure.
03119 * AUBO_REQUEST_IGNORE
03120 * AUBO_BUSY
03121 * AUBO_BAD_STATE
03122 * -AUBO_INVL_ARGUMENT
03123 * -AUBO_BAD_STATE
03124 *
03125 * @throws arcs::common_interface::AuboException
03126 *
03127 * @par Python function prototype
03128 * setToolDigitalOutputPulse(self: pyaubo_sdk.IoControl, arg0: int, arg1:
03129 * bool, arg2: float) -> int
03130 *
03131 * @par Lua function prototype
03132 * setToolDigitalOutputPulse(index: number, value: boolean, duration:
03133 * number) -> nil
03134 *
03135 * @par Lua example
03136 * setToolDigitalOutputPulse(0,true,2)
03137 *
03138 * @par JSON-RPC request example
03139 * {"jsonrpc":"2.0","method":"rob1.IoControl.setToolDigitalOutputPulse","params":[0,true,0.5],"id":1}
03140 *
03141 * @par JSON-RPC response example
03142 * {"id":1,"jsonrpc":"2.0","result":0}
03143 *
03144 * \endenglish
03145 */
03146 int setToolDigitalOutputPulse(int index, bool value, double duration);
03147
03148 /**
03149 * @ingroup IoControl
03150 * \chinese
03151 *
03152 *
03153 * @param index: IO      0
03154 * 0
03155 * @param value:
03156 *
03157 * @return 0
03158 * AUBO_REQUEST_IGNORE
03159 * AUBO_BUSY
03160 * AUBO_BAD_STATE
03161 * -AUBO_INVL_ARGUMENT
03162 * -AUBO_BAD_STATE
03163 *
03164 * @throws arcs::common_interface::AuboException
03165 *
03166 * @par Python
03167 * setConfigurableDigitalOutput(self: pyaubo_sdk.IoControl, arg0: int, arg1:
03168 * bool) -> int
03169 *
03170 * @par Lua
03171 * setConfigurableDigitalOutput(index: number, value: boolean) -> nil
03172 *
03173 * @par Lua
03174 * setConfigurableDigitalOutput(0,true)
03175 *
03176 * @par JSON-RPC
03177 * {"jsonrpc":"2.0","method":"rob1.IoControl.setConfigurableDigitalOutput","params":[0,true],"id":1}

```



```

03178 *
03179 * @par JSON-RPC
03180 * {"id":1,"jsonrpc":"2.0","result":0}
03181 *
03182 * \endchinese
03183 * \english
03184 * Set the value of configurable digital output.
03185 *
03186 * @param index: Indicates the IO pin, starting from 0.
03187 * For example, 0 means the first pin.
03188 * @param value: Output value.
03189 *
03190 * @return Returns 0 on success; error code on failure.
03191 * AUBO_REQUEST_IGNORE
03192 * AUBO_BUSY
03193 * AUBO_BAD_STATE
03194 * -AUBO_INVL_ARGUMENT
03195 * -AUBO_BAD_STATE
03196 *
03197 * @throws arcs::common_interface::AuboException
03198 *
03199 * @par Python function prototype
03200 * setConfigurableDigitalOutput(self: pyaubo_sdk.IoControl, arg0: int, arg1:
03201 * bool) -> int
03202 *
03203 * @par Lua function prototype
03204 * setConfigurableDigitalOutput(index: number, value: boolean) -> nil
03205 *
03206 * @par Lua example
03207 * setConfigurableDigitalOutput(0,true)
03208 *
03209 * @par JSON-RPC request example
03210 * {"jsonrpc":"2.0","method":"rob1.IoControl.setConfigurableDigitalOutput","params":[0,true],"id":1}
03211 *
03212 * @par JSON-RPC response example
03213 * {"id":1,"jsonrpc":"2.0","result":0}
03214 *
03215 * \endenglish
03216 */
03217 int setConfigurableDigitalOutput(int index, bool value);
03218
03219 /**
03220 * @ingroup IoControl
03221 * \chinese
03222 *
03223 *
03224 * @param index
03225 * @param value
03226 * @param duration
03227 *
03228 * @return 0
03229 * AUBO_REQUEST_IGNORE
03230 * AUBO_BUSY
03231 * AUBO_BAD_STATE
03232 * -AUBO_INVL_ARGUMENT
03233 * -AUBO_BAD_STATE
03234 *
03235 * @throws arcs::common_interface::AuboException
03236 *
03237 * @par Python
03238 * setConfigurableDigitalOutputPulse(self: pyaubo_sdk.IoControl, arg0: int,
03239 * arg1: bool, arg2: float) -> int
03240 *
03241 * @par Lua
03242 * setConfigurableDigitalOutputPulse(index: number, value: boolean,
03243 * duration: number) -> nil
03244 *
03245 * @par Lua
03246 * setConfigurableDigitalOutputPulse(0,true,2.3)
03247 *
03248 * @par JSON-RPC
03249 * {"jsonrpc":"2.0","method":"rob1.IoControl.setConfigurableDigitalOutputPulse","params":[0,true,0.5],"id":1}
03250 *
03251 * @par JSON-RPC
03252 * {"id":1,"jsonrpc":"2.0","result":0}
03253 *
03254 * \endchinese
03255 * \english
03256 * Set configurable digital output pulse.
03257 *
03258 * @param index
03259 * @param value
03260 * @param duration
03261 *
03262 * @return Returns 0 on success; error code on failure.
03263 * AUBO_REQUEST_IGNORE
03264 * AUBO_BUSY

```

```

03265 * AUBO_BAD_STATE
03266 * -AUBO_INVL_ARGUMENT
03267 * -AUBO_BAD_STATE
03268 *
03269 * @throws arcs::common_interface::AuboException
03270 *
03271 * @par Python function prototype
03272 * setConfigurableDigitalOutputPulse(self: pyaubo_sdk.IoControl, arg0: int,
03273 * arg1: bool, arg2: float) -> int
03274 *
03275 * @par Lua function prototype
03276 * setConfigurableDigitalOutputPulse(index: number, value: boolean,
03277 * duration: number) -> nil
03278 *
03279 * @par Lua example
03280 * setConfigurableDigitalOutputPulse(0,true,2.3)
03281 *
03282 * @par JSON-RPC request example
03283 * {"jsonrpc":"2.0","method":"rob1.IoControl.setConfigurableDigitalOutputPulse","params":[0,true,0.5],"id":1}
03284 *
03285 * @par JSON-RPC response example
03286 * {"id":1,"jsonrpc":"2.0","result":0}
03287 *
03288 * \endenglish
03289 */
03290 int setConfigurableDigitalOutputPulse(int index, bool value,
03291                                     double duration);
03292
03293 /**
03294 * @ingroup IoControl
03295 * \chinese
03296 *
03297 *
03298 * @param index: IO      0
03299 *              0
03300 * @param value:
03301 *
03302 * @return 0
03303 * AUBO_REQUEST_IGNORE
03304 * AUBO_BUSY
03305 * AUBO_BAD_STATE
03306 * -AUBO_INVL_ARGUMENT
03307 * -AUBO_BAD_STATE
03308 *
03309 * @throws arcs::common_interface::AuboException
03310 *
03311 * @par Python
03312 * setStandardAnalogOutput(self: pyaubo_sdk.IoControl, arg0: int, arg1:
03313 * float) -> int
03314 *
03315 * @par Lua
03316 * setStandardAnalogOutput(index: number, value: number) -> nil
03317 *
03318 * @par Lua
03319 * setStandardAnalogOutput(0,5.4)
03320 *
03321 * \endchinese
03322 * \english
03323 * Set the value of standard analog output.
03324 *
03325 * @param index: Indicates the IO pin, starting from 0.
03326 * For example, 0 means the first pin.
03327 * @param value: Output value.
03328 *
03329 * @return Returns 0 on success; error code on failure.
03330 * AUBO_REQUEST_IGNORE
03331 * AUBO_BUSY
03332 * AUBO_BAD_STATE
03333 * -AUBO_INVL_ARGUMENT
03334 * -AUBO_BAD_STATE
03335 *
03336 * @throws arcs::common_interface::AuboException
03337 *
03338 * @par Python function prototype
03339 * setStandardAnalogOutput(self: pyaubo_sdk.IoControl, arg0: int, arg1:
03340 * float) -> int
03341 *
03342 * @par Lua function prototype
03343 * setStandardAnalogOutput(index: number, value: number) -> nil
03344 *
03345 * @par Lua example
03346 * setStandardAnalogOutput(0,5.4)
03347 *
03348 * \endenglish
03349 */
03350 int setStandardAnalogOutput(int index, double value);
03351

```

```

03352  /**
03353  * @ingroup IoControl
03354  * \chinese
03355  *
03356  *
03357  * @param index: IO      0
03358  * 0
03359  * @param value:
03360  *
03361  * @return 0
03362  * AUBO_REQUEST_IGNORE
03363  * AUBO_BUSY
03364  * AUBO_BAD_STATE
03365  * -AUBO_INVL_ARGUMENT
03366  * -AUBO_BAD_STATE
03367  *
03368  * @throws arcs::common_interface::AuboException
03369  *
03370  * @par Python
03371  * setToolAnalogOutput(self: pyaubo_sdk.IoControl, arg0: int, arg1: float)
03372  * -> int
03373  *
03374  * @par Lua
03375  * setToolAnalogOutput(index: number, value: number) -> nil
03376  *
03377  * @par Lua
03378  * setToolAnalogOutput(0,1.2)
03379  *
03380  * @par JSON-RPC
03381  * {"jsonrpc":"2.0","method":"rob1.IoControl.setToolAnalogOutput","params":[0,0.5],"id":1}
03382  *
03383  * @par JSON-RPC
03384  * {"id":1,"jsonrpc":"2.0","result":13}
03385  *
03386  * \endchinese
03387  * \english
03388  * Set the value of tool analog output.
03389  *
03390  * @param index: Indicates the IO pin, starting from 0.
03391  * For example, 0 means the first pin.
03392  * @param value: Output value.
03393  *
03394  * @return Returns 0 on success; error code on failure.
03395  * AUBO_REQUEST_IGNORE
03396  * AUBO_BUSY
03397  * AUBO_BAD_STATE
03398  * -AUBO_INVL_ARGUMENT
03399  * -AUBO_BAD_STATE
03400  *
03401  * @throws arcs::common_interface::AuboException
03402  *
03403  * @par Python function prototype
03404  * setToolAnalogOutput(self: pyaubo_sdk.IoControl, arg0: int, arg1: float)
03405  * -> int
03406  *
03407  * @par Lua function prototype
03408  * setToolAnalogOutput(index: number, value: number) -> nil
03409  *
03410  * @par Lua example
03411  * setToolAnalogOutput(0,1.2)
03412  *
03413  * @par JSON-RPC request example
03414  * {"jsonrpc":"2.0","method":"rob1.IoControl.setToolAnalogOutput","params":[0,0.5],"id":1}
03415  *
03416  * @par JSON-RPC response example
03417  * {"id":1,"jsonrpc":"2.0","result":13}
03418  *
03419  * \endenglish
03420  */
03421  int setToolAnalogOutput(int index, double value);
03422
03423  /**
03424  * @ingroup IoControl
03425  * \chinese
03426  *
03427  *
03428  * @param index: IO      0
03429  * 0
03430  *
03431  * @return true; false
03432  *
03433  * @throws arcs::common_interface::AuboException
03434  *
03435  * @par Python
03436  * getStandardDigitalInput(self: pyaubo_sdk.IoControl, arg0: int) -> bool
03437  *
03438  * @par Lua

```

```

03439 * getStandardDigitalInput(index: number) -> boolean
03440 *
03441 * @par Lua
03442 * status = getStandardDigitalInput(0)
03443 *
03444 * @par JSON-RPC
03445 * {"jsonrpc":"2.0","method":"rob1.IoControl.getStandardDigitalInput","params":[0],"id":1}
03446 *
03447 * @par JSON-RPC
03448 * {"id":1,"jsonrpc":"2.0","result":false}
03449 *
03450 * \endchinese
03451 * \english
03452 * Get the value of a standard digital input.
03453 *
03454 * @param index: Indicates the IO pin, starting from 0.
03455 * For example, 0 means the first pin.
03456 *
03457 * @return Returns true for high level; false for low level.
03458 *
03459 * @throws arcs::common_interface::AuboException
03460 *
03461 * @par Python function prototype
03462 * getStandardDigitalInput(self: pyaubo_sdk.IoControl, arg0: int) -> bool
03463 *
03464 * @par Lua function prototype
03465 * getStandardDigitalInput(index: number) -> boolean
03466 *
03467 * @par Lua example
03468 * status = getStandardDigitalInput(0)
03469 *
03470 * @par JSON-RPC request example
03471 * {"jsonrpc":"2.0","method":"rob1.IoControl.getStandardDigitalInput","params":[0],"id":1}
03472 *
03473 * @par JSON-RPC response example
03474 * {"id":1,"jsonrpc":"2.0","result":false}
03475 *
03476 * \endenglish
03477 */
03478 bool getStandardDigitalInput(int index);
03479
03480 /**
03481 * @ingroup IoControl
03482 * \chinese
03483 *
03484 *
03485 * @return \n
03486 *      2863267846 , 2  1010101010101000000000000000110
03487 *      16
03488 *      DI00    ,    DI01    \n
03489 *      1      0
03490 *
03491 * @throws arcs::common_interface::AuboException
03492 *
03493 * @par Python
03494 * getStandardDigitalInputs(self: pyaubo_sdk.IoControl) -> int
03495 *
03496 * @par Lua
03497 * getStandardDigitalInputs() -> number
03498 *
03499 * @par Lua
03500 * num = getStandardDigitalInputs()
03501 *
03502 * @par JSON-RPC
03503 * {"jsonrpc":"2.0","method":"rob1.IoControl.getStandardDigitalInputs","params":[],"id":1}
03504 *
03505 * @par JSON-RPC
03506 * {"id":1,"jsonrpc":"2.0","result":0}
03507 *
03508 * \endchinese
03509 * \english
03510 * Get all standard digital input values.
03511 *
03512 * @return All standard digital input values.\n
03513 * For example, if the return value is 2863267846, its binary representation
03514 * is 1010101010101000000000000000110. The lower 16 bits represent the
03515 * status of all standard digital inputs, the least significant bit
03516 * indicates the input status of DI00, the second least significant bit
03517 * indicates DI01, and so on.\n 1 means high level, 0 means low level.
03518 *
03519 * @throws arcs::common_interface::AuboException
03520 *
03521 * @par Python function prototype
03522 * getStandardDigitalInputs(self: pyaubo_sdk.IoControl) -> int
03523 *
03524 * @par Lua function prototype
03525 * getStandardDigitalInputs() -> number

```

```

03526 *
03527 * @par Lua example
03528 * num = getStandardDigitalInputs()
03529 *
03530 * @par JSON-RPC request example
03531 * {"jsonrpc":"2.0","method":"rob1.IoControl.getStandardDigitalInputs","params":[],"id":1}
03532 *
03533 * @par JSON-RPC response example
03534 * {"id":1,"jsonrpc":"2.0","result":0}
03535 *
03536 * \endenglish
03537 */
03538 uint32_t getStandardDigitalInputs();
03539
03540 /**
03541 * @ingroup IoControl
03542 * \chinese
03543 *
03544 *
03545 * @param index: IO      0
03546 * 0
03547 *
03548 * @return true; false
03549 *
03550 * @throws arcs::common_interface::AuboException
03551 *
03552 * @par Python
03553 * getToolDigitalInput(self: pyaubo_sdk.IoControl, arg0: int) -> bool
03554 *
03555 * @par Lua
03556 * getToolDigitalInput(index: number) -> boolean
03557 *
03558 * @par Lua
03559 * status = getToolDigitalInput(0)
03560 *
03561 * @par JSON-RPC
03562 * {"jsonrpc":"2.0","method":"rob1.IoControl.getToolDigitalInput","params":[0],"id":1}
03563 *
03564 * @par JSON-RPC
03565 * {"id":1,"jsonrpc":"2.0","result":false}
03566 *
03567 * \endchinese
03568 * \english
03569 * Get the value of tool digital input.
03570 *
03571 * @param index: Indicates the IO pin, starting from 0.
03572 * For example, 0 means the first pin.
03573 *
03574 * @return Returns true for high level; false for low level.
03575 *
03576 * @throws arcs::common_interface::AuboException
03577 *
03578 * @par Python function prototype
03579 * getToolDigitalInput(self: pyaubo_sdk.IoControl, arg0: int) -> bool
03580 *
03581 * @par Lua function prototype
03582 * getToolDigitalInput(index: number) -> boolean
03583 *
03584 * @par Lua example
03585 * status = getToolDigitalInput(0)
03586 *
03587 * @par JSON-RPC request example
03588 * {"jsonrpc":"2.0","method":"rob1.IoControl.getToolDigitalInput","params":[0],"id":1}
03589 *
03590 * @par JSON-RPC response example
03591 * {"id":1,"jsonrpc":"2.0","result":false}
03592 *
03593 * \endenglish
03594 */
03595 bool getToolDigitalInput(int index);
03596
03597 /**
03598 * @ingroup IoControl
03599 * \chinese
03600 *
03601 *
03602 * @return
03603 *
03604 * @throws arcs::common_interface::AuboException
03605 *
03606 * @par Python
03607 * getToolDigitalInputs(self: pyaubo_sdk.IoControl) -> int
03608 *
03609 * @par Lua
03610 * getToolDigitalInputs() -> number
03611 *
03612 * @par Lua

```

```

03613 * num = getToolDigitalInputs()
03614 *
03615 * @par JSON-RPC
03616 * {"jsonrpc":"2.0","method":"rob1.IoControl.getToolDigitalInputs","params":[],"id":1}
03617 *
03618 * @par JSON-RPC
03619 * {"id":1,"jsonrpc":"2.0","result":0}
03620 *
03621 * \endchinese
03622 * \english
03623 * Get all tool digital input values.
03624 *
03625 * @return Returns all tool digital input values.
03626 *
03627 * @throws arcs::common_interface::AuboException
03628 *
03629 * @par Python function prototype
03630 * getToolDigitalInputs(self: pyaubo_sdk.IoControl) -> int
03631 *
03632 * @par Lua function prototype
03633 * getToolDigitalInputs() -> number
03634 *
03635 * @par Lua example
03636 * num = getToolDigitalInputs()
03637 *
03638 * @par JSON-RPC request example
03639 * {"jsonrpc":"2.0","method":"rob1.IoControl.getToolDigitalInputs","params":[],"id":1}
03640 *
03641 * @par JSON-RPC response example
03642 * {"id":1,"jsonrpc":"2.0","result":0}
03643 *
03644 * \endenglish
03645 */
03646 uint32_t getToolDigitalInputs();
03647
03648 /**
03649 * @ingroup IoControl
03650 * \chinese
03651 *
03652 *
03653 * @note IO
03654 *
03655 * @param index: IO 0
03656 * 0
03657 *
03658 * @return true; false
03659 *
03660 * @throws arcs::common_interface::AuboException
03661 *
03662 * @par Python
03663 * getConfigurableDigitalInput(self: pyaubo_sdk.IoControl, arg0: int) ->
03664 * bool
03665 *
03666 * @par Lua
03667 * getConfigurableDigitalInput(index: number) -> boolean
03668 *
03669 * @par Lua
03670 * status = getConfigurableDigitalInput(0)
03671 *
03672 * @par JSON-RPC
03673 * {"jsonrpc":"2.0","method":"rob1.IoControl.getConfigurableDigitalInput","params":[0],"id":1}
03674 *
03675 * @par JSON-RPC
03676 * {"id":1,"jsonrpc":"2.0","result":false}
03677 *
03678 * \endchinese
03679 * \english
03680 * Get the value of configurable digital input.
03681 *
03682 * @note Can be used to get the value of safety IO.
03683 *
03684 * @param index: Indicates the IO pin, starting from 0.
03685 * For example, 0 means the first pin.
03686 *
03687 * @return Returns true for high level; false for low level.
03688 *
03689 * @throws arcs::common_interface::AuboException
03690 *
03691 * @par Python function prototype
03692 * getConfigurableDigitalInput(self: pyaubo_sdk.IoControl, arg0: int) ->
03693 * bool
03694 *
03695 * @par Lua function prototype
03696 * getConfigurableDigitalInput(index: number) -> boolean
03697 *
03698 * @par Lua example
03699 * status = getConfigurableDigitalInput(0)

```

```

03700 *
03701 * @par JSON-RPC request example
03702 * {"jsonrpc":"2.0","method":"rob1.IoControl.getConfigurableDigitalInput","params":[0],"id":1}
03703 *
03704 * @par JSON-RPC response example
03705 * {"id":1,"jsonrpc":"2.0","result":false}
03706 *
03707 * \endenglish
03708 */
03709 bool getConfigurableDigitalInput(int index);
03710
03711 /**
03712 * @ingroup IoControl
03713 * \chinese
03714 *
03715 *
03716 * @note IO
03717 *
03718 * @return \n
03719 * 2863267846, 2 10101010101010100000000000000110
03720 * 16
03721 * 0 , 1 \n
03722 * 1 0
03723 *
03724 * @throws arcs::common_interface::AuboException
03725 *
03726 * @par Python
03727 * getConfigurableDigitalInputs(self: pyaubo_sdk.IoControl) -> int
03728 *
03729 * @par Lua
03730 * getConfigurableDigitalInputs() -> number
03731 *
03732 * @par Lua
03733 * num = getConfigurableDigitalInputs()
03734 *
03735 * @par JSON-RPC
03736 * {"jsonrpc":"2.0","method":"rob1.IoControl.getConfigurableDigitalInputs","params":[],"id":1}
03737 *
03738 * @par JSON-RPC
03739 * {"id":1,"jsonrpc":"2.0","result":0}
03740 *
03741 * \endchinese
03742 * \english
03743 * Get all configurable digital input values.
03744 *
03745 * @note Can be used to get the value of safety IO.
03746 *
03747 * @return All configurable digital input values.\n
03748 * For example, if the return value is 2863267846, its binary representation
03749 * is 10101010101010100000000000000110. The lower 16 bits represent the
03750 * status of all input pins, the least significant bit indicates the input
03751 * status of pin 0, the second least significant bit indicates pin 1, and so
03752 * on.\n 1 means high level, 0 means low level.
03753 *
03754 * @throws arcs::common_interface::AuboException
03755 *
03756 * @par Python function prototype
03757 * getConfigurableDigitalInputs(self: pyaubo_sdk.IoControl) -> int
03758 *
03759 * @par Lua function prototype
03760 * getConfigurableDigitalInputs() -> number
03761 *
03762 * @par Lua example
03763 * num = getConfigurableDigitalInputs()
03764 *
03765 * @par JSON-RPC request example
03766 * {"jsonrpc":"2.0","method":"rob1.IoControl.getConfigurableDigitalInputs","params":[],"id":1}
03767 *
03768 * @par JSON-RPC response example
03769 * {"id":1,"jsonrpc":"2.0","result":0}
03770 *
03771 * \endenglish
03772 */
03773 uint32_t getConfigurableDigitalInputs();
03774
03775 /**
03776 * @ingroup IoControl
03777 * \chinese
03778 *
03779 *
03780 * @param index: IO 0
03781 * 0
03782 *
03783 * @return true; false
03784 *
03785 * @throws arcs::common_interface::AuboException
03786 *

```

```

03787 * @par Python
03788 * getStandardDigitalOutput(self: pyaubo_sdk.IoControl, arg0: int) -> bool
03789 *
03790 * @par Lua
03791 * getStandardDigitalOutput(index: number) -> boolean
03792 *
03793 * @par Lua
03794 * status = getStandardDigitalOutput(0)
03795 *
03796 * @par JSON-RPC
03797 * {"jsonrpc": "2.0", "method": "rob1.IoControl.getStandardDigitalOutput", "params": [0], "id": 1}
03798 *
03799 * @par JSON-RPC
03800 * {"id": 1, "jsonrpc": "2.0", "result": true}
03801 *
03802 * \endchinese
03803 * \english
03804 * Get the value of a standard digital output.
03805 *
03806 * @param index: Indicates the IO pin, starting from 0.
03807 * For example, 0 means the first pin.
03808 *
03809 * @return Returns true for high level; false for low level.
03810 *
03811 * @throws arcs::common_interface::AuboException
03812 *
03813 * @par Python function prototype
03814 * getStandardDigitalOutput(self: pyaubo_sdk.IoControl, arg0: int) -> bool
03815 *
03816 * @par Lua function prototype
03817 * getStandardDigitalOutput(index: number) -> boolean
03818 *
03819 * @par Lua example
03820 * status = getStandardDigitalOutput(0)
03821 *
03822 * @par JSON-RPC request example
03823 * {"jsonrpc": "2.0", "method": "rob1.IoControl.getStandardDigitalOutput", "params": [0], "id": 1}
03824 *
03825 * @par JSON-RPC response example
03826 * {"id": 1, "jsonrpc": "2.0", "result": true}
03827 *
03828 * \endenglish
03829 */
03830 bool getStandardDigitalOutput(int index);
03831
03832 /**
03833 * @ingroup IoControl
03834 * \chinese
03835 *
03836 *
03837 * @return \n
03838 *      2863267846 , 2  101010101010101000000000000000110
03839 *      16
03840 *      DI00   ,   DI01   \n
03841 *      1      0      .
03842 *
03843 * @throws arcs::common_interface::AuboException
03844 *
03845 * @par Python
03846 * getStandardDigitalOutputs(self: pyaubo_sdk.IoControl) -> int
03847 *
03848 * @par Lua
03849 * getStandardDigitalOutputs() -> number
03850 *
03851 * @par Lua
03852 * num = getStandardDigitalOutputs()
03853 *
03854 * @par JSON-RPC
03855 * {"jsonrpc": "2.0", "method": "rob1.IoControl.getStandardDigitalOutputs", "params": [], "id": 1}
03856 *
03857 * @par JSON-RPC
03858 * {"id": 1, "jsonrpc": "2.0", "result": 69}
03859 *
03860 * \endchinese
03861 * \english
03862 * Get all standard digital output values.
03863 *
03864 * @return All standard digital output values.\n
03865 * For example, if the return value is 2863267846, its binary representation
03866 * is 1010101010101010100000000000000110. The lower 16 bits represent the
03867 * status of all standard digital outputs, the least significant bit
03868 * indicates the output status of DO00, the second least significant bit
03869 * indicates DO01, and so on.\n 1 means high level, 0 means low level.
03870 *
03871 * @throws arcs::common_interface::AuboException
03872 *
03873 * @par Python function prototype

```



```

03874 * getStandardDigitalOutputs(self: pyaubo_sdk.IoControl) -> int
03875 *
03876 * @par Lua function prototype
03877 * getStandardDigitalOutputs() -> number
03878 *
03879 * @par Lua example
03880 * num = getStandardDigitalOutputs()
03881 *
03882 * @par JSON-RPC request example
03883 * {"jsonrpc":"2.0","method":"rob1.IoControl.getStandardDigitalOutputs","params":[],"id":1}
03884 *
03885 * @par JSON-RPC response example
03886 * {"id":1,"jsonrpc":"2.0","result":69}
03887 *
03888 * \endenglish
03889 */
03890 uint32_t getStandardDigitalOutputs();
03891
03892 /**
03893 * @ingroup IoControl
03894 * \chinese
03895 *
03896 *
03897 * @param index: IO      0
03898 *      0
03899 *
03900 * @return true; false
03901 *
03902 * @throws arcs::common_interface::AuboException
03903 *
03904 * @par Python
03905 * getToolDigitalOutput(self: pyaubo_sdk.IoControl, arg0: int) -> bool
03906 *
03907 * @par Lua
03908 * getToolDigitalOutput(index: number) -> boolean
03909 *
03910 * @par Lua
03911 * status = getToolDigitalOutput(0)
03912 *
03913 * @par JSON-RPC
03914 * {"jsonrpc":"2.0","method":"rob1.IoControl.getToolDigitalOutput","params":[0],"id":1}
03915 *
03916 * @par JSON-RPC
03917 * {"id":1,"jsonrpc":"2.0","result":false}
03918 *
03919 * \endchinese
03920 * \english
03921 * Get the value of tool digital output.
03922 *
03923 * @param index: Indicates the IO pin, starting from 0.
03924 * For example, 0 means the first pin.
03925 *
03926 * @return Returns true for high level; false for low level.
03927 *
03928 * @throws arcs::common_interface::AuboException
03929 *
03930 * @par Python function prototype
03931 * getToolDigitalOutput(self: pyaubo_sdk.IoControl, arg0: int) -> bool
03932 *
03933 * @par Lua function prototype
03934 * getToolDigitalOutput(index: number) -> boolean
03935 *
03936 * @par Lua example
03937 * status = getToolDigitalOutput(0)
03938 *
03939 * @par JSON-RPC request example
03940 * {"jsonrpc":"2.0","method":"rob1.IoControl.getToolDigitalOutput","params":[0],"id":1}
03941 *
03942 * @par JSON-RPC response example
03943 * {"id":1,"jsonrpc":"2.0","result":false}
03944 *
03945 * \endenglish
03946 */
03947 bool getToolDigitalOutput(int index);
03948
03949 /**
03950 * @ingroup IoControl
03951 * \chinese
03952 *
03953 *
03954 * @return
03955 *
03956 * @throws arcs::common_interface::AuboException
03957 *
03958 * @par Python
03959 * getToolDigitalOutputs(self: pyaubo_sdk.IoControl) -> int
03960 *

```

```

03961 * @par Lua
03962 * getToolDigitalOutputs() -> number
03963 *
03964 * @par Lua
03965 * num = getToolDigitalOutputs()
03966 *
03967 * @par JSON-RPC
03968 * {"jsonrpc":"2.0","method":"rob1.IoControl.getToolDigitalOutputs","params":[],"id":1}
03969 *
03970 * @par JSON-RPC
03971 * {"id":1,"jsonrpc":"2.0","result":9}
03972 *
03973 * \endchinese
03974 * \english
03975 * Get all tool digital output values.
03976 *
03977 * @return All tool digital output values.
03978 *
03979 * @throws arcs::common_interface::AuboException
03980 *
03981 * @par Python function prototype
03982 * getToolDigitalOutputs(self: pyaubo_sdk.IoControl) -> int
03983 *
03984 * @par Lua function prototype
03985 * getToolDigitalOutputs() -> number
03986 *
03987 * @par Lua example
03988 * num = getToolDigitalOutputs()
03989 *
03990 * @par JSON-RPC request example
03991 * {"jsonrpc":"2.0","method":"rob1.IoControl.getToolDigitalOutputs","params":[],"id":1}
03992 *
03993 * @par JSON-RPC response example
03994 * {"id":1,"jsonrpc":"2.0","result":9}
03995 *
03996 * \endenglish
03997 */
03998 uint32_t getToolDigitalOutputs();
03999
04000 /**
04001 * @ingroup IoControl
04002 * \chinese
04003 *
04004 *
04005 * @note IO
04006 *
04007 * @param index: IO 0
04008 * 0
04009 *
04010 * @return true; false
04011 *
04012 * @throws arcs::common_interface::AuboException
04013 *
04014 * @par Python
04015 * getConfigurableDigitalOutput(self: pyaubo_sdk.IoControl, arg0: int) ->
04016 * bool
04017 *
04018 * @par Lua
04019 * getConfigurableDigitalOutput(index: number) -> boolean
04020 *
04021 * @par Lua
04022 * status = getConfigurableDigitalOutput(0)
04023 *
04024 * @par JSON-RPC
04025 * {"jsonrpc":"2.0","method":"rob1.IoControl.getConfigurableDigitalOutput","params":[0],"id":1}
04026 *
04027 * @par JSON-RPC
04028 * {"id":1,"jsonrpc":"2.0","result":true}
04029 *
04030 * \endchinese
04031 * \english
04032 * Get the value of configurable digital output.
04033 *
04034 * @note Can be used to get the value of safety IO.
04035 *
04036 * @param index: Indicates the IO pin, starting from 0.
04037 * For example, 0 means the first pin.
04038 *
04039 * @return Returns true for high level; false for low level.
04040 *
04041 * @throws arcs::common_interface::AuboException
04042 *
04043 * @par Python function prototype
04044 * getConfigurableDigitalOutput(self: pyaubo_sdk.IoControl, arg0: int) ->
04045 * bool
04046 *
04047 * @par Lua function prototype

```

```

04048 * getConfigurableDigitalOutput(index: number) -> boolean
04049 *
04050 * @par Lua example
04051 * status = getConfigurableDigitalOutput(0)
04052 *
04053 * @par JSON-RPC request example
04054 * {"jsonrpc":"2.0","method":"rob1.IoControl.getConfigurableDigitalOutput","params":[0],"id":1}
04055 *
04056 * @par JSON-RPC response example
04057 * {"id":1,"jsonrpc":"2.0","result":true}
04058 *
04059 * \endenglish
04060 */
04061 bool getConfigurableDigitalOutput(int index);
04062
04063 /**
04064 * @ingroup IoControl
04065 * \chinese
04066 *
04067 *
04068 * @note IO
04069 *
04070 * @return \n
04071 * 2863267846, 2 101010101010101000000000000000110
04072 * 16
04073 * 0 , 1 \n
04074 * 1 0 .
04075 *
04076 * @throws arcs::common_interface::AuboException
04077 *
04078 * @par Python
04079 * getConfigurableDigitalOutputs(self: pyaubo_sdk.IoControl) -> int
04080 *
04081 * @par Lua
04082 * getConfigurableDigitalOutputs() -> number
04083 *
04084 * @par Lua
04085 * num = getConfigurableDigitalOutputs()
04086 *
04087 * @par JSON-RPC
04088 * {"jsonrpc":"2.0","method":"rob1.IoControl.getConfigurableDigitalOutputs","params":[],"id":1}
04089 *
04090 * @par JSON-RPC
04091 * {"id":1,"jsonrpc":"2.0","result":1}
04092 *
04093 * \endchinese
04094 * \english
04095 * Get all configurable digital output values.
04096 *
04097 * @note Can be used to get the value of safety IO.
04098 *
04099 * @return All configurable digital output values.\n
04100 * For example, if the return value is 2863267846, its binary representation
04101 * is 101010101010101000000000000000110. The lower 16 bits represent the
04102 * status of all output pins, the least significant bit indicates the output
04103 * status of pin 0, the second least significant bit indicates pin 1, and so
04104 * on.\n 1 means high level, 0 means low level.
04105 *
04106 * @throws arcs::common_interface::AuboException
04107 *
04108 * @par Python function prototype
04109 * getConfigurableDigitalOutputs(self: pyaubo_sdk.IoControl) -> int
04110 *
04111 * @par Lua function prototype
04112 * getConfigurableDigitalOutputs() -> number
04113 *
04114 * @par Lua example
04115 * num = getConfigurableDigitalOutputs()
04116 *
04117 * @par JSON-RPC request example
04118 * {"jsonrpc":"2.0","method":"rob1.IoControl.getConfigurableDigitalOutputs","params":[],"id":1}
04119 *
04120 * @par JSON-RPC response example
04121 * {"id":1,"jsonrpc":"2.0","result":1}
04122 *
04123 * \endenglish
04124 */
04125 uint32_t getConfigurableDigitalOutputs();
04126
04127 /**
04128 * @ingroup IoControl
04129 * \chinese
04130 *
04131 *
04132 * @param index: IO 0
04133 * 0
04134 *

```

```

04135 * @return
04136 *
04137 * @throws arcs::common_interface::AuboException
04138 *
04139 * @par Python
04140 * getStandardAnalogInput(self: pyaubo_sdk.IoControl, arg0: int) -> float
04141 *
04142 * @par Lua
04143 * getStandardAnalogInput(index: number) -> number
04144 *
04145 * @par Lua
04146 * getStandardAnalogInput(0)
04147 *
04148 * @par JSON-RPC
04149 * {"jsonrpc":"2.0","method":"rob1.IoControl.getStandardAnalogInput","params":[0],"id":1}
04150 *
04151 * @par JSON-RPC
04152 * {"id":1,"jsonrpc":"2.0","result":0.0}
04153 *
04154 * \endchinese
04155 * \english
04156 * Get the value of standard analog input.
04157 *
04158 * @param index: Indicates the IO pin, starting from 0.
04159 * For example, 0 means the first pin.
04160 *
04161 * @return Standard analog input value.
04162 *
04163 * @throws arcs::common_interface::AuboException
04164 *
04165 * @par Python function prototype
04166 * getStandardAnalogInput(self: pyaubo_sdk.IoControl, arg0: int) -> float
04167 *
04168 * @par Lua function prototype
04169 * getStandardAnalogInput(index: number) -> number
04170 *
04171 * @par Lua example
04172 * getStandardAnalogInput(0)
04173 *
04174 * @par JSON-RPC request example
04175 * {"jsonrpc":"2.0","method":"rob1.IoControl.getStandardAnalogInput","params":[0],"id":1}
04176 *
04177 * @par JSON-RPC response example
04178 * {"id":1,"jsonrpc":"2.0","result":0.0}
04179 *
04180 * \endenglish
04181 */
04182 double getStandardAnalogInput(int index);
04183
04184 /**
04185 * @ingroup IoControl
04186 * \chinese
04187 *
04188 *
04189 * @param index: IO      0
04190 *      0
04191 *
04192 * @return
04193 *
04194 * @throws arcs::common_interface::AuboException
04195 *
04196 * @par Python
04197 * getToolAnalogInput(self: pyaubo_sdk.IoControl, arg0: int) -> float
04198 *
04199 * @par Lua
04200 * getToolAnalogInput(index: number) -> number
04201 *
04202 * @par Lua
04203 * num = getToolAnalogInput(0)
04204 *
04205 * @par JSON-RPC
04206 * {"jsonrpc":"2.0","method":"rob1.IoControl.getToolAnalogInput","params":[0],"id":1}
04207 *
04208 * @par JSON-RPC
04209 * {"id":1,"jsonrpc":"2.0","result":0.0}
04210 *
04211 * \endchinese
04212 * \english
04213 * Get the value of tool analog input.
04214 *
04215 * @param index: Indicates the IO pin, starting from 0.
04216 * For example, 0 means the first pin.
04217 *
04218 * @return Tool analog input value.
04219 *
04220 * @throws arcs::common_interface::AuboException
04221 *

```

```

04222 * @par Python function prototype
04223 * getToolAnalogInput(self: pyaubo_sdk.IoControl, arg0: int) -> float
04224 *
04225 * @par Lua function prototype
04226 * getToolAnalogInput(index: number) -> number
04227 *
04228 * @par Lua example
04229 * num = getToolAnalogInput(0)
04230 *
04231 * @par JSON-RPC request example
04232 * {"jsonrpc":"2.0","method":"rob1.IoControl.getToolAnalogInput","params":[0],"id":1}
04233 *
04234 * @par JSON-RPC response example
04235 * {"id":1,"jsonrpc":"2.0","result":0.0}
04236 *
04237 * \endenglish
04238 */
04239 double getToolAnalogInput(int index);
04240
04241 /**
04242 * @ingroup IoControl
04243 * \chinese
04244 *
04245 *
04246 * @param index: IO      0
04247 *      0
04248 *
04249 * @return
04250 *
04251 * @throws arcs::common_interface::AuboException
04252 *
04253 * @par Python
04254 * getStandardAnalogOutput(self: pyaubo_sdk.IoControl, arg0: int) -> float
04255 *
04256 * @par Lua
04257 * getStandardAnalogOutput(index: number) -> number
04258 *
04259 * @par Lua
04260 * num = getStandardAnalogOutput(0)
04261 *
04262 * @par JSON-RPC
04263 * {"jsonrpc":"2.0","method":"rob1.IoControl.getStandardAnalogOutput","params":[0],"id":1}
04264 *
04265 * @par JSON-RPC
04266 * {"id":1,"jsonrpc":"2.0","result":0.0}
04267 *
04268 * \endchinese
04269 * \english
04270 * Get the value of standard analog output.
04271 *
04272 * @param index: Indicates the IO pin, starting from 0.
04273 * For example, 0 means the first pin.
04274 *
04275 * @return Standard analog output value.
04276 *
04277 * @throws arcs::common_interface::AuboException
04278 *
04279 * @par Python function prototype
04280 * getStandardAnalogOutput(self: pyaubo_sdk.IoControl, arg0: int) -> float
04281 *
04282 * @par Lua function prototype
04283 * getStandardAnalogOutput(index: number) -> number
04284 *
04285 * @par Lua example
04286 * num = getStandardAnalogOutput(0)
04287 *
04288 * @par JSON-RPC request example
04289 * {"jsonrpc":"2.0","method":"rob1.IoControl.getStandardAnalogOutput","params":[0],"id":1}
04290 *
04291 * @par JSON-RPC response example
04292 * {"id":1,"jsonrpc":"2.0","result":0.0}
04293 *
04294 * \endenglish
04295 */
04296 double getStandardAnalogOutput(int index);
04297
04298 /**
04299 * @ingroup IoControl
04300 * \chinese
04301 *
04302 *
04303 * @param index: IO      0
04304 *      0
04305 *
04306 * @return
04307 *
04308 * @throws arcs::common_interface::AuboException

```

```

04309 *
04310 * @par Python
04311 * getToolAnalogOutput(self: pyaubo_sdk.IoControl, arg0: int) -> float
04312 *
04313 * @par Lua
04314 * getToolAnalogOutput(index: number) -> number
04315 *
04316 * @par Lua
04317 * num = getToolAnalogOutput(0)
04318 *
04319 * @par JSON-RPC
04320 * {"jsonrpc":"2.0","method":"rob1.IoControl.getToolAnalogOutput","params":[0],"id":1}
04321 *
04322 * @par JSON-RPC
04323 * {"id":1,"jsonrpc":"2.0","result":0.0}
04324 *
04325 * \endchinese
04326 * \english
04327 * Get the value of tool analog output.
04328 *
04329 * @param index: Indicates the IO pin, starting from 0.
04330 * For example, 0 means the first pin.
04331 *
04332 * @return Tool analog output value.
04333 *
04334 * @throws arcs::common_interface::AuboException
04335 *
04336 * @par Python function prototype
04337 * getToolAnalogOutput(self: pyaubo_sdk.IoControl, arg0: int) -> float
04338 *
04339 * @par Lua function prototype
04340 * getToolAnalogOutput(index: number) -> number
04341 *
04342 * @par Lua example
04343 * num = getToolAnalogOutput(0)
04344 *
04345 * @par JSON-RPC request example
04346 * {"jsonrpc":"2.0","method":"rob1.IoControl.getToolAnalogOutput","params":[0],"id":1}
04347 *
04348 * @par JSON-RPC response example
04349 * {"id":1,"jsonrpc":"2.0","result":0.0}
04350 *
04351 * \endenglish
04352 */
04353 double getToolAnalogOutput(int index);
04354
04355 /**
04356 * @ingroup IoControl
04357 * \chinese
04358 *
04359 *
04360 * @return
04361 *
04362 * @throws arcs::common_interface::AuboException
04363 *
04364 * @par Python
04365 * getStaticLinkInputNum(self: pyaubo_sdk.IoControl) -> int
04366 *
04367 * @par Lua
04368 * getStaticLinkInputNum() -> number
04369 *
04370 * @par Lua
04371 * num = getStaticLinkInputNum()
04372 *
04373 * @par JSON-RPC
04374 * {"jsonrpc":"2.0","method":"rob1.IoControl.getStaticLinkInputNum","params":[],"id":1}
04375 *
04376 * @par JSON-RPC
04377 * {"id":1,"jsonrpc":"2.0","result":8}
04378 *
04379 * \endchinese
04380 * \english
04381 * Get the number of static link inputs.
04382 *
04383 * @return Number of static link inputs.
04384 *
04385 * @throws arcs::common_interface::AuboException
04386 *
04387 * @par Python function prototype
04388 * getStaticLinkInputNum(self: pyaubo_sdk.IoControl) -> int
04389 *
04390 * @par Lua function prototype
04391 * getStaticLinkInputNum() -> number
04392 *
04393 * @par Lua example
04394 * num = getStaticLinkInputNum()
04395 *

```

```

04396 * @par JSON-RPC request example
04397 * {"jsonrpc":"2.0","method":"rob1.IoControl.getStaticLinkInputNum","params":[],"id":1}
04398 *
04399 * @par JSON-RPC response example
04400 * {"id":1,"jsonrpc":"2.0","result":8}
04401 *
04402 * \endenglish
04403 */
04404 int getStaticLinkInputNum();
04405
04406 /**
04407 * @ingroup IoControl
04408 * \chinese
04409 *
04410 *
04411 * @return
04412 *
04413 * @throws arcs::common_interface::AuboException
04414 *
04415 * @par Python
04416 * getStaticLinkOutputNum(self: pyaubo_sdk.IoControl) -> int
04417 *
04418 * @par Lua
04419 * getStaticLinkOutputNum() -> number
04420 *
04421 * @par Lua
04422 * num = getStaticLinkOutputNum()
04423 *
04424 * @par JSON-RPC
04425 * {"jsonrpc":"2.0","method":"rob1.IoControl.getStaticLinkOutputNum","params":[],"id":1}
04426 *
04427 * @par JSON-RPC
04428 * {"id":1,"jsonrpc":"2.0","result":0}
04429 *
04430 * \endchinese
04431 * \english
04432 * Get the number of static link outputs.
04433 *
04434 * @return Number of static link outputs.
04435 *
04436 * @throws arcs::common_interface::AuboException
04437 *
04438 * @par Python function prototype
04439 * getStaticLinkOutputNum(self: pyaubo_sdk.IoControl) -> int
04440 *
04441 * @par Lua function prototype
04442 * getStaticLinkOutputNum() -> number
04443 *
04444 * @par Lua example
04445 * num = getStaticLinkOutputNum()
04446 *
04447 * @par JSON-RPC request example
04448 * {"jsonrpc":"2.0","method":"rob1.IoControl.getStaticLinkOutputNum","params":[],"id":1}
04449 *
04450 * @par JSON-RPC response example
04451 * {"id":1,"jsonrpc":"2.0","result":0}
04452 *
04453 * \endenglish
04454 */
04455 int getStaticLinkOutputNum();
04456
04457 /**
04458 * @ingroup IoControl
04459 * \chinese
04460 *
04461 *
04462 * @return \n
04463 *      2863267846 , 2  101010101010101000000000000000110
04464 * 16
04465 *      0      ,      1      \n
04466 * 1      0      .
04467 *
04468 * @throws arcs::common_interface::AuboException
04469 *
04470 * @par Python
04471 * getStaticLinkInputs(self: pyaubo_sdk.IoControl) -> int
04472 *
04473 * @par Lua
04474 * getStaticLinkInputs() -> number
04475 *
04476 * @par Lua
04477 * num = getStaticLinkInputs()
04478 *
04479 * @par JSON-RPC
04480 * {"jsonrpc":"2.0","method":"rob1.IoControl.getStaticLinkInputs","params":[],"id":1}
04481 *
04482 * @par JSON-RPC

```

```

04483 * {"id":1,"jsonrpc":"2.0","result":0}
04484 *
04485 * \endchinese
04486 * \english
04487 * Get all static link input values.
04488 *
04489 * @return All static link input values.\n
04490 * For example, if the return value is 2863267846, its binary representation
04491 * is 1010101010101000000000000000110. The lower 16 bits represent the
04492 * status of all static link inputs, the least significant bit indicates the
04493 * input status of pin 0, the second least significant bit indicates pin 1,
04494 * and so on.\n 1 means high level, 0 means low level.
04495 *
04496 * @throws arcs::common_interface::AuboException
04497 *
04498 * @par Python function prototype
04499 * getStaticLinkInputs(self: pyaubo_sdk.IoControl) -> int
04500 *
04501 * @par Lua function prototype
04502 * getStaticLinkInputs() -> number
04503 *
04504 * @par Lua example
04505 * num = getStaticLinkInputs()
04506 *
04507 * @par JSON-RPC request example
04508 * {"jsonrpc":"2.0","method":"rob1.IoControl.getStaticLinkInputs","params":[],"id":1}
04509 *
04510 * @par JSON-RPC response example
04511 * {"id":1,"jsonrpc":"2.0","result":0}
04512 *
04513 * \endenglish
04514 */
04515 uint32_t getStaticLinkInputs();
04516
04517 /**
04518 * @ingroup IoControl
04519 * \chinese
04520 *
04521 *
04522 * @return \n
04523 *      2863267846, 2  1010101010101000000000000000110
04524 *      16
04525 *      0    ,    1    \n
04526 *      1    0    .
04527 *
04528 * @throws arcs::common_interface::AuboException
04529 *
04530 * @par Python
04531 * getStaticLinkOutputs(self: pyaubo_sdk.IoControl) -> int
04532 *
04533 * @par Lua
04534 * getStaticLinkOutputs() -> number
04535 *
04536 * @par Lua
04537 * num = getStaticLinkOutputs()
04538 *
04539 * @par JSON-RPC
04540 * {"jsonrpc":"2.0","method":"rob1.IoControl.getStaticLinkOutputs","params":[],"id":1}
04541 *
04542 * @par JSON-RPC
04543 * {"id":1,"jsonrpc":"2.0","result":0}
04544 *
04545 * \endchinese
04546 * \english
04547 * Get all static link output values.
04548 *
04549 * @return Returns all static link output values.\n
04550 * For example, if the return value is 2863267846, its binary representation
04551 * is 1010101010101000000000000000110. The lower 16 bits represent the
04552 * status of all static link outputs, the least significant bit indicates
04553 * the output status of pin 0, the second least significant bit indicates
04554 * pin 1, and so on.\n 1 means high level, 0 means low level.
04555 *
04556 * @throws arcs::common_interface::AuboException
04557 *
04558 * @par Python function prototype
04559 * getStaticLinkOutputs(self: pyaubo_sdk.IoControl) -> int
04560 *
04561 * @par Lua function prototype
04562 * getStaticLinkOutputs() -> number
04563 *
04564 * @par Lua example
04565 * num = getStaticLinkOutputs()
04566 *
04567 * @par JSON-RPC request example
04568 * {"jsonrpc":"2.0","method":"rob1.IoControl.getStaticLinkOutputs","params":[],"id":1}
04569 *

```



```

04570 * @par JSON-RPC response example
04571 * {"id":1,"jsonrpc":"2.0","result":0}
04572 *
04573 * \endenglish
04574 */
04575 uint32_t getStaticLinkOutputs();
04576
04577 /**
04578 * @ingroup IoControl
04579 * \chinese
04580 *
04581 * 0
04582 *
04583 * @return true, false
04584 *
04585 * @throws arcs::common_interface::AuboException
04586 *
04587 * @par JSON-RPC
04588 * {"jsonrpc":"2.0","method":"rob1.IoControl.hasEncoderSensor","params":[],"id":1}
04589 *
04590 * @par JSON-RPC
04591 * {"id":1,"jsonrpc":"2.0","result":true}
04592 *
04593 * \endchinese
04594 * \english
04595 * Whether the robot is equipped with an encoder.
04596 * The integrated encoder number is 0.
04597 *
04598 * @return Returns true if the robot is equipped with an encoder, otherwise
04599 * false.
04600 *
04601 * @throws arcs::common_interface::AuboException
04602 *
04603 * @par JSON-RPC request example
04604 * {"jsonrpc":"2.0","method":"rob1.IoControl.hasEncoderSensor","params":[],"id":1}
04605 *
04606 * @par JSON-RPC response example
04607 * {"id":1,"jsonrpc":"2.0","result":true}
04608 *
04609 * \endenglish
04610 */
04611 bool hasEncoderSensor();
04612 /**
04613 * @ingroup IoControl
04614 * \chinese
04615 *
04616 *
04617 * @param type
04618 * 0-
04619 * 1-AB
04620 * 2-AB +Z
04621 * 3-AB
04622 * 4-AB +Z
04623 *
04624 * @param range_id
04625 * 0 32 [-2147483648, 2147483647]
04626 * 1 8 [0, 255]
04627 * 2 16 [0, 65535]
04628 * 3 24 [0, 16777215]
04629 * 4 32 [0, 4294967295]
04630 *
04631 * @return 0
04632 * AUBO_NO_ACCESS
04633 * AUBO_BUSY
04634 * AUBO_BAD_STATE
04635 * -AUBO_INVL_ARGUMENT
04636 * -AUBO_BAD_STATE
04637 *
04638 * @throws arcs::common_interface::AuboException
04639 * \endchinese
04640 * \english
04641 * Set the decoding method of the integrated encoder.
04642 *
04643 * @param type
04644 * 0 - Disable encoder
04645 * 1 - AB quadrature
04646 * 2 - AB quadrature + Z
04647 * 3 - AB differential quadrature
04648 * 4 - AB differential quadrature + Z differential
04649 *
04650 * @param range_id
04651 * 0 is a 32-bit signed encoder, range [-2147483648, 2147483647]
04652 * 1 is an 8-bit unsigned encoder, range [0, 255]
04653 * 2 is a 16-bit unsigned encoder, range [0, 65535]
04654 * 3 is a 24-bit unsigned encoder, range [0, 16777215]
04655 * 4 is a 32-bit unsigned encoder, range [0, 4294967295]
04656 *

```

```

04657 * @return Returns 0 on success; error code on failure.
04658 * AUBO_NO_ACCESS
04659 * AUBO_BUSY
04660 * AUBO_BAD_STATE
04661 * -AUBO_INVL_ARGUMENT
04662 * -AUBO_BAD_STATE
04663 *
04664 * @throws arcs::common_interface::AuboException
04665 * \endenglish
04666 */
04667 int setEncDecoderType(int type, int range_id);
04668
04669 /**
04670 * @ingroup IoControl
04671 * \chinese
04672 *
04673 *
04674 * @param tick
04675 *
04676 * @return 0
04677 * AUBO_NO_ACCESS
04678 * AUBO_BUSY
04679 * AUBO_BAD_STATE
04680 * -AUBO_INVL_ARGUMENT
04681 * -AUBO_BAD_STATE
04682 *
04683 * @throws arcs::common_interface::AuboException
04684 * \endchinese
04685 * \english
04686 * Set the tick count of the integrated encoder.
04687 *
04688 * @param tick Tick count
04689 *
04690 * @return Returns 0 on success; error code on failure.
04691 * AUBO_NO_ACCESS
04692 * AUBO_BUSY
04693 * AUBO_BAD_STATE
04694 * -AUBO_INVL_ARGUMENT
04695 * -AUBO_BAD_STATE
04696 *
04697 * @throws arcs::common_interface::AuboException
04698 * \endenglish
04699 */
04700 int setEncTickCount(int tick);
04701
04702 /**
04703 * @ingroup IoControl
04704 * \chinese
04705 *
04706 *
04707 * @return 0
04708 * AUBO_NO_ACCESS
04709 * AUBO_BUSY
04710 * AUBO_BAD_STATE
04711 * -AUBO_BAD_STATE
04712 *
04713 * @throws arcs::common_interface::AuboException
04714 *
04715 * @par JSON-RPC
04716 * {"jsonrpc":"2.0","method":"rob1.IoControl.getEncDecoderType","params":[],"id":1}
04717 *
04718 * @par JSON-RPC
04719 * {"id":1,"jsonrpc":"2.0","result":0}
04720 *
04721 * \endchinese
04722 * \english
04723 * Get the decoder type of the encoder.
04724 *
04725 * @return Returns 0 on success; error code on failure.
04726 * AUBO_NO_ACCESS
04727 * AUBO_BUSY
04728 * AUBO_BAD_STATE
04729 * -AUBO_BAD_STATE
04730 *
04731 * @throws arcs::common_interface::AuboException
04732 *
04733 * @par JSON-RPC request example
04734 * {"jsonrpc":"2.0","method":"rob1.IoControl.getEncDecoderType","params":[],"id":1}
04735 *
04736 * @par JSON-RPC response example
04737 * {"id":1,"jsonrpc":"2.0","result":0}
04738 *
04739 * \endenglish
04740 */
04741 int getEncDecoderType();
04742
04743 /**

```

```

04744 * @ingroup IoControl
04745 * \chinese
04746 *
04747 *
04748 * @return 0
04749 * AUBO_NO_ACCESS
04750 * AUBO_BUSY
04751 * AUBO_BAD_STATE
04752 * -AUBO_BAD_STATE
04753 *
04754 * @throws arcs::common_interface::AuboException
04755 *
04756 * @par JSON-RPC
04757 * {"jsonrpc":"2.0","method":"rob1.IoControl.getEncTickCount","params":[],"id":1}
04758 *
04759 * @par JSON-RPC
04760 * {"id":1,"jsonrpc":"2.0","result":0}
04761 *
04762 * \endchinese
04763 * \english
04764 * Get the tick count
04765 *
04766 * @return Returns 0 on success; error code on failure.
04767 * AUBO_NO_ACCESS
04768 * AUBO_BUSY
04769 * AUBO_BAD_STATE
04770 * -AUBO_BAD_STATE
04771 *
04772 * @throws arcs::common_interface::AuboException
04773 *
04774 * @par JSON-RPC request example
04775 * {"jsonrpc":"2.0","method":"rob1.IoControl.getEncTickCount","params":[],"id":1}
04776 *
04777 * @par JSON-RPC response example
04778 * {"id":1,"jsonrpc":"2.0","result":0}
04779 *
04780 * \endenglish
04781 */
04782 int getEncTickCount();
04783
04784 /**
04785 * @ingroup IoControl
04786 * \chinese
04787 *
04788 *
04789 * @param delta_count
04790 *
04791 * @return 0
04792 * AUBO_NO_ACCESS
04793 * AUBO_BUSY
04794 * AUBO_BAD_STATE
04795 * -AUBO_BAD_STATE
04796 *
04797 * @throws arcs::common_interface::AuboException
04798 * \endchinese
04799 * \english
04800 * Prevent counting errors when the count exceeds the range
04801 *
04802 * @param delta_count
04803 *
04804 * @return Returns 0 on success; error code on failure.
04805 * AUBO_NO_ACCESS
04806 * AUBO_BUSY
04807 * AUBO_BAD_STATE
04808 * -AUBO_BAD_STATE
04809 *
04810 * @throws arcs::common_interface::AuboException
04811 * \endenglish
04812 */
04813 int unwindEncDeltaTickCount(int delta_count);
04814
04815 /**
04816 * @ingroup IoControl
04817 * \chinese
04818 *
04819 *
04820 * @return true; false
04821 *
04822 * @throws arcs::common_interface::AuboException
04823 *
04824 * @par Python
04825 * getToolButtonStatus() -> bool
04826 *
04827 * @par Lua
04828 * getToolButtonStatus() -> boolean
04829 *
04830 * @par Lua

```

```

04831     * status = getToolButtonStatus()
04832     *
04833     * @par JSON-RPC
04834     * {"jsonrpc":"2.0","method":"rob1.IoControl.getToolButtonStatus","params":[],"id":1}
04835     *
04836     * @par JSON-RPC
04837     * {"id":1,"jsonrpc":"2.0","result":false}
04838     *
04839     * \endchinese
04840     * \english
04841     * Get the status of the tool button.
04842     *
04843     * @return Returns true if pressed; otherwise false.
04844     *
04845     * @throws arcs::common_interface::AuboException
04846     *
04847     * @par Python function prototype
04848     * getToolButtonStatus() -> bool
04849     *
04850     * @par Lua function prototype
04851     * getToolButtonStatus() -> boolean
04852     *
04853     * @par Lua example
04854     * status = getToolButtonStatus()
04855     *
04856     * @par JSON-RPC request example
04857     * {"jsonrpc":"2.0","method":"rob1.IoControl.getToolButtonStatus","params":[],"id":1}
04858     *
04859     * @par JSON-RPC response example
04860     * {"id":1,"jsonrpc":"2.0","result":false}
04861     *
04862     * \endenglish
04863     */
04864 bool getToolButtonStatus();
04865
04866 /**
04867  * @ingroup IoControl
04868  * \chinese
04869  *
04870  *
04871  * @note
04872  *
04873  * @return \n
04874  *      2863267846 , 2   1010101010101000000000000000110
04875  *      16
04876  *      0   ,   1   \n
04877  *      1   0
04878  *
04879  * @throws arcs::common_interface::AuboException
04880  *
04881  * @par Python
04882  * getHandleIoStatus(self: pyaubo_sdk.IoControl) -> int
04883  *
04884  * @par Lua
04885  * getHandleIoStatus() -> number
04886  *
04887  * @par Lua
04888  * num = getHandleIoStatus()
04889  *
04890  * @par JSON-RPC
04891  * {"jsonrpc":"2.0","method":"rob1.IoControl.getHandleIoStatus","params":[],"id":1}
04892  *
04893  * @par JSON-RPC
04894  * {"id":1,"jsonrpc":"2.0","result":0}
04895  *
04896  * \endchinese
04897  * \english
04898  * Get the status of handle buttons.
04899  *
04900  * @note Get the status of handle buttons.
04901  *
04902  * @return All handle button input values.\n
04903  * For example, if the return value is 2863267846, its binary representation
04904  * is 101010101010101000000000000000110. The lower 16 bits represent the
04905  * status of all input pins, the least significant bit indicates the input
04906  * status of pin 0, the second least significant bit indicates pin 1, and so
04907  * on.\n 1 means high level, 0 means low level.
04908  *
04909  * @throws arcs::common_interface::AuboException
04910  *
04911  * @par Python function prototype
04912  * getHandleIoStatus(self: pyaubo_sdk.IoControl) -> int
04913  *
04914  * @par Lua function prototype
04915  * getHandleIoStatus() -> number
04916  *
04917  * @par Lua example

```

```

04918     * num = getHandleIoStatus()
04919     *
04920     * @par JSON-RPC request example
04921     * {"jsonrpc":"2.0","method":"rob1.IoControl.getHandleIoStatus","params":[],"id":1}
04922     *
04923     * @par JSON-RPC response example
04924     * {"id":1,"jsonrpc":"2.0","result":0}
04925     *
04926     * \endenglish
04927     */
04928     uint32_t getHandleIoStatus();
04929
04930     /**
04931     * @ingroup IoControl
04932     * \chinese
04933     *
04934     *
04935     * @return type
04936     *
04937     * @throws arcs::common_interface::AuboException
04938     *
04939     * @par Python
04940     * getHandleType() -> int
04941     *
04942     * @par Lua
04943     * getHandleType() -> int
04944     *
04945     * @par Lua
04946     * int_num = getHandleType()
04947     *
04948     * @par JSON-RPC
04949     * {"jsonrpc":"2.0","method":"rob1.IoControl.getHandleType","params":[],"id":1}
04950     *
04951     * @par JSON-RPC
04952     * {"id":1,"jsonrpc":"2.0","result":0}
04953     *
04954     * \endchinese
04955     * \english
04956     * Get the handle type.
04957     *
04958     * @return type
04959     *
04960     * @throws arcs::common_interface::AuboException
04961     *
04962     * @par Python function prototype
04963     * getHandleType() -> int
04964     *
04965     * @par Lua function prototype
04966     * getHandleType() -> int
04967     *
04968     * @par Lua example
04969     * int_num = getHandleType()
04970     *
04971     * @par JSON-RPC request example
04972     * {"jsonrpc":"2.0","method":"rob1.IoControl.getHandleType","params":[],"id":1}
04973     *
04974     * @par JSON-RPC response example
04975     * {"id":1,"jsonrpc":"2.0","result":0}
04976     *
04977     * \endenglish
04978     */
04979     int getHandleType();
04980
04981     protected:
04982     void *d_;
04983 };
04984 using IoControlPtr = std::shared_ptr<IoControl>;
04985 } // namespace common_interface
04986 } // namespace arcs
04987
04988 #endif // AUBO_SDK_IO_CONTROL_INTERFACE_H

```

12.24 include/aubo/robot/motion_control.h File Reference

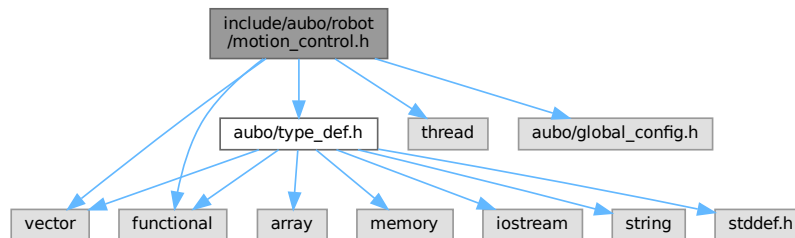
Motion control interface.

```

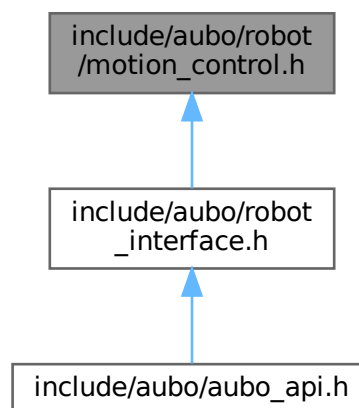
#include <vector>
#include <functional>
#include <thread>

```

```
#include <aubo/global_config.h>
#include <aubo/type_def.h>
Include dependency graph for motion_control.h:
```



This graph shows which files directly or indirectly include this file:



Classes

- class [arcs::common_interface::MotionControl](#)

Namespaces

- namespace [arcs](#)
- namespace [arcs::common_interface](#)

Typedefs

- using [arcs::common_interface::MotionControlPtr](#) = `std::shared_ptr<MotionControl>`

Enumerations

- enum `arcs::common_interface::PathBufferType` {
`arcs::common_interface::PathBuffer_TOPPRA = 1`, `arcs::common_interface::PathBuffer_CubicSpline`
`, arcs::common_interface::PathBuffer_JointSpline = 3`, `arcs::common_interface::PathBuffer_JointSplineC`
`= 4`,
`arcs::common_interface::PathBuffer_JointBSpline = 5`, `arcs::common_interface::PathBuffer_JointBSplineC`
`= 6` }
`pathBuffer` Type

12.24.1 Detailed Description

Motion control interface.

Definition in file [motion_control.h](#).

12.25 motion_control.h

[Go to the documentation of this file.](#)

```
00001 /** @file motion_control.h
00002 * \~chinese @brief
00003 * \~english @brief Motion control interface
00004 */
00005 #ifndef AUBO_SDK_MOTION_CONTROL_INTERFACE_H
00006 #define AUBO_SDK_MOTION_CONTROL_INTERFACE_H
00007
00008 #include <vector>
00009 #include <functional>
00010 #include <thread>
00011
00012 #include <aubo/global_config.h>
00013 #include <aubo/type_def.h>
00014
00015 namespace arcs {
00016 namespace common_interface {
00017
00018 /**
00019 * \~chinese
00020 * pathBuffer
00021 *
00022 *
00023 * \endchinese
00024 * \english
00025 *
00026 * pathBuffer Type
00027 *
00028 * The robot movements are programmed as pose-to-pose movements, that is move
00029 * from the current position to a new position. The path between these two
00030 * positions is then automatically calculated by the robot.
00031 * \endenglish
00032 */
00033
00034 enum PathBufferType
00035 {
00036     PathBuffer_TOPPRA = 1, ///< \~chinese 1: toppra \~english
00037                             ///< 1: toppra time optimal path planning
00038     PathBuffer_CubicSpline =
00039         2, ///< \~chinese 2: cubic_spline( ) \~english 2: cubic_spline
00040           ///< (recorded trajectory)
00041
00042     ///< \~chinese 3: B \~english 3: Joint B-spline
00043     ///< interpolation, at least three points
00044     ///< \~chinese 5 CUBIC_SPLINE \~english
00045     ///< Deprecated, use 5 instead, currently it is joint space CUBIC_SPLINE
00046     PathBuffer_JointSpline = 3,
00047
00048     ///< \~chinese 4: B
00049     ///< \~english 4: Joint B-spline interpolation, at least three points, but
00050     ///< the input is Cartesian space pose 6
00051     ///< CUBIC_SPLINE
00052     PathBuffer_JointSplineC = 4,
```

```

00053
00054 ///< ~chinese 5: B      \~english 5: Joint B-spline
00055 ///< interpolation, at least three points
00056 PathBuffer_JointBSpline = 5,
00057
00058 ///< ~chinese 6: B
00059 ///< ~english 6: Joint B-spline interpolation, at least three points, but
00060 ///< the input is Cartesian space pose
00061 PathBuffer_JointBSplineC = 6,
00062 };
00063
00064 /**
00065 * @defgroup MotionControl MotionControl ( )
00066 * @ingroup RobotInterface
00067 * MotionControl
00068 */
00069 class ARCS_ABI_EXPORT MotionControl
00070 {
00071 public:
00072     MotionControl();
00073     virtual ~MotionControl();
00074
00075     /**
00076      * @ingroup MotionControl
00077      * \chinese
00078      * m
00079      * moveLine/moveCircle
00080      * setEqradius 1
00081      *
00082      * @return
00083      *
00084      * @throws arcs::common_interface::AuboException
00085      *
00086      * @par Python
00087      * getEqradius(self: pyaubo_sdk.MotionControl) -> float
00088      *
00089      * @par Lua
00090      * getEqradius() -> number
00091      *
00092      * @par Lua
00093      * num = getEqradius()
00094      *
00095      * @par JSON-RPC
00096      * {"jsonrpc":"2.0","method":"rob1.MotionControl.getEqradius","params":[],"id":1}
00097      *
00098      * @par JSON-RPC
00099      * {"id":1,"jsonrpc":"2.0","result":1.0}
00100      * \endchinese
00101      * \english
00102      * Get the equivalent radius, in meters.
00103      * When using moveLine/moveCircle, the rotation angle of the end effector's
00104      * pose is converted to an equivalent end position movement. Can be set via
00105      * setEqradius, default is 1.
00106      *
00107      * @return Returns the equivalent radius.
00108      *
00109      * @throws arcs::common_interface::AuboException
00110      *
00111      * @par Python function prototype
00112      * getEqradius(self: pyaubo_sdk.MotionControl) -> float
00113      *
00114      * @par Lua function prototype
00115      * getEqradius() -> number
00116      *
00117      * @par Lua example
00118      * num = getEqradius()
00119      *
00120      * @par JSON-RPC request example
00121      * {"jsonrpc":"2.0","method":"rob1.MotionControl.getEqradius","params":[],"id":1}
00122      *
00123      * @par JSON-RPC response example
00124      * {"id":1,"jsonrpc":"2.0","result":1.0}
00125      * \endenglish
00126      */
00127     double getEqradius();
00128
00129     /**
00130      * @ingroup MotionControl
00131      * \chinese
00132      * m
00133      * moveLine/moveCircle
00134      *
00135      * @param eqradius 0
00136      *
00137      * @return 0;
00138      * AUBO_BAD_STATE
00139      * AUBO_BUSY

```



```

00140 * -AUBO_INVL_ARGUMENT
00141 * -AUBO_BAD_STATE
00142 *
00143 * @throws arcs::common_interface::AuboException
00144 *
00145 * @par JSON-RPC
00146 * {"jsonrpc":"2.0","method":"rob1.MotionControl.setEqradius","params":[0.8],"id":1}
00147 *
00148 * @par JSON-RPC
00149 * {"id":1,"jsonrpc":"2.0","result":0}
00150 *
00151 * @par Python
00152 * setEqradius(self: pyaubo_sdk.MotionControl, arg0: float) -> int
00153 *
00154 * @par Lua
00155 * setEqradius(eqradius: number) -> number
00156 *
00157 * @par Lua
00158 * num = setEqradius(1)
00159 *
00160 * \endchinese
00161 * \english
00162 * Set the equivalent radius, in meters.
00163 * When using moveLine/moveCircle, the rotation angle of the end effector's
00164 * pose is converted to an equivalent end position movement. The larger the
00165 * value, the faster the posture rotation speed.
00166 *
00167 * @param eqradius 0 means only plan the movement, and the posture rotation
00168 * follows the movement.
00169 *
00170 * @return Returns 0 on success; otherwise returns an error code:
00171 * AUBO_BAD_STATE
00172 * AUBO_BUSY
00173 * -AUBO_INVL_ARGUMENT
00174 * -AUBO_BAD_STATE
00175 *
00176 * @throws arcs::common_interface::AuboException
00177 *
00178 * @par JSON-RPC request example
00179 * {"jsonrpc":"2.0","method":"rob1.MotionControl.setEqradius","params":[0.8],"id":1}
00180 *
00181 * @par JSON-RPC response example
00182 * {"id":1,"jsonrpc":"2.0","result":0}
00183 *
00184 * @par Python function prototype
00185 * setEqradius(self: pyaubo_sdk.MotionControl, arg0: float) -> int
00186 *
00187 * @par Lua function prototype
00188 * setEqradius(eqradius: number) -> number
00189 *
00190 * @par Lua example
00191 * num = setEqradius(1)
00192 *
00193 * \endenglish
00194 */
00195 int setEqradius(double eqradius);
00196
00197 /**
00198 * @ingroup MotionControl
00199 * \chinese
00200 * (0., 1.]
00201 *
00202 * @param fraction
00203 * @return 0;
00204 * AUBO_INVL_ARGUMENT
00205 * AUBO_BUSY
00206 * AUBO_BAD_STATE
00207 * -AUBO_INVL_ARGUMENT
00208 * -AUBO_BAD_STATE
00209 *
00210 * @throws arcs::common_interface::AuboException
00211 *
00212 * @par Python
00213 * setSpeedFraction(self: pyaubo_sdk.MotionControl, arg0: float) -> int
00214 *
00215 * @par Lua
00216 * setSpeedFraction(fraction: number) -> nil
00217 *
00218 * @par Lua
00219 * setSpeedFraction(0.5)
00220 *
00221 * @par JSON-RPC
00222 * {"jsonrpc":"2.0","method":"rob1.MotionControl.setSpeedFraction","params":[0.8],"id":1}
00223 *
00224 * @par JSON-RPC
00225 * {"id":1,"jsonrpc":"2.0","result":0}
00226 * \endchinese

```

```

00227 * \english
00228 * Dynamically adjust the robot's speed and acceleration ratio (0., 1.]
00229 *
00230 * @param fraction The ratio of robot speed and acceleration
00231 * @return Returns 0 on success; otherwise returns an error code:
00232 * AUBO_INVL_ARGUMENT
00233 * AUBO_BUSY
00234 * AUBO_BAD_STATE
00235 * -AUBO_INVL_ARGUMENT
00236 * -AUBO_BAD_STATE
00237 *
00238 * @throws arcs::common_interface::AuboException
00239 *
00240 * @par Python function prototype
00241 * setSpeedFraction(self: pyaubo_sdk.MotionControl, arg0: float) -> int
00242 *
00243 * @par Lua function prototype
00244 * setSpeedFraction(fraction: number) -> nil
00245 *
00246 * @par Lua example
00247 * setSpeedFraction(0.5)
00248 *
00249 * @par JSON-RPC request example
00250 * {"jsonrpc": "2.0", "method": "rob1.MotionControl.setSpeedFraction", "params": [0.8], "id": 1}
00251 *
00252 * @par JSON-RPC response example
00253 * {"id": 1, "jsonrpc": "2.0", "result": 0}
00254 * \endenglish
00255 */
00256 int setSpeedFraction(double fraction);
00257
00258 /**
00259 * @ingroup MotionControl
00260 * \chinese
00261 * 1
00262 * setSpeedFraction
00263 *
00264 * @return
00265 *
00266 * @throws arcs::common_interface::AuboException
00267 *
00268 * @par Python
00269 * getSpeedFraction(self: pyaubo_sdk.MotionControl) -> float
00270 *
00271 * @par Lua
00272 * getSpeedFraction() -> number
00273 *
00274 * @par Lua
00275 * num = getSpeedFraction()
00276 *
00277 * @par JSON-RPC
00278 * {"jsonrpc": "2.0", "method": "rob1.MotionControl.getSpeedFraction", "params": [], "id": 1}
00279 *
00280 * @par JSON-RPC
00281 * {"id": 1, "jsonrpc": "2.0", "result": 1.0}
00282 * \endchinese
00283 * \english
00284 * Get the speed and acceleration ratio, default is 1.
00285 * Can be set via setSpeedFraction interface.
00286 *
00287 * @return Returns the speed and acceleration ratio.
00288 *
00289 * @throws arcs::common_interface::AuboException
00290 *
00291 * @par Python function prototype
00292 * getSpeedFraction(self: pyaubo_sdk.MotionControl) -> float
00293 *
00294 * @par Lua function prototype
00295 * getSpeedFraction() -> number
00296 *
00297 * @par Lua example
00298 * num = getSpeedFraction()
00299 *
00300 * @par JSON-RPC request example
00301 * {"jsonrpc": "2.0", "method": "rob1.MotionControl.getSpeedFraction", "params": [], "id": 1}
00302 *
00303 * @par JSON-RPC response example
00304 * {"id": 1, "jsonrpc": "2.0", "result": 1.0}
00305 * \endenglish
00306 */
00307 double getSpeedFraction();
00308
00309 /**
00310 * @ingroup MotionControl
00311 * \chinese
00312 * 1. ,
00313 *

```

```

00314 *
00315 * @param enable
00316 * @return 0;
00317 * AUBO_BAD_STATE
00318 * AUBO_BUSY
00319 * -AUBO_BAD_STATE
00320 *
00321 * @throws arcs::common_interface::AuboException
00322 *
00323 * @par Lua
00324 * speedFractionCritical() -> bool
00325 *
00326 * @par Lua
00327 * speedFractionCritical(true)
00328 *
00329 * @par JSON-RPC
00330 * {"jsonrpc":"2.0","method":"rob1.MotionControl.speedFractionCritical","params":[true],"id":1}
00331 *
00332 * @par JSON-RPC
00333 * {"id":1,"jsonrpc":"2.0","result":0}
00334 * \endchinese
00335 * \english
00336 * Speed fraction critical section. When enabled, the speed fraction is
00337 * forced to 1. When disabled, the previous speed fraction is restored.
00338 *
00339 * @param enable
00340 * @return Returns 0 on success; otherwise returns an error code:
00341 * AUBO_BAD_STATE
00342 * AUBO_BUSY
00343 * -AUBO_BAD_STATE
00344 *
00345 * @throws arcs::common_interface::AuboException
00346 *
00347 * @par Lua function prototype
00348 * speedFractionCritical() -> nil
00349 *
00350 * @par Lua example
00351 * speedFractionCritical(true)
00352 *
00353 * @par JSON-RPC request example
00354 * {"jsonrpc":"2.0","method":"rob1.MotionControl.speedFractionCritical","params":[true],"id":1}
00355 *
00356 * @par JSON-RPC response example
00357 * {"id":1,"jsonrpc":"2.0","result":0}
00358 * \endenglish
00359 */
00360 int speedFractionCritical(bool enable);
00361
00362 /**
00363 * @ingroup MotionControl
00364 * \chinese
00365 *
00366 *
00367 * @return true; false
00368 *
00369 * @throws arcs::common_interface::AuboException
00370 *
00371 * @par Lua
00372 * isSpeedFractionCritical() -> bool
00373 *
00374 * @par Lua
00375 * status = isSpeedFractionCritical()
00376 *
00377 * @par JSON-RPC
00378 * {"jsonrpc":"2.0","method":"rob1.MotionControl.isSpeedFractionCritical","params":[],"id":1}
00379 *
00380 * @par JSON-RPC
00381 * {"id":1,"jsonrpc":"2.0","result":true}
00382 * \endchinese
00383 * \english
00384 * Whether it is in the speed fraction critical section
00385 *
00386 * @return Returns true if in the speed fraction critical section; otherwise
00387 * returns false
00388 *
00389 * @throws arcs::common_interface::AuboException
00390 *
00391 * @par Lua function prototype
00392 * isSpeedFractionCritical() -> bool
00393 *
00394 * @par Lua example
00395 * status = isSpeedFractionCritical()
00396 *
00397 * @par JSON-RPC request example
00398 * {"jsonrpc":"2.0","method":"rob1.MotionControl.isSpeedFractionCritical","params":[],"id":1}
00399 *
00400 * @par JSON-RPC response example

```

```

00401     * {"id":1,"jsonrpc":"2.0","result":true}
00402     * \endenglish
00403     */
00404 bool isSpeedFractionCritical();
00405
00406 /**
00407  * @ingroup MotionControl
00408  * \chinese
00409  *
00410  *
00411  * @return true; false
00412  *
00413  * @throws arcs::common_interface::AuboException
00414  *
00415  * @par Lua
00416  * isBlending() -> bool
00417  *
00418  * @par Lua
00419  * status = isBlending()
00420  *
00421  * @par JSON-RPC
00422  * {"jsonrpc":"2.0","method":"rob1.MotionControl.isBlending","params":[],"id":1}
00423  *
00424  * @par JSON-RPC
00425  * {"id":1,"jsonrpc":"2.0","result":false}
00426  * \endchinese
00427  * \english
00428  * Whether it is in the blending area
00429  *
00430  * @return Returns true if in the blending area; otherwise returns false
00431  *
00432  * @throws arcs::common_interface::AuboException
00433  *
00434  * @par Lua function prototype
00435  * isBlending() -> bool
00436  *
00437  * @par Lua example
00438  * status = isBlending()
00439  *
00440  * @par JSON-RPC request example
00441  * {"jsonrpc":"2.0","method":"rob1.MotionControl.isBlending","params":[],"id":1}
00442  *
00443  * @par JSON-RPC response example
00444  * {"id":1,"jsonrpc":"2.0","result":false}
00445  * \endenglish
00446  */
00447 bool isBlending();
00448
00449 /**
00450  * @ingroup MotionControl
00451  * \chinese
00452  *
00453  * pathOffsetSet type=1
00454  * @param v
00455  * @param a
00456  * @return 0
00457  * AUBO_BUSY
00458  * AUBO_BAD_STATE
00459  * -AUBO_INVL_ARGUMENT
00460  * -AUBO_BAD_STATE
00461  *
00462  * @throws arcs::common_interface::AuboException
00463  *
00464  * @par Python
00465  * pathOffsetLimits(self: pyaubo_sdk.MotionControl, arg0: float, arg1:
00466  * float) -> int
00467  *
00468  * @par Lua
00469  * pathOffsetLimits(v: number, a: number) -> nil
00470  *
00471  * @par Lua
00472  * pathOffsetLimits(1.5,2.5)
00473  *
00474  * \endchinese
00475  * \english
00476  * Set the maximum speed and maximum acceleration for offset.
00477  * Only valid when type=1 in pathOffsetSet.
00478  * @param v Maximum speed
00479  * @param a Maximum acceleration
00480  * @return Returns 0 on success; otherwise returns an error code:
00481  * AUBO_BUSY
00482  * AUBO_BAD_STATE
00483  * -AUBO_INVL_ARGUMENT
00484  * -AUBO_BAD_STATE
00485  *
00486  * @throws arcs::common_interface::AuboException
00487  *

```

```

00488 * @par Python function prototype
00489 * pathOffsetLimits(self: pyaubo_sdk.MotionControl, arg0: float, arg1:
00490 * float) -> int
00491 *
00492 * @par Lua function prototype
00493 * pathOffsetLimits(v: number, a: number) -> nil
00494 *
00495 * @par Lua example
00496 * pathOffsetLimits(1.5,2.5)
00497 *
00498 * \endenglish
00499 */
00500 int pathOffsetLimits(double v, double a);
00501
00502 /**
00503 * @ingroup MotionControl
00504 * \chinese
00505 *
00506 * pathOffsetSet type=1
00507 * @param ref_coord 0- 1-TCP
00508 *
00509 * @throws arcs::common_interface::AuboException
00510 *
00511 * @par Python
00512 * pathOffsetCoordinate(self: pyaubo_sdk.MotionControl, arg0: int) -> float
00513 *
00514 * @par Lua
00515 * pathOffsetCoordinate(ref_coord: number) -> number
00516 *
00517 * @par Lua
00518 * num = pathOffsetCoordinate(0)
00519 *
00520 * @par JSON-RPC
00521 * {"jsonrpc":"2.0","method":"rob1.MotionControl.pathOffsetCoordinate","params":[0],"id":1}
00522 *
00523 * @par JSON-RPC
00524 * {"id":1,"jsonrpc":"2.0","result":0}
00525 * \endchinese
00526 * \english
00527 * Set the reference coordinate system for offset.
00528 * Only valid when type=1 in pathOffsetSet.
00529 * @param ref_coord Reference coordinate system: 0-base coordinate, 1-TCP
00530 *
00531 * @throws arcs::common_interface::AuboException
00532 *
00533 * @par Python function prototype
00534 * pathOffsetCoordinate(self: pyaubo_sdk.MotionControl, arg0: int) -> float
00535 *
00536 * @par Lua function prototype
00537 * pathOffsetCoordinate(ref_coord: number) -> number
00538 *
00539 * @par Lua example
00540 * num = pathOffsetCoordinate(0)
00541 *
00542 * @par JSON-RPC request example
00543 * {"jsonrpc":"2.0","method":"rob1.MotionControl.pathOffsetCoordinate","params":[0],"id":1}
00544 *
00545 * @par JSON-RPC response example
00546 * {"id":1,"jsonrpc":"2.0","result":0}
00547 * \endenglish
00548 */
00549 int pathOffsetCoordinate(int ref_coord);
00550
00551 /**
00552 * @ingroup MotionControl
00553 * \chinese
00554 *
00555 *
00556 * @return 0;
00557 * AUBO_BUSY
00558 * AUBO_BAD_STATE
00559 * -AUBO_BAD_STATE
00560 *
00561 * @throws arcs::common_interface::AuboException
00562 *
00563 * @par Python
00564 * pathOffsetEnable(self: pyaubo_sdk.MotionControl) -> int
00565 *
00566 * @par Lua
00567 * pathOffsetEnable() -> number
00568 *
00569 * @par Lua
00570 * num = pathOffsetEnable()
00571 *
00572 * @par JSON-RPC
00573 * {"jsonrpc":"2.0","method":"rob1.MotionControl.pathOffsetEnable","params":[],"id":1}
00574 *

```

```

00575 * @par JSON-RPC
00576 * {"id":1,"jsonrpc":"2.0","result":0}
00577 * \endchinese
00578 * \english
00579 * Enable path offset
00580 *
00581 * @return Returns 0 on success; otherwise returns an error code:
00582 * AUBO_BUSY
00583 * AUBO_BAD_STATE
00584 * -AUBO_BAD_STATE
00585 *
00586 * @throws arcs::common_interface::AuboException
00587 *
00588 * @par Python function prototype
00589 * pathOffsetEnable(self: pyaubo_sdk.MotionControl) -> int
00590 *
00591 * @par Lua function prototype
00592 * pathOffsetEnable() -> number
00593 *
00594 * @par Lua example
00595 * num = pathOffsetEnable()
00596 *
00597 * @par JSON-RPC request example
00598 * {"jsonrpc":"2.0","method":"rob1.MotionControl.pathOffsetEnable","params":[],"id":1}
00599 *
00600 * @par JSON-RPC response example
00601 * {"id":1,"jsonrpc":"2.0","result":0}
00602 * \endenglish
00603 */
00604 int pathOffsetEnable();
00605
00606 /**
00607 * @ingroup MotionControl
00608 * \chinese
00609 *
00610 *
00611 * @param offset
00612 * @param type 0- 1-
00613 * @return 0;
00614 * AUBO_BAD_STATE
00615 * AUBO_BUSY
00616 * -AUBO_INVL_ARGUMENT
00617 * -AUBO_BAD_STATE
00618 *
00619 * @throws arcs::common_interface::AuboException
00620 *
00621 * @par Python
00622 * pathOffsetSet(self: pyaubo_sdk.MotionControl, arg0: List[float], arg1:
00623 * int) -> int
00624 *
00625 * @par Lua
00626 * pathOffsetSet(offset: table, type: number) -> nil
00627 *
00628 * @par Lua
00629 * pathOffsetSet({ 0, 0, 0.1, 0, 0, 0 }, 0)
00630 *
00631 * @par JSON-RPC
00632 * {"jsonrpc":"2.0","method":"rob1.MotionControl.pathOffsetSet","params":[[0,0,0.01,0,0,0],0],"id":1}
00633 *
00634 * @par JSON-RPC
00635 * {"id":1,"jsonrpc":"2.0","result":0}
00636 * \endchinese
00637 * \english
00638 * Set path offset
00639 *
00640 * @param offset Pose offset in each direction
00641 * @param type Motion type 0-position planning 1-velocity planning
00642 * @return Returns 0 on success; otherwise returns error code
00643 * AUBO_BAD_STATE
00644 * AUBO_BUSY
00645 * -AUBO_INVL_ARGUMENT
00646 * -AUBO_BAD_STATE
00647 *
00648 * @throws arcs::common_interface::AuboException
00649 *
00650 * @par Python function prototype
00651 * pathOffsetSet(self: pyaubo_sdk.MotionControl, arg0: List[float], arg1:
00652 * int) -> int
00653 *
00654 * @par Lua function prototype
00655 * pathOffsetSet(offset: table, type: number) -> nil
00656 *
00657 * @par Lua example
00658 * pathOffsetSet({ 0, 0, 0.1, 0, 0, 0 }, 0)
00659 *
00660 * @par JSON-RPC request example
00661 * {"jsonrpc":"2.0","method":"rob1.MotionControl.pathOffsetSet","params":[[0,0,0.01,0,0,0],0],"id":1}

```

```

00662  *
00663  * @par JSON-RPC response example
00664  * {"id":1,"jsonrpc":"2.0","result":0}
00665  * \endenglish
00666  */
00667  int pathOffsetSet(const std::vector<double> &offset, int type = 0);
00668
00669  /**
00670  * @ingroup MotionControl
00671  * \chinese
00672  *
00673  *
00674  * @return 0;
00675  * AUBO_BAD_STATE
00676  * AUBO_BUSY
00677  * -AUBO_BAD_STATE
00678  *
00679  * @throws arcs::common_interface::AuboException
00680  *
00681  * @par Python
00682  * pathOffsetDisable(self: pyaubo_sdk.MotionControl) -> int
00683  *
00684  * @par Lua
00685  * pathOffsetDisable() -> nil
00686  *
00687  * @par Lua
00688  * pathOffsetDisable()
00689  *
00690  * @par JSON-RPC
00691  * {"jsonrpc":"2.0","method":"rob1.MotionControl.pathOffsetDisable","params":[],"id":1}
00692  *
00693  * @par JSON-RPC
00694  * {"id":1,"jsonrpc":"2.0","result":0}
00695  * \endchinese
00696  * \english
00697  * Disable path offset
00698  *
00699  * @return Returns 0 on success; otherwise returns an error code:
00700  * AUBO_BAD_STATE
00701  * AUBO_BUSY
00702  * -AUBO_BAD_STATE
00703  *
00704  * @throws arcs::common_interface::AuboException
00705  *
00706  * @par Python function prototype
00707  * pathOffsetDisable(self: pyaubo_sdk.MotionControl) -> int
00708  *
00709  * @par Lua function prototype
00710  * pathOffsetDisable() -> nil
00711  *
00712  * @par Lua example
00713  * pathOffsetDisable()
00714  *
00715  * @par JSON-RPC request example
00716  * {"jsonrpc":"2.0","method":"rob1.MotionControl.pathOffsetDisable","params":[],"id":1}
00717  *
00718  * @par JSON-RPC response example
00719  * {"id":1,"jsonrpc":"2.0","result":0}
00720  * \endenglish
00721  */
00722  int pathOffsetDisable();
00723
00724  /**
00725  * @ingroup MotionControl
00726  * \chinese
00727  * @brief
00728  * @param min:
00729  * @param max:
00730  * @param strategy:
00731  * 0-
00732  * 1-
00733  * 2-
00734  * @return
00735  *
00736  * @throws arcs::common_interface::AuboException
00737  *
00738  * @par Python
00739  * pathOffsetSupv(self: pyaubo_sdk.MotionControl, arg0: List[float], arg1:
00740  * List[float], arg2: int) -> int
00741  *
00742  * @par Lua
00743  * pathOffsetSupv(min: table, max: table, strategy: number) -> number
00744  *
00745  * @par Lua
00746  * num = pathOffsetSupv({0,0,-0.2,0,0,0},{0,0,0.5,0,0,0},0)
00747  *
00748  * \endchinese

```

```

00749 * \english
00750 * @brief Monitor trajectory offset range
00751 * @param min: Maximum offset in the negative direction of the coordinate
00752 * axis
00753 * @param max: Maximum offset in the positive direction of the coordinate
00754 * axis
00755 * @param strategy: Monitoring strategy after reaching the maximum offset
00756 * 0 - Disable monitoring
00757 * 1 - Saturation limit, i.e., maintain maximum pose
00758 * 2 - Protective stop
00759 * @return
00760 *
00761 * @throws arcs::common_interface::AuboException
00762 *
00763 * @par Python function prototype
00764 * pathOffsetSupv(self: pyaubo_sdk.MotionControl, arg0: List[float], arg1:
00765 * List[float], arg2: int) -> int
00766 *
00767 * @par Lua function prototype
00768 * pathOffsetSupv(min: table, max: table, strategy: number) -> number
00769 *
00770 * @par Lua example
00771 * num = pathOffsetSupv({0,0,-0.2,0,0,0},{0,0,0.5,0,0,0},0)
00772 *
00773 * \endenglish
00774 */
00775 int pathOffsetSupv(const std::vector<double> &min,
00776                   const std::vector<double> &max, int strategy);
00777
00778 /**
00779 * @ingroup MotionControl
00780 * \chinese
00781 *
00782 *
00783 * @return 0;
00784 * AUBO_BAD_STATE
00785 * AUBO_BUSY
00786 * -AUBO_BAD_STATE
00787 *
00788 * @throws arcs::common_interface::AuboException
00789 *
00790 * @par Python
00791 * jointOffsetEnable(self: pyaubo_sdk.MotionControl) -> int
00792 *
00793 * @par Lua
00794 * jointOffsetEnable() -> nil
00795 *
00796 * @par Lua
00797 * jointOffsetEnable()
00798 *
00799 * @par JSON-RPC
00800 * {"jsonrpc":"2.0","method":"rob1.MotionControl.jointOffsetEnable","params":[],"id":1}
00801 *
00802 * @par JSON-RPC
00803 * {"id":1,"jsonrpc":"2.0","result":0}
00804 * \endchinese
00805 * \english
00806 * Enable joint offset
00807 *
00808 * @return Returns 0 on success; otherwise returns an error code:
00809 * AUBO_BAD_STATE
00810 * AUBO_BUSY
00811 * -AUBO_BAD_STATE
00812 *
00813 * @throws arcs::common_interface::AuboException
00814 *
00815 * @par Python function prototype
00816 * jointOffsetEnable(self: pyaubo_sdk.MotionControl) -> int
00817 *
00818 * @par Lua function prototype
00819 * jointOffsetEnable() -> nil
00820 *
00821 * @par Lua example
00822 * jointOffsetEnable()
00823 *
00824 * @par JSON-RPC request example
00825 * {"jsonrpc":"2.0","method":"rob1.MotionControl.jointOffsetEnable","params":[],"id":1}
00826 *
00827 * @par JSON-RPC response example
00828 * {"id":1,"jsonrpc":"2.0","result":0}
00829 * \endenglish
00830 */
00831 int jointOffsetEnable();
00832
00833 /**
00834 * @ingroup MotionControl
00835 * \chinese

```



```

00836 *
00837 *
00838 * @param offset
00839 * @param type
00840 * @return 0;
00841 * AUBO_BAD_STATE
00842 * AUBO_BUSY
00843 * -AUBO_INVL_ARGUMENT
00844 * -AUBO_BAD_STATE
00845 *
00846 * @throws arcs::common_interface::AuboException
00847 *
00848 * @par Python
00849 * jointOffsetSet(self: pyaubo_sdk.MotionControl, arg0: List[float], arg1:
00850 * int) -> int
00851 *
00852 * @par Lua
00853 * jointOffsetSet(offset: table, type: number) -> nil
00854 *
00855 * @par Lua
00856 * jointOffsetSet({0.1,0,0,0,0},1)
00857 *
00858 * @par JSON-RPC
00859 * {"jsonrpc":"2.0","method":"rob1.MotionControl.jointOffsetSet","params":[[0.1,0,0,0,0],1],"id":1}
00860 *
00861 * @par JSON-RPC
00862 * {"id":1,"jsonrpc":"2.0","result":0}
00863 * \endchinese
00864 * \english
00865 * Set joint offset
00866 *
00867 * @param offset Pose offset for each joint
00868 * @param type
00869 * @return Returns 0 on success; otherwise returns error code
00870 * AUBO_BAD_STATE
00871 * AUBO_BUSY
00872 * -AUBO_INVL_ARGUMENT
00873 * -AUBO_BAD_STATE
00874 *
00875 * @throws arcs::common_interface::AuboException
00876 *
00877 * @par Python function prototype
00878 * jointOffsetSet(self: pyaubo_sdk.MotionControl, arg0: List[float], arg1:
00879 * int) -> int
00880 *
00881 * @par Lua function prototype
00882 * jointOffsetSet(offset: table, type: number) -> nil
00883 *
00884 * @par Lua example
00885 * jointOffsetSet({0.1,0,0,0,0},1)
00886 *
00887 * @par JSON-RPC request example
00888 * {"jsonrpc":"2.0","method":"rob1.MotionControl.jointOffsetSet","params":[[0.1,0,0,0,0],1],"id":1}
00889 *
00890 * @par JSON-RPC response example
00891 * {"id":1,"jsonrpc":"2.0","result":0}
00892 * \endenglish
00893 */
00894 int jointOffsetSet(const std::vector<double> &offset, int type = 1);
00895
00896 /**
00897 * @ingroup MotionControl
00898 * \chinese
00899 *
00900 *
00901 * @return 0;
00902 * AUBO_BAD_STATE
00903 * AUBO_BUSY
00904 * -AUBO_BAD_STATE
00905 *
00906 * @throws arcs::common_interface::AuboException
00907 *
00908 * @par Python
00909 * jointOffsetDisable(self: pyaubo_sdk.MotionControl) -> int
00910 *
00911 * @par Lua
00912 * jointOffsetDisable() -> nil
00913 *
00914 * @par Lua
00915 * jointOffsetDisable()
00916 *
00917 * @par JSON-RPC
00918 * {"jsonrpc":"2.0","method":"rob1.MotionControl.jointOffsetDisable","params":[],"id":1}
00919 *
00920 * @par JSON-RPC
00921 * {"id":1,"jsonrpc":"2.0","result":0}
00922 * \endchinese

```

```

00923 * \english
00924 * Disable joint offset
00925 *
00926 * @return Returns 0 on success; otherwise returns an error code:
00927 * AUBO_BAD_STATE
00928 * AUBO_BUSY
00929 * -AUBO_BAD_STATE
00930 *
00931 * @throws arcs::common_interface::AuboException
00932 *
00933 * @par Python function prototype
00934 * jointOffsetDisable(self: pyaubo_sdk.MotionControl) -> int
00935 *
00936 * @par Lua function prototype
00937 * jointOffsetDisable() -> nil
00938 *
00939 * @par Lua example
00940 * jointOffsetDisable()
00941 *
00942 * @par JSON-RPC request example
00943 * {"jsonrpc":"2.0","method":"rob1.MotionControl.jointOffsetDisable","params":[],"id":1}
00944 *
00945 * @par JSON-RPC response example
00946 * {"id":1,"jsonrpc":"2.0","result":0}
00947 * \endenglish
00948 */
00949 int jointOffsetDisable();
00950
00951 /**
00952 * @ingroup MotionControl
00953 * \chinese
00954 * (INST)
00955 * moveJoint/moveLine/moveCircle      setPayload
00956 *
00957 * _INST , moveJoint
00958 * _INST(MotionControl, 5, moveJoint, q, a, v, blend_radius, duration)
00959 *
00960 * @return
00961 *
00962 * @throws arcs::common_interface::AuboException
00963 *
00964 * @par Python
00965 * getQueueSize(self: pyaubo_sdk.MotionControl) -> int
00966 *
00967 * @par Lua
00968 * getQueueSize() -> number
00969 *
00970 * @par Lua
00971 * num = getQueueSize()
00972 *
00973 * @par JSON-RPC
00974 * {"jsonrpc":"2.0","method":"rob1.MotionControl.getQueueSize","params":[],"id":1}
00975 *
00976 * @par JSON-RPC
00977 * {"id":1,"jsonrpc":"2.0","result":0}
00978 *
00979 * \endchinese
00980 * \english
00981 * Get the number of enqueued instruction segments (INST), including motion
00982 * instructions such as moveJoint/moveLine/moveCircle and configuration
00983 * instructions such as setPayload.
00984 *
00985 * Instructions are generally indicated with _INST in macro definitions, for
00986 * example: _INST(MotionControl, 5, moveJoint, q, a, v, blend_radius,
00987 * duration)
00988 *
00989 * @return The number of enqueued instruction segments.
00990 *
00991 * @throws arcs::common_interface::AuboException
00992 *
00993 * @par Python function prototype
00994 * getQueueSize(self: pyaubo_sdk.MotionControl) -> int
00995 *
00996 * @par Lua function prototype
00997 * getQueueSize() -> number
00998 *
00999 * @par Lua example
01000 * num = getQueueSize()
01001 *
01002 * @par JSON-RPC request example
01003 * {"jsonrpc":"2.0","method":"rob1.MotionControl.getQueueSize","params":[],"id":1}
01004 *
01005 * @par JSON-RPC response example
01006 * {"id":1,"jsonrpc":"2.0","result":0}
01007 *
01008 * \endenglish
01009 */

```

```

01010 int getQueueSize();
01011
01012 /**
01013  * @ingroup MotionControl
01014  * \chinese
01015  *
01016  *
01017  * @return
01018  *
01019  * @throws arcs::common_interface::AuboException
01020  *
01021  * @par Python
01022  * getTrajectoryQueueSize(self: pyaubo_sdk.MotionControl) -> int
01023  *
01024  * @par Lua
01025  * getTrajectoryQueueSize() -> number
01026  *
01027  * @par Lua
01028  * num = getTrajectoryQueueSize()
01029  *
01030  * @par JSON-RPC
01031  * {"jsonrpc":"2.0","method":"rob1.MotionControl.getTrajectoryQueueSize","params":[],"id":1}
01032  *
01033  * @par JSON-RPC
01034  * {"id":1,"jsonrpc":"2.0","result":0}
01035  * \endchinese
01036  * \english
01037  * Get the number of enqueued motion planning interpolation points
01038  *
01039  * @return The number of enqueued motion planning interpolation points
01040  *
01041  * @throws arcs::common_interface::AuboException
01042  *
01043  * @par Python function prototype
01044  * getTrajectoryQueueSize(self: pyaubo_sdk.MotionControl) -> int
01045  *
01046  * @par Lua function prototype
01047  * getTrajectoryQueueSize() -> number
01048  *
01049  * @par Lua example
01050  * num = getTrajectoryQueueSize()
01051  *
01052  * @par JSON-RPC request example
01053  * {"jsonrpc":"2.0","method":"rob1.MotionControl.getTrajectoryQueueSize","params":[],"id":1}
01054  *
01055  * @par JSON-RPC response example
01056  * {"id":1,"jsonrpc":"2.0","result":0}
01057  * \endenglish
01058  */
01059 int getTrajectoryQueueSize();
01060
01061 /**
01062  * @ingroup MotionControl
01063  * \chinese
01064  * ID
01065  *
01066  * @return ID
01067  * @retval -1 \n
01068  * movePathBuffer buffer (moveJoint moveLine )
01069  *
01070  * @throws arcs::common_interface::AuboException
01071  *
01072  * @par Python
01073  * getExecId(self: pyaubo_sdk.MotionControl) -> int
01074  *
01075  * @par Lua
01076  * getExecId() -> number
01077  *
01078  * @par Lua
01079  * num = getExecId()
01080  *
01081  * @par JSON-RPC
01082  * {"jsonrpc":"2.0","method":"rob1.MotionControl.getExecId","params":[],"id":1}
01083  *
01084  * @par JSON-RPC
01085  * {"id":1,"jsonrpc":"2.0","result":-1}
01086  *
01087  * \endchinese
01088  * \english
01089  * Get the ID of the currently interpolating motion instruction segment.
01090  *
01091  * @return The ID of the currently interpolating motion instruction segment.
01092  * @retval -1 Indicates the trajectory queue is empty. \n
01093  * Both the buffer in movePathBuffer motion and the queue in the planner
01094  * (such as moveJoint and moveLine) belong to the trajectory queue.
01095  *
01096  * @throws arcs::common_interface::AuboException

```

```

01097 *
01098 * @par Python function prototype
01099 * getExecId(self: pyaubo_sdk.MotionControl) -> int
01100 *
01101 * @par Lua function prototype
01102 * getExecId() -> number
01103 *
01104 * @par Lua example
01105 * num = getExecId()
01106 *
01107 * @par JSON-RPC request example
01108 * {"jsonrpc":"2.0","method":"rob1.MotionControl.getExecId","params":[],"id":1}
01109 *
01110 * @par JSON-RPC response example
01111 * {"id":1,"jsonrpc":"2.0","result":-1}
01112 *
01113 * \endenglish
01114 */
01115 int getExecId();
01116
01117 /**
01118 * @ingroup MotionControl
01119 * \chinese
01120 * ID
01121 *
01122 * @param id ID
01123 * @return
01124 *
01125 * @throws arcs::common_interface::AuboException
01126 *
01127 * @par Python
01128 * getDuration(self: pyaubo_sdk.MotionControl, arg0: int) -> float
01129 *
01130 * @par Lua
01131 * getDuration(id: number) -> number
01132 *
01133 * @par Lua
01134 * num = getDuration(16)
01135 *
01136 * @par JSON-RPC
01137 * {"jsonrpc":"2.0","method":"rob1.MotionControl.getDuration","params":[16],"id":1}
01138 *
01139 * @par JSON-RPC
01140 * {"id":1,"jsonrpc":"2.0","result":0.0}
01141 *
01142 * \endchinese
01143 * \english
01144 * Get the expected execution duration of the motion segment with the
01145 * specified ID.
01146 *
01147 * @param id Motion segment ID
01148 * @return Returns the expected execution duration
01149 *
01150 * @throws arcs::common_interface::AuboException
01151 *
01152 * @par Python function prototype
01153 * getDuration(self: pyaubo_sdk.MotionControl, arg0: int) -> float
01154 *
01155 * @par Lua function prototype
01156 * getDuration(id: number) -> number
01157 *
01158 * @par Lua example
01159 * num = getDuration(16)
01160 *
01161 * @par JSON-RPC request example
01162 * {"jsonrpc":"2.0","method":"rob1.MotionControl.getDuration","params":[16],"id":1}
01163 *
01164 * @par JSON-RPC response example
01165 * {"id":1,"jsonrpc":"2.0","result":0.0}
01166 *
01167 * \endenglish
01168 */
01169 double getDuration(int id);
01170
01171 /**
01172 * @ingroup MotionControl
01173 * \chinese
01174 * ID
01175 *
01176 * @param id ID
01177 * @return
01178 *
01179 * @throws arcs::common_interface::AuboException
01180 *
01181 * @par Python
01182 * getMotionLeftTime(self: pyaubo_sdk.MotionControl, arg0: int) -> float
01183 *

```

```

01184 * @par Lua
01185 * getMotionLeftTime(id: number) -> number
01186 *
01187 * @par Lua
01188 * num = getMotionLeftTime(16)
01189 *
01190 * @par JSON-RPC
01191 * {"jsonrpc":"2.0","method":"rob1.MotionControl.getMotionLeftTime","params":[16],"id":1}
01192 *
01193 * @par JSON-RPC
01194 * {"id":1,"jsonrpc":"2.0","result":0.0}
01195 * \endchinese
01196 * \english
01197 * Get the remaining execution time of the motion segment with the specified
01198 * ID.
01199 *
01200 * @param id Motion segment ID
01201 * @return Returns the remaining execution time
01202 *
01203 * @throws arcs::common_interface::AuboException
01204 *
01205 * @par Python function prototype
01206 * getMotionLeftTime(self: pyaubo_sdk.MotionControl, arg0: int) -> float
01207 *
01208 * @par Lua function prototype
01209 * getMotionLeftTime(id: number) -> number
01210 *
01211 * @par Lua example
01212 * num = getMotionLeftTime(16)
01213 *
01214 * @par JSON-RPC request example
01215 * {"jsonrpc":"2.0","method":"rob1.MotionControl.getMotionLeftTime","params":[16],"id":1}
01216 *
01217 * @par JSON-RPC response example
01218 * {"id":1,"jsonrpc":"2.0","result":0.0}
01219 * \endenglish
01220 */
01221 double getMotionLeftTime(int id);
01222
01223 /**
01224 * @ingroup MotionControl
01225 * \chinese
01226 * StopMove
01227 * StartMove
01228 *
01229 *
01230 *
01231 * @param quick true:          quick
01232 *
01233 *
01234 * @return 0;
01235 * AUBO_BAD_STATE
01236 * AUBO_BUSY
01237 * -AUBO_BAD_STATE
01238 *
01239 * @throws arcs::common_interface::AuboException
01240 *
01241 * @par Lua
01242 * StopMove(quick: bool,all_tasks: bool) -> number
01243 *
01244 * @par Lua
01245 * num = StopMove(true,true)
01246 *
01247 * \endchinese
01248 * \english
01249 * StopMove is used to stop robot and external axes movements and any
01250 * belonging process temporarily. If the instruction StartMove is given then
01251 * the movement and process resumes.
01252 *
01253 * This instruction can, for example, be used in a trap routine to stop the
01254 * robot temporarily when an interrupt occurs.
01255 *
01256 * @param quick true: Stops the robot on the path as fast as possible.
01257 * Without the optional parameter \Quick, the robot stops on the path, but
01258 * the braking distance is longer (same as for normal Program Stop).
01259 *
01260 * @return Returns 0 on success; otherwise returns an error code:
01261 * AUBO_BAD_STATE
01262 * AUBO_BUSY
01263 * -AUBO_BAD_STATE
01264 *
01265 * @throws arcs::common_interface::AuboException
01266 *
01267 * @par Lua function prototype
01268 * StopMove(quick: bool,all_tasks: bool) -> number
01269 *
01270 * @par Lua example

```

```

01271     * num = StopMove(true,true)
01272     *
01273     * \endenglish
01274     */
01275     int stopMove(bool quick, bool all_tasks);
01276
01277     /**
01278     * @ingroup MotionControl
01279     * \chinese
01280     * StartMove
01281     *   • StopMove
01282     *   • StorePath ... RestoPath
01283     *   • ERR_PATH_STOP      ERROR
01284     *
01285     *
01286     * @return 0;
01287     * AUBO_BAD_STATE
01288     * AUBO_BUSY
01289     * -AUBO_BAD_STATE
01290     *
01291     * @throws arcs::common_interface::AuboException
01292     *
01293     * @par Lua
01294     * startMove() -> number
01295     *
01296     * @par Lua
01297     * num = startMove()
01298     *
01299     * @par JSON-RPC
01300     * {"jsonrpc":"2.0","method":"rob1.MotionControl.startMove","params":[],"id":1}
01301     *
01302     * @par JSON-RPC
01303     * {"id":1,"jsonrpc":"2.0","result":0}
01304     * \endchinese
01305     * \english
01306     * StartMove is used to resume robot, external axes movement and belonging
01307     * process after the movement has been stopped
01308     *
01309     *   • by the instruction StopMove.
01310     *   • after execution of StorePath ... RestoPath sequence.
01311     *   • after asynchronously raised movements errors, such as ERR_PATH_STOP or
01312     *     specific process error after handling in the ERROR handler.
01313     *
01314     * @return Returns 0 on success; otherwise returns an error code:
01315     * AUBO_BAD_STATE
01316     * AUBO_BUSY
01317     * -AUBO_BAD_STATE
01318     *
01319     * @throws arcs::common_interface::AuboException
01320     *
01321     * @par Lua function prototype
01322     * startMove() -> number
01323     *
01324     * @par Lua example
01325     * num = startMove()
01326     *
01327     * @par JSON-RPC request example
01328     * {"jsonrpc":"2.0","method":"rob1.MotionControl.startMove","params":[],"id":1}
01329     *
01330     * @par JSON-RPC response example
01331     * {"id":1,"jsonrpc":"2.0","result":0}
01332     * \endenglish
01333     */
01334     int startMove();
01335
01336     /**
01337     * @ingroup MotionControl
01338     * \chinese
01339     * storePath
01340     *
01341     * @param keep_sync
01342     * @return 0;
01343     * AUBO_BAD_STATE
01344     * AUBO_BUSY
01345     * -AUBO_BAD_STATE
01346     *
01347     * @throws arcs::common_interface::AuboException
01348     *
01349     * @par Lua
01350     * storePath(keep_sync: bool) -> number
01351     *
01352     * @par Lua
01353     * num = storePath()
01354     *
01355     * \endchinese
01356     * \english
01357     * storePath

```

```

01358 *
01359 * @param keep_sync
01360 * @return Returns 0 on success; otherwise returns an error code:
01361 * AUBO_BAD_STATE
01362 * AUBO_BUSY
01363 * -AUBO_BAD_STATE
01364 *
01365 * @throws arcs::common_interface::AuboException
01366 *
01367 * @par Lua function prototype
01368 * storePath(keep_sync: bool) -> number
01369 *
01370 * @par Lua example
01371 * num = storePath()
01372 *
01373 * \endenglish
01374 */
01375 int storePath(bool keep_sync);
01376
01377 /**
01378 * @ingroup MotionControl
01379 * \chinese
01380 * ClearPath ( )          StorePath
01381 *
01382 *
01383 * @return 0;
01384 * AUBO_BAD_STATE
01385 * AUBO_BUSY
01386 * -AUBO_BAD_STATE
01387 *
01388 * @throws arcs::common_interface::AuboException
01389 *
01390 * @par Lua
01391 * clearPath() -> number
01392 *
01393 * @par Lua
01394 * num = clearPath()
01395 *
01396 * @par JSON-RPC
01397 * {"jsonrpc":"2.0","method":"rob1.MotionControl.clearPath","params":[],"id":1}
01398 *
01399 * @par JSON-RPC
01400 * {"id":1,"jsonrpc":"2.0","result":0}
01401 *
01402 * \endchinese
01403 * \english
01404 * ClearPath clears the whole motion path on the current motion path level
01405 * (base level or StorePath level).
01406 *
01407 * @return Returns 0 on success; otherwise returns an error code:
01408 * AUBO_BAD_STATE
01409 * AUBO_BUSY
01410 * -AUBO_BAD_STATE
01411 *
01412 * @throws arcs::common_interface::AuboException
01413 *
01414 * @par Lua function prototype
01415 * clearPath() -> number
01416 *
01417 * @par Lua example
01418 * num = clearPath()
01419 *
01420 * @par JSON-RPC request example
01421 * {"jsonrpc":"2.0","method":"rob1.MotionControl.clearPath","params":[],"id":1}
01422 *
01423 * @par JSON-RPC response example
01424 * {"id":1,"jsonrpc":"2.0","result":0}
01425 *
01426 * \endenglish
01427 */
01428 int clearPath();
01429
01430 /**
01431 * @ingroup MotionControl
01432 * \chinese
01433 * restoPath
01434 *
01435 * @return 0;
01436 * AUBO_BAD_STATE
01437 * AUBO_BUSY
01438 * -AUBO_BAD_STATE
01439 *
01440 * @throws arcs::common_interface::AuboException
01441 *
01442 * @par Lua
01443 * restoPath() -> number
01444 *

```

```

01445 * @par Lua
01446 * num = restoPath()
01447 *
01448 * @par JSON-RPC
01449 * {"jsonrpc":"2.0","method":"rob1.MotionControl.restoPath","params":[],"id":1}
01450 *
01451 * @par JSON-RPC
01452 * {"id":1,"jsonrpc":"2.0","result":0}
01453 * \endchinese
01454 * \english
01455 * restoPath
01456 *
01457 * @return Returns 0 on success; otherwise returns an error code:
01458 * AUBO_BAD_STATE
01459 * AUBO_BUSY
01460 * -AUBO_BAD_STATE
01461 *
01462 * @throws arcs::common_interface::AuboException
01463 *
01464 * @par Lua function prototype
01465 * restoPath() -> number
01466 *
01467 * @par Lua example
01468 * num restoPath()
01469 *
01470 * @par JSON-RPC request example
01471 * {"jsonrpc":"2.0","method":"rob1.MotionControl.restoPath","params":[],"id":1}
01472 *
01473 * @par JSON-RPC response example
01474 * {"id":1,"jsonrpc":"2.0","result":0}
01475 * \endenglish
01476 */
01477 int restoPath();
01478
01479 /**
01480 * @ingroup MotionControl
01481 * \chinese
01482 *
01483 *
01484 * @return
01485 *
01486 * @throws arcs::common_interface::AuboException
01487 *
01488 * @par Python
01489 * getProgress(self: pyaubo_sdk.MotionControl) -> float
01490 *
01491 * @par Lua
01492 * getProgress() -> number
01493 *
01494 * @par Lua
01495 * num = getProgress()
01496 *
01497 * @par JSON-RPC
01498 * {"jsonrpc":"2.0","method":"rob1.MotionControl.getProgress","params":[],"id":1}
01499 *
01500 * @par JSON-RPC
01501 * {"id":1,"jsonrpc":"2.0","result":0.0}
01502 *
01503 * \endchinese
01504 * \english
01505 * Get the execution progress of the current motion instruction segment.
01506 *
01507 * @return Returns the execution progress.
01508 *
01509 * @throws arcs::common_interface::AuboException
01510 *
01511 * @par Python function prototype
01512 * getProgress(self: pyaubo_sdk.MotionControl) -> float
01513 *
01514 * @par Lua function prototype
01515 * getProgress() -> number
01516 *
01517 * @par Lua example
01518 * num = getProgress()
01519 *
01520 * @par JSON-RPC request example
01521 * {"jsonrpc":"2.0","method":"rob1.MotionControl.getProgress","params":[],"id":1}
01522 *
01523 * @par JSON-RPC response example
01524 * {"id":1,"jsonrpc":"2.0","result":0.0}
01525 *
01526 * \endenglish
01527 */
01528 double getProgress();
01529
01530 /**
01531 * @ingroup MotionControl

```



```

01532 * \chinese
01533 *
01534 *
01535 * @note
01536 *
01537 * @param module_name
01538 * @param mounting_pose
01539 * TCP
01540 * @return 0;
01541 * AUBO_BAD_STATE
01542 * AUBO_BUSY
01543 * -AUBO_INVL_ARGUMENT
01544 * -AUBO_BAD_STATE
01545 *
01546 * @throws arcs::common_interface::AuboException
01547 *
01548 * @par Python
01549 * setWorkObjectHold(self: pyaubo_sdk.MotionControl, arg0: str, arg1:
01550 * List[float]) -> int
01551 *
01552 * @par Lua
01553 * setWorkObjectHold(module_name: string, mounting_pose: table) -> nil
01554 *
01555 * @par Lua
01556 * setWorkObjectHold("object",{0.2,0.1,-0.4,3.14,0,-1.57})
01557 *
01558 * \endchinese
01559 * \english
01560 * Specify the name and mounting position when the workpiece is installed on
01561 * the end of another robot or external axis.
01562 *
01563 * @note Not implemented yet
01564 *
01565 * @param module_name Name of the control module
01566 * @param mounting_pose Relative position of the grasp,
01567 * if it is a robot, it is relative to the robot's end center point (not the
01568 * TCP point)
01569 * @return Returns 0 on success; otherwise returns an error code:
01570 * AUBO_BAD_STATE
01571 * AUBO_BUSY
01572 * -AUBO_INVL_ARGUMENT
01573 * -AUBO_BAD_STATE
01574 *
01575 * @throws arcs::common_interface::AuboException
01576 *
01577 * @par Python function prototype
01578 * setWorkObjectHold(self: pyaubo_sdk.MotionControl, arg0: str, arg1:
01579 * List[float]) -> int
01580 *
01581 * @par Lua function prototype
01582 * setWorkObjectHold(module_name: string, mounting_pose: table) -> nil
01583 *
01584 * @par Lua example
01585 * setWorkObjectHold("object",{0.2,0.1,-0.4,3.14,0,-1.57})
01586 *
01587 * \endenglish
01588 */
01589 int setWorkObjectHold(const std::string &module_name,
01590                      const std::vector<double> &mounting_pose);
01591
01592 /**
01593 * @ingroup MotionControl
01594 * \chinese
01595 *
01596 *
01597 * @note
01598 *
01599 * @return
01600 *
01601 * @throws arcs::common_interface::AuboException
01602 *
01603 * @par Python
01604 * getWorkObjectHold(self: pyaubo_sdk.MotionControl) -> Tuple[str,
01605 * List[float]]
01606 *
01607 * @par Lua
01608 * getWorkObjectHold() -> table
01609 *
01610 * @par Lua
01611 * Object_table = getWorkObjectHold()
01612 *
01613 * @par JSON-RPC
01614 * {"jsonrpc":"2.0","method":"rob1.MotionControl.getWorkObjectHold","params":[],"id":1}
01615 *
01616 * @par JSON-RPC
01617 * {"id":1,"jsonrpc":"2.0","result":["",""]}
01618 * \endchinese

```

```

01619 * \english
01620 * getWorkObjectHold
01621 *
01622 * @note Not implemented yet
01623 *
01624 * @return Returns a tuple containing the control module name and mounting
01625 * pose
01626 *
01627 * @throws arcs::common_interface::AuboException
01628 *
01629 * @par Python function prototype
01630 * getWorkObjectHold(self: pyaubo_sdk.MotionControl) -> Tuple[str,
01631 * List[float]]
01632 *
01633 * @par Lua function prototype
01634 * getWorkObjectHold() -> table
01635 *
01636 * @par Lua example
01637 * Object_table = getWorkObjectHold()
01638 *
01639 * @par JSON-RPC request example
01640 * {"jsonrpc": "2.0", "method": "rob1.MotionControl.getWorkObjectHold", "params": [], "id": 1}
01641 *
01642 * @par JSON-RPC response example
01643 * {"id": 1, "jsonrpc": "2.0", "result": ["", []]}
01644 * \endenglish
01645 */
01646 std::tuple<std::string, std::vector<double>» getWorkObjectHold();
01647
01648 /**
01649 * @ingroup MotionControl
01650 * \chinese
01651 * @note
01652 *
01653 * ( resumeMoveJoint resumeMoveLine , )
01654 *
01655 * @return
01656 *
01657 * @throws arcs::common_interface::AuboException
01658 *
01659 * @par Python
01660 * getPauseJointPositions(self: pyaubo_sdk.MotionControl) -> List[float]
01661 *
01662 * @par Lua
01663 * getPauseJointPositions() -> table
01664 *
01665 * @par Lua
01666 * JointPositions = getPauseJointPositions()
01667 *
01668 * @par JSON-RPC
01669 * {"jsonrpc": "2.0", "method": "rob1.MotionControl.getPauseJointPositions", "params": [], "id": 1}
01670 *
01671 * @par JSON-RPC
01672 * {"id": 1, "jsonrpc": "2.0", "result": [8.2321e-13, -0.200999, 1.33999, 0.334999, 1.206, -6.39383e-12]}
01673 * \endchinese
01674 * \english
01675 * @note Get the joint positions at the pause point.
01676 *
01677 * Commonly used during process recovery after a collision occurs (first
01678 * move to the pause position using resumeMoveJoint or resumeMoveLine, then
01679 * resume the process).
01680 *
01681 * @return Pause joint positions
01682 *
01683 * @throws arcs::common_interface::AuboException
01684 *
01685 * @par Python function prototype
01686 * getPauseJointPositions(self: pyaubo_sdk.MotionControl) -> List[float]
01687 *
01688 * @par Lua function prototype
01689 * getPauseJointPositions() -> table
01690 *
01691 * @par Lua example
01692 * JointPositions = getPauseJointPositions()
01693 *
01694 * @par JSON-RPC request example
01695 * {"jsonrpc": "2.0", "method": "rob1.MotionControl.getPauseJointPositions", "params": [], "id": 1}
01696 *
01697 * @par JSON-RPC response example
01698 * {"id": 1, "jsonrpc": "2.0", "result": [8.2321e-13, -0.200999, 1.33999, 0.334999, 1.206, -6.39383e-12]}
01699 * \endenglish
01700 */
01701 std::vector<double> getPauseJointPositions();
01702
01703 /**
01704 * @ingroup MotionControl
01705 * \chinese

```

```

01706  *
01707  *
01708  * @param q
01709  * @param move_type 0: 1
01710  * @param blend_radius
01711  * @param qdmax (6 )
01712  * @param qddmax (6 )
01713  * @param vmax , (2 )
01714  * @param amax (2 )
01715  * @return 0;
01716  * AUBO_BAD_STATE
01717  * AUBO_BUSY
01718  * -AUBO_BAD_STATE
01719  *
01720  * @throws arcs::common_interface::AuboException
01721  *
01722  * @par Python
01723  * setResumeStartPoint(self: pyaubo_sdk.MotionControl,
01724  * arg0 List[float],arg1: int arg2: float arg3: List[float], arg4:
01725  * List[float],arg5:float,arg5:float) -> int
01726  *
01727  * @par Lua
01728  * setResumeStartPoint(q : table, move_type: number,blend_radius:
01729  * number, qdmax: table, qddmax:
01730  * table,vmax:number,amax:number) -> nilr
01731  *
01732  * @par Lua
01733  * setResumeStartPoint({0,0,0,0,0,0},1,1,{0,0,0,0,0,0},{0,0,0,0,0,0},{1,1},{1,1})
01734  *
01735  * @par JSON-RPC
01736  * {"json-
rpc":"2.0","method":"rob1.MotionControl.setResumeStartPoint","params":[[0,0,0,0,0,0],1,1,[0,0,0,0,0,0],[0,0,0,0,0,0],1,1],"id":1}
01737  *
01738  * @par JSON-RPC
01739  * {"id":1,"jsonrpc":"2.0","result":0}
01740  * \endchinese
01741  * \english
01742  * Set resume motion parameters
01743  *
01744  * @param q Resume start position
01745  * @param move_type 0: Joint space, 1: Cartesian space
01746  * @param blend_radius Blend radius
01747  * @param qdmax Maximum joint speed (6 elements)
01748  * @param qddmax Maximum joint acceleration (6 elements)
01749  * @param vmax Maximum linear and angular speed for linear motion (2
01750  * elements)
01751  * @param amax Maximum linear and angular acceleration for linear motion (2
01752  * elements)
01753  * @return Returns 0 on success; otherwise returns error code
01754  * AUBO_BAD_STATE
01755  * AUBO_BUSY
01756  * -AUBO_BAD_STATE
01757  *
01758  * @throws arcs::common_interface::AuboException
01759  *
01760  * @par Python function prototype
01761  * setResumeStartPoint(self: pyaubo_sdk.MotionControl,
01762  * arg0: List[float], arg1: int, arg2: float, arg3: List[float], arg4:
01763  * List[float], arg5: float, arg6: float) -> int
01764  *
01765  * @par Lua function prototype
01766  * setResumeStartPoint(q: table, move_type: number, blend_radius:
01767  * number, qdmax: table, qddmax:
01768  * table, vmax: number, amax: number) -> nil
01769  *
01770  * @par Lua example
01771  * setResumeStartPoint({0,0,0,0,0,0},1,1,{0,0,0,0,0,0},{0,0,0,0,0,0},{1,1},{1,1})
01772  *
01773  * @par JSON-RPC request example
01774  * {"json-
rpc":"2.0","method":"rob1.MotionControl.setResumeStartPoint","params":[[0,0,0,0,0,0],1,1,[0,0,0,0,0,0],[0,0,0,0,0,0],1,1],"id":1}
01775  *
01776  * @par JSON-RPC response example
01777  * {"id":1,"jsonrpc":"2.0","result":0}
01778  * \endenglish
01779  */
01780 int setResumeStartPoint(const std::vector<double> &q, int move_type,
01781 double blend_radius,
01782 const std::vector<double> &qdmax,
01783 const std::vector<double> &qddmax,
01784 const std::vector<double> &vmax,
01785 const std::vector<double> &amax);
01786 /**
01787 * @ingroup MotionControl
01788 * \chinese
01789 *
01790 *

```

```

01791     * @return 0:      1:
01792     *
01793     * @throws arcs::common_interface::AuboException
01794     *
01795     * @par Python
01796     * getResumeMode(self: pyaubo_sdk.MotionControl) -> int
01797     *
01798     * @par Lua
01799     * getResumeMode() -> int
01800     *
01801     * @par Lua
01802     * num = getResumeMode()
01803     *
01804     * @par JSON-RPC
01805     * {"jsonrpc":"2.0","method":"rob1.MotionControl.getResumeMode","params":[],"id":1}
01806     *
01807     * @par JSON-RPC
01808     * {"id":1,"jsonrpc":"2.0","result":false}
01809     *
01810     * \endchinese
01811     * \english
01812     * Get resume motion mode
01813     *
01814     * @return 0: Resume start point is the pause point; 1: Resume start point
01815     * is the specified point
01816     *
01817     * @throws arcs::common_interface::AuboException
01818     *
01819     * @par Python function prototype
01820     * getResumeMode(self: pyaubo_sdk.MotionControl) -> int
01821     *
01822     * @par Lua function prototype
01823     * getResumeMode() -> int
01824     *
01825     * @par Lua example
01826     * num = getResumeMode()
01827     *
01828     * @par JSON-RPC request example
01829     * {"jsonrpc":"2.0","method":"rob1.MotionControl.getResumeMode","params":[],"id":1}
01830     *
01831     * @par JSON-RPC response example
01832     * {"id":1,"jsonrpc":"2.0","result":false}
01833     *
01834     * \endenglish
01835     */
01836 int getResumeMode();
01837
01838 /**
01839  * @ingroup MotionControl
01840  * \chinese
01841  *
01842  * setServoModeSelect
01843  *
01844  * @param enable
01845  * @return 0;
01846  * AUBO_BAD_STATE
01847  * AUBO_BUSY
01848  * -AUBO_BAD_STATE
01849  *
01850  * @throws arcs::common_interface::AuboException
01851  *
01852  * @par Python
01853  * setServoMode(self: pyaubo_sdk.MotionControl, arg0: bool) -> int
01854  *
01855  * @par Lua
01856  * setServoMode(enable: boolean) -> nil
01857  *
01858  * @par Lua
01859  * setServoMode(true)
01860  *
01861  * @par JSON-RPC
01862  * {"jsonrpc":"2.0","method":"rob1.MotionControl.setServoMode","params":[true],"id":1}
01863  *
01864  * @par JSON-RPC
01865  * {"id":1,"jsonrpc":"2.0","result":0}
01866  * \endchinese
01867  * \english
01868  * Set servo mode
01869  * Use setServoModeSelect instead
01870  *
01871  * @param enable Whether to enable
01872  * @return Returns 0 on success; otherwise returns an error code:
01873  * AUBO_BAD_STATE
01874  * AUBO_BUSY
01875  * -AUBO_BAD_STATE
01876  *
01877  * @throws arcs::common_interface::AuboException

```

```

01878 *
01879 * @par Python function prototype
01880 * setServoMode(self: pyaubo_sdk.MotionControl, arg0: bool) -> int
01881 *
01882 * @par Lua function prototype
01883 * setServoMode(enable: boolean) -> nil
01884 *
01885 * @par Lua example
01886 * setServoMode(true)
01887 *
01888 * @par JSON-RPC request example
01889 * {"jsonrpc":"2.0","method":"rob1.MotionControl.setServoMode","params":[true],"id":1}
01890 *
01891 * @par JSON-RPC response example
01892 * {"id":1,"jsonrpc":"2.0","result":0}
01893 * \endenglish
01894 */
01895 ARCS_DEPRECATED int setServoMode(bool enable);
01896
01897 /**
01898 * @ingroup MotionControl
01899 * \chinese
01900 *
01901 *   getServoModeSelect
01902 *
01903 * @return true false
01904 *
01905 * @throws arcs::common_interface::AuboException
01906 *
01907 * @par Python
01908 * isServoModeEnabled(self: pyaubo_sdk.MotionControl) -> bool
01909 *
01910 * @par Lua
01911 * isServoModeEnabled() -> boolean
01912 *
01913 * @par Lua
01914 * Servo_status = isServoModeEnabled()
01915 *
01916 * @par JSON-RPC
01917 * {"jsonrpc":"2.0","method":"rob1.MotionControl.isServoModeEnabled","params":[],"id":1}
01918 *
01919 * @par JSON-RPC
01920 * {"id":1,"jsonrpc":"2.0","result":false}
01921 *
01922 * \endchinese
01923 * \english
01924 * Determine whether the servo mode is enabled.
01925 * Use getServoModeSelect instead.
01926 *
01927 * @return Returns true if enabled, otherwise returns false.
01928 *
01929 * @throws arcs::common_interface::AuboException
01930 *
01931 * @par Python function prototype
01932 * isServoModeEnabled(self: pyaubo_sdk.MotionControl) -> bool
01933 *
01934 * @par Lua function prototype
01935 * isServoModeEnabled() -> boolean
01936 *
01937 * @par Lua example
01938 * Servo_status = isServoModeEnabled()
01939 *
01940 * @par JSON-RPC request example
01941 * {"jsonrpc":"2.0","method":"rob1.MotionControl.isServoModeEnabled","params":[],"id":1}
01942 *
01943 * @par JSON-RPC response example
01944 * {"id":1,"jsonrpc":"2.0","result":false}
01945 *
01946 * \endenglish
01947 */
01948 ARCS_DEPRECATED bool isServoModeEnabled();
01949
01950 /**
01951 * @ingroup MotionControl
01952 * \chinese
01953 *
01954 *
01955 * @param mode
01956 * 0-
01957 * 1- ( )
01958 * 2- ( )
01959 * 3- ( )
01960 * 4-1ms ( )
01961 * 5-
01962 * 6- ( ) ,
01963 * 7-
01964 * 1 ( )

```

```

01965 * 5
01966 * @return
01967 *
01968 * @par Lua
01969 * setServoModeSelect(0) -> nil
01970 *
01971 * @par Lua
01972 * setServoModeSelect(0)
01973 *
01974 * @par JSON-RPC
01975 * {"jsonrpc":"2.0","method":"rob1.MotionControl.setServoModeSelect","params":[0],"id":1}
01976 *
01977 * @par JSON-RPC
01978 * {"id":1,"jsonrpc":"2.0","result":0}
01979 *
01980 * \endchinese
01981 * \english
01982 * Set servo motion mode
01983 *
01984 * @param mode
01985 * 0 - Exit servo mode
01986 * 1 - Planning servo mode
01987 * 2 - Transparent mode (direct send)
01988 * 3 - Transparent mode (buffered)
01989 * @return
01990 *
01991 * @par Lua function prototype
01992 * setServoModeSelect(0) -> nil
01993 *
01994 * @par Lua example
01995 * setServoModeSelect(0)
01996 *
01997 * @par JSON-RPC request example
01998 * {"jsonrpc":"2.0","method":"rob1.MotionControl.setServoModeSelect","params":[0],"id":1}
01999 *
02000 * @par JSON-RPC response example
02001 * {"id":1,"jsonrpc":"2.0","result":0}
02002 *
02003 * \endenglish
02004 */
02005 int setServoModeSelect(int mode);
02006
02007 /**
02008 * @ingroup MotionControl
02009 * \chinese
02010 *
02011 *
02012 * @return
02013 *
02014 * @par Lua
02015 * getServoModeSelect() -> number
02016 *
02017 * @par Lua
02018 * num = getServoModeSelect()
02019 *
02020 * @par JSON-RPC
02021 * {"jsonrpc":"2.0","method":"rob1.MotionControl.getServoModeSelect","params":[],"id":1}
02022 *
02023 * @par JSON-RPC
02024 * {"id":1,"jsonrpc":"2.0","result":0}
02025 *
02026 * \endchinese
02027 * \english
02028 * Get the servo motion mode
02029 *
02030 * @return
02031 *
02032 * @par Lua function prototype
02033 * getServoModeSelect() -> number
02034 *
02035 * @par Lua example
02036 * num = getServoModeSelect()
02037 *
02038 * @par JSON-RPC request example
02039 * {"jsonrpc":"2.0","method":"rob1.MotionControl.getServoModeSelect","params":[],"id":1}
02040 *
02041 * @par JSON-RPC response example
02042 * {"id":1,"jsonrpc":"2.0","result":0}
02043 *
02044 * \endenglish
02045 */
02046 int getServoModeSelect();
02047
02048 /**
02049 * @ingroup MotionControl
02050 * \chinese
02051 *

```

```

02052 *
02053 *      q t;
02054 * @param q      , rad,
02055 * @param a      , rad/s^2,
02056 * @param v      rad/s,
02057 * @param t      s \n
02058 * t      ,      ;
02059 *      servoJoint
02060 * @param lookahead_time      s \n
02061 *      [0.03, 0.2],
02062 *      lookahead_time      ,
02063 *
02064 * @param gain
02065 *      [100, 200],
02066 *      ,
02067 *
02068 *
02069 * @retval 0
02070 * @retval AUBO_BAD_STATE(1)
02071 * Normal ReducedMode Recovery
02072 * @retval AUBO_QUEUE_FULL(2)
02073 * @retval AUBO_BUSY(3)
02074 * @retval -AUBO_BAD_STATE(-1)
02075 *      task_id
02076 *      Running
02077 * @retval -AUBO_TIMEOUT(-4)
02078 * @retval -AUBO_INVL_ARGUMENT(-5)
02079 * @retval -AUBO_REQUEST_IGNORE(-13)      servo
02080 *
02081 * @throws arcs::common_interface::AuboException
02082 *
02083 * @par Python
02084 * servoJoint(self: pyaubo_sdk.MotionControl, arg0: List[float], arg1:
02085 * float, arg2: float, arg3: float, arg4: float, arg5: float) -> int
02086 *
02087 * @par Lua
02088 * servoJoint(q: table, a: number, v: number, t: number, lookahead_time:
02089 * number, gain: number) -> nil
02090 *
02091 * @par Lua
02092 * servoJoint({0,0,0,0,0,0},0,0,10,0,0)
02093 *
02094 * @par JSON-RPC
02095 * {"jsonrpc":"2.0","method":"rob1.MotionControl.servoJoint","params":[[0,0,0,0,0,0],0,0,10,0,0],"id":1}
02096 *
02097 * @par JSON-RPC
02098 * {"id":1,"jsonrpc":"2.0","result":-13}
02099 *
02100 * \endchinese
02101 * \english
02102 * Joint space servo
02103 *
02104 * Currently only q and t parameters are available;
02105 * @param q Joint angles, unit: rad
02106 * @param a Acceleration, unit: rad/s^2
02107 * @param v Velocity, unit: rad/s
02108 * @param t Execution time, unit: s \n
02109 * The larger the t value, the slower the robot moves, and vice versa;
02110 * The optimal value for this parameter is the interval time for continuous
02111 * calls to the servoJoint interface.
02112 * @param lookahead_time Lookahead time, unit: s \n
02113 * Specifies the duration to move before the robot starts to decelerate,
02114 * used to smooth the trajectory [0.03, 0.2]. When lookahead_time is less
02115 * than one control cycle, the smaller it is, the greater the overshoot. The
02116 * optimal value for this parameter is one control cycle.
02117 * @param gain Proportional gain
02118 * Proportional gain for tracking the target position [100, 200],
02119 * used to control the smoothness and accuracy of the motion.
02120 * The larger the proportional gain, the longer it takes to reach the target
02121 * position, and the smaller the overshoot.
02122 *
02123 * @retval 0 Success
02124 * @retval AUBO_BAD_STATE(1) The current safety mode is not Normal,
02125 * ReducedMode, or Recovery
02126 * @retval AUBO_QUEUE_FULL(2) Planning queue is full
02127 * @retval AUBO_BUSY(3) The previous instruction is being executed
02128 * @retval -AUBO_BAD_STATE(-1)
02129 * Possible reasons include but are not limited to: thread has been
02130 * detached, thread terminated, task_id not found, or the current robot mode
02131 * is not Running
02132 * @retval -AUBO_TIMEOUT(-4) Interface call timeout
02133 * @retval -AUBO_INVL_ARGUMENT(-5) Trajectory position or velocity out of
02134 * range
02135 * @retval -AUBO_REQUEST_IGNORE(-13) Not in servo mode
02136 *
02137 * @throws arcs::common_interface::AuboException
02138 *

```

```

02139 * @par Python function prototype
02140 * servoJoint(self: pyaubo_sdk.MotionControl, arg0: List[float], arg1:
02141 * float, arg2: float, arg3: float, arg4: float, arg5: float) -> int
02142 *
02143 * @par Lua function prototype
02144 * servoJoint(q: table, a: number, v: number, t: number, lookahead_time:
02145 * number, gain: number) -> nil
02146 *
02147 * @par Lua example
02148 * servoJoint({0,0,0,0,0,0},0,0,10,0,0)
02149 *
02150 * @par JSON-RPC request example
02151 * {"jsonrpc":"2.0","method":"rob1.MotionControl.servoJoint","params":[[0,0,0,0,0,0],0,0,10,0,0],"id":1}
02152 *
02153 * @par JSON-RPC response example
02154 * {"id":1,"jsonrpc":"2.0","result":-13}
02155 *
02156 * \endenglish
02157 */
02158 int servoJoint(const std::vector<double> &q, double a, double v, double t,
02159               double lookahead_time, double gain);
02160
02161 /**
02162 * @ingroup MotionControl
02163 * \chinese
02164 *
02165 *
02166 *     pose t;
02167 * @param pose , m,
02168 * @param a , m/s^2,
02169 * @param v m/s,
02170 * @param t s \n
02171 * t , , ;
02172 *     servoCartesian
02173 * @param lookahead_time s \n
02174 * [0.03, 0.2],
02175 * lookahead_time ,
02176 *
02177 * @param gain
02178 * [100, 200],
02179 * ,
02180 *
02181 *
02182 * @retval 0
02183 * @retval AUBO_BAD_STATE(1)
02184 * Normal ReducedMode Recovery
02185 * @retval AUBO_QUEUE_FULL(2)
02186 * @retval AUBO_BUSY(3)
02187 * @retval -AUBO_BAD_STATE(-1)
02188 * task_id
02189 * Running
02190 * @retval -AUBO_TIMEOUT(-4)
02191 * @retval -AUBO_INVL_ARGUMENT(-5)
02192 * @retval -AUBO_REQUEST_IGNORE(-13) servo
02193 * @retval -AUBO_IK_NO_CONVERGE(-23) ;
02194 * @retval -AUBO_IK_OUT_OF_RANGE(-24) ;
02195 * @retval AUBO_IK_CONFIG_DISMISMATCH(-25) ;
02196 * @retval -AUBO_IK_JACOBIAN_FAILED(-26) ;
02197 * @retval -AUBO_IK_NO_SOLU(-27) ;
02198 * @retval -AUBO_IK_UNKOWN_ERROR(-28) ;
02199 *
02200 * @throws arcs::common_interface::AuboException
02201 *
02202 * @par Python
02203 * servoCartesian(self: pyaubo_sdk.MotionControl, arg0: List[float], arg1:
02204 * float, arg2: float, arg3: float, arg4: float, arg5: float) -> int
02205 *
02206 * @par Lua
02207 * servoCartesian(pose: table, a: number, v: number, t: number,
02208 * lookahead_time: number, gain: number) -> nil
02209 *
02210 * @par Lua
02211 * servoCartesian({0.58712,-0.15775,0.48703,2.76,0.344,1.432},0,0,10,0,0)
02212 *
02213 * @par JSON-RPC
02214 * {"jsonrpc":"2.0","method":"rob1.MotionControl.servoCartesian","params":[[0.58712,-
0.15775,0.48703,2.76,0.344,1.432],0,0,10,0,0],"id":1}
02215 *
02216 * @par JSON-RPC
02217 * {"id":1,"jsonrpc":"2.0","result":-13}
02218 * \endchinese
02219 * \english
02220 * Cartesian space servo
02221 *
02222 * Currently, only pose and t parameters are available;
02223 * @param pose Pose, unit: m
02224 * @param a Acceleration, unit: m/s^2

```



```

02225 * @param v Velocity, unit: m/s
02226 * @param t Execution time, unit: s \n
02227 * The larger the t value, the slower the robot moves, and vice versa;
02228 * The optimal value for this parameter is the interval time for continuous
02229 * calls to the servoCartesian interface.
02230 * @param lookahead_time Lookahead time, unit: s \n
02231 * Specifies the duration to move before the robot starts to decelerate,
02232 * used to smooth the trajectory [0.03, 0.2]. When lookahead_time is less
02233 * than one control cycle, the smaller it is, the greater the overshoot. The
02234 * optimal value for this parameter is one control cycle.
02235 * @param gain Proportional gain
02236 * Proportional gain for tracking the target position [100, 200],
02237 * used to control the smoothness and accuracy of the motion,
02238 * The larger the proportional gain, the longer it takes to reach the target
02239 * position, and the smaller the overshoot.
02240 *
02241 * @retval 0 Success
02242 * @retval AUBO_BAD_STATE(1) The current safety mode is not Normal,
02243 * ReducedMode, or Recovery
02244 * @retval AUBO_QUEUE_FULL(2) Planning queue is full
02245 * @retval AUBO_BUSY(3) The previous instruction is being executed
02246 * @retval -AUBO_BAD_STATE(-1)
02247 * Possible reasons include but are not limited to: thread has been
02248 * detached, thread terminated, task_id not found, or the current robot mode
02249 * is not Running
02250 * @retval -AUBO_TIMEOUT(-4) Interface call timeout
02251 * @retval -AUBO_INVL_ARGUMENT(-5) Trajectory position or velocity out of
02252 * range
02253 * @retval -AUBO_REQUEST_IGNORE(-13) Not in servo mode
02254 * @retval -AUBO_IK_NO_CONVERGE(-23) Inverse kinematics calculation does not
02255 * converge, calculation error;
02256 * @retval -AUBO_IK_OUT_OF_RANGE(-24) Inverse kinematics calculation exceeds
02257 * robot maximum limits;
02258 * @retval AUBO_IK_CONFIG_DISMATH(-25) Inverse kinematics input
02259 * configuration error;
02260 * @retval -AUBO_IK_JACOBIAN_FAILED(-26) Inverse kinematics Jacobian
02261 * calculation failed;
02262 * @retval -AUBO_IK_NO_SOLU(-27) Analytical solution exists for the target
02263 * point, but none meet the selection criteria;
02264 * @retval -AUBO_IK_UNKOWN_ERROR(-28) Inverse kinematics returned an unknown
02265 * error type;
02266 *
02267 * @throws arcs::common_interface::AuboException
02268 *
02269 * @par Python function prototype
02270 * servoCartesian(self: pyaubo_sdk.MotionControl, arg0: List[float], arg1:
02271 * float, arg2: float, arg3: float, arg4: float, arg5: float) -> int
02272 *
02273 * @par Lua function prototype
02274 * servoCartesian(pose: table, a: number, v: number, t: number,
02275 * lookahead_time: number, gain: number) -> nil
02276 *
02277 * @par Lua example
02278 * servoCartesian({0.58712,-0.15775,0.48703,2.76,0.344,1.432},0,0,10,0,0)
02279 *
02280 * @par JSON-RPC request example
02281 * {"jsonrpc":"2.0","method":"rob1.MotionControl.servoCartesian","params":[[0.58712,-
0.15775,0.48703,2.76,0.344,1.432],0,0,10,0,0],"id":1}
02282 *
02283 * @par JSON-RPC response example
02284 * {"id":1,"jsonrpc":"2.0","result":-13}
02285 * \endenglish
02286 */
02287 int servoCartesian(const std::vector<double> &pose, double a, double v,
02288 double t, double lookahead_time, double gain);
02289
02290 /**
02291 * @ingroup MotionControl
02292 * \chinese
02293 *
02294 *
02295 * @param q
02296 * @param extq
02297 * @param t
02298 * @param smooth_scale
02299 * @param delay_sacle
02300 * @return
02301 * \endchinese
02302 * \english
02303 * Servo motion (with external axes), used for executing offline
02304 * trajectories, pass-through user planned trajectories, etc.
02305 *
02306 * @param q
02307 * @param extq
02308 * @param t
02309 * @param smooth_scale
02310 * @param delay_sacle

```

```

02311     * @return
02312     * \endenglish
02313     */
02314 int servoJointWithAxes(const std::vector<double> &q,
02315                       const std::vector<double> &extq, double a, double v,
02316                       double t, double lookahead_time, double gain);
02317
02318 int servoJointWithAxisGroup(const std::vector<double> &q, double a,
02319                             double v, double t, double lookahead_time,
02320                             double gain, const std::string &group_name,
02321                             const std::vector<double> &extq);
02322
02323 /**
02324  * @ingroup MotionControl
02325  * \chinese
02326  *
02327  * servoJointWithAxes
02328  * (      )
02329  *
02330  * @param pose
02331  * @param extq
02332  * @param t
02333  * @param smooth_scale
02334  * @param delay_sacle
02335  * @return
02336  * \endchinese
02337  * \english
02338  * Servo motion (with external axes), used for executing offline
02339  * trajectories, pass-through user planned trajectories, etc. The difference
02340  * from servoJointWithAxes is that it receives Cartesian poses instead of
02341  * joint angles (inverse kinematics is performed internally by the
02342  * software).
02343  *
02344  * @param pose
02345  * @param extq
02346  * @param t
02347  * @param smooth_scale
02348  * @param delay_sacle
02349  * @return
02350  * \endenglish
02351  */
02352 int servoCartesianWithAxes(const std::vector<double> &pose,
02353                           const std::vector<double> &extq, double a,
02354                           double v, double t, double lookahead_time,
02355                           double gain);
02356
02357 int servoCartesianWithAxisGroup(const std::vector<double> &pose, double a,
02358                                 double v, double t, double lookahead_time,
02359                                 double gain, const std::string &group_name,
02360                                 const std::vector<double> &extq);
02361
02362 /**
02363  * @ingroup MotionControl
02364  * \chinese
02365  *
02366  *
02367  * @param q
02368  * @param smooth_scale
02369  * @param delay_sacle
02370  * @return
02371  *
02372  * @par Lua
02373  * trackJoint(q: table,t: number,smooth_scale: number,delay_sacle: number)
02374  * -> nil
02375  *
02376  * @par Lua
02377  * trackJoint({0,0,0,0,0,0},0.01,0.5,1)
02378  *
02379  * @par JSON-RPC
02380  * {"jsonrpc":"2.0","method":"rob1.MotionControl.trackJoint","params":[[0,0,0,0,0,0],0.01,0.5,1],"id":1}
02381  *
02382  * @par JSON-RPC
02383  * {"id":1,"jsonrpc":"2.0","result":0}
02384  *
02385  * \endchinese
02386  * \english
02387  * Tracking motion, used for executing offline trajectories or passing
02388  * through user-planned trajectories, etc.
02389  *
02390  * @param q
02391  * @param smooth_scale
02392  * @param delay_sacle
02393  * @return
02394  *
02395  * @par Lua function prototype
02396  * trackJoint(q: table,t: number,smooth_scale: number,delay_sacle: number)
02397  * -> nil

```

```

02398  *
02399  * @par Lua example
02400  * trackJoint({0,0,0,0,0},0.01,0.5,1)
02401  *
02402  * @par JSON-RPC request example
02403  * {"jsonrpc":"2.0","method":"rob1.MotionControl.trackJoint","params":[[0,0,0,0,0],0.01,0.5,1],"id":1}
02404  *
02405  * @par JSON-RPC response example
02406  * {"id":1,"jsonrpc":"2.0","result":0}
02407  *
02408  * \endenglish
02409  */
02410 int trackJoint(const std::vector<double> &q, double t, double smooth_scale,
02411               double delay_sacle);
02412
02413 /**
02414  * @ingroup MotionControl
02415  * \chinese
02416  *
02417  * trackJoint
02418  * (
02419  *
02420  * @param pose
02421  * @param t
02422  * @param smooth_scale
02423  * @param delay_sacle
02424  * @return
02425  *
02426  * @par Lua
02427  * trackCartesian(pose: table,t: number,smooth_scale: number,delay_sacle:
02428  * number) -> nil
02429  *
02430  * @par Lua
02431  * trackCartesian({0.58712,-0.15775,0.48703,2.76,0.344,1.432},0.01,0.5,1)
02432  *
02433  * @par JSON-RPC
02434  * {"jsonrpc":"2.0","method":"rob1.MotionControl.trackCartesian","params":[[0.58712,-
0.15775,0.48703,2.76,0.344,1.432],0.01,0.5,1],"id":1}
02435  *
02436  * @par JSON-RPC
02437  * {"id":1,"jsonrpc":"2.0","result":0}
02438  * \endchinese
02439  * \english
02440  * Tracking motion, used for executing offline trajectories or passing
02441  * through user-planned trajectories, etc. The difference from trackJoint is
02442  * that it receives Cartesian poses instead of joint angles (inverse
02443  * kinematics is performed internally by the software).
02444  *
02445  * @param pose
02446  * @param t
02447  * @param smooth_scale
02448  * @param delay_sacle
02449  * @return
02450  *
02451  * @par Lua function prototype
02452  * trackCartesian(pose: table,t: number,smooth_scale: number,delay_sacle:
02453  * number) -> nil
02454  *
02455  * @par Lua example
02456  * trackCartesian({0.58712,-0.15775,0.48703,2.76,0.344,1.432},0.01,0.5,1)
02457  *
02458  * @par JSON-RPC request example
02459  * {"jsonrpc":"2.0","method":"rob1.MotionControl.trackCartesian","params":[[0.58712,-
0.15775,0.48703,2.76,0.344,1.432],0.01,0.5,1],"id":1}
02460  *
02461  * @par JSON-RPC response example
02462  * {"id":1,"jsonrpc":"2.0","result":0}
02463  * \endenglish
02464  */
02465 int trackCartesian(const std::vector<double> &pose, double t,
02466                   double smooth_scale, double delay_sacle);
02467
02468 /**
02469  * @ingroup MotionControl
02470  * \chinese
02471  *
02472  *
02473  * @note
02474  *
02475  * @throws arcs::common_interface::AuboException
02476  *
02477  * @par Python
02478  * followJoint(self: pyaubo_sdk.MotionControl, arg0: List[float]) -> int
02479  *
02480  * @par Lua
02481  * followJoint(q: table) -> nil
02482  *

```

```

02483 * @par Lua
02484 * followJoint({0,0,0,0,0,0})
02485 *
02486 * \endchinese
02487 * \english
02488 * Joint space following
02489 *
02490 * @note Not implemented yet
02491 *
02492 * @throws arcs::common_interface::AuboException
02493 *
02494 * @par Python function prototype
02495 * followJoint(self: pyaubo_sdk.MotionControl, arg0: List[float]) -> int
02496 *
02497 * @par Lua function prototype
02498 * followJoint(q: table) -> nil
02499 *
02500 * @par Lua example
02501 * followJoint({0,0,0,0,0,0})
02502 *
02503 * \endenglish
02504 */
02505 int followJoint(const std::vector<double> &q);
02506
02507 /**
02508 * @ingroup MotionControl
02509 * \chinese
02510 *
02511 *
02512 * @note
02513 *
02514 * @throws arcs::common_interface::AuboException
02515 *
02516 * @par Python
02517 * followLine(self: pyaubo_sdk.MotionControl, arg0: List[float]) -> int
02518 *
02519 * @par Lua
02520 * followLine(pose: table) -> nil
02521 *
02522 * @par Lua
02523 * followLine({0.58712,-0.15775,0.48703,2.76,0.344,1.432})
02524 *
02525 * \endchinese
02526 * \english
02527 * Cartesian space following
02528 *
02529 * @note Not implemented yet
02530 *
02531 * @throws arcs::common_interface::AuboException
02532 *
02533 * @par Python function prototype
02534 * followLine(self: pyaubo_sdk.MotionControl, arg0: List[float]) -> int
02535 *
02536 * @par Lua function prototype
02537 * followLine(pose: table) -> nil
02538 *
02539 * @par Lua example
02540 * followLine({0.58712,-0.15775,0.48703,2.76,0.344,1.432})
02541 *
02542 * \endenglish
02543 */
02544 int followLine(const std::vector<double> &pose);
02545
02546 /**
02547 * @ingroup MotionControl
02548 * \chinese
02549 *
02550 *
02551 *
02552 *
02553 * @param qd      , rad/s
02554 * @param a      , rad/s^2
02555 * @param t      , s \n
02556 * t = 0
02557 * t          \n
02558 *
02559 *
02560 * @retval 0
02561 * @retval AUBO_BAD_STATE(1)
02562 * Normal ReducedMode Recovery
02563 * @retval AUBO_BUSY(3)
02564 * @retval -AUBO_BAD_STATE(-1)
02565 * task_id
02566 * Running
02567 * @retval -AUBO_TIMEOUT(-4)
02568 * @retval -AUBO_INVL_ARGUMENT(-5) qd
02569 *

```

```

02570 * @throws arcs::common_interface::AuboException
02571 *
02572 * @par Python
02573 * speedJoint(self: pyaubo_sdk.MotionControl, arg0: List[float], arg1:
02574 * float, arg2: float) -> int
02575 *
02576 * @par Lua
02577 * speedJoint(qd: table, a: number, t: number) -> nil
02578 *
02579 * @par Lua
02580 * speedJoint({0.2,0,0,0,0,0}, 1.5,10)
02581 *
02582 * @par JSON-RPC
02583 * {"jsonrpc":"2.0","method":"rob1.MotionControl.speedJoint","params":[[0.2,0,0,0,0,0],1.5,100],"id":1}
02584 *
02585 * @par JSON-RPC
02586 * {"id":1,"jsonrpc":"2.0","result":0}
02587 *
02588 * \endchinese
02589 * \english
02590 * Joint space velocity following
02591 *
02592 * When the robot arm has not yet reached the target velocity, giving a new
02593 * target velocity will cause the robot arm to immediately reach the new
02594 * target velocity.
02595 *
02596 * @param qd Target joint velocity, unit: rad/s
02597 * @param a Main axis acceleration, unit: rad/s^2
02598 * @param t Time required for the function to return, unit: s \n
02599 * If t = 0, the function returns when the target velocity is reached;
02600 * otherwise, the function returns after t seconds, regardless of whether
02601 * the target velocity is reached.\n If the target velocity is not reached,
02602 * it will decelerate to zero. If the target velocity is reached, it will
02603 * move at a constant speed.
02604 * @retval 0 Success
02605 * @retval AUBO_BAD_STATE(1) The current safety mode is not Normal,
02606 * ReducedMode, or Recovery
02607 * @retval AUBO_BUSY(3) The previous instruction is being executed
02608 * @retval -AUBO_BAD_STATE(-1)
02609 * Possible reasons include but are not limited to: thread has been
02610 * detached, thread terminated, task_id not found, or the current robot mode
02611 * is not Running
02612 * @retval -AUBO_TIMEOUT(-4) Interface call timeout
02613 * @retval -AUBO_INVL_ARGUMENT(-5) The length of the qd array is less than
02614 * the degrees of freedom of the current robot arm
02615 *
02616 * @throws arcs::common_interface::AuboException
02617 *
02618 * @par Python function prototype
02619 * speedJoint(self: pyaubo_sdk.MotionControl, arg0: List[float], arg1:
02620 * float, arg2: float) -> int
02621 *
02622 * @par Lua function prototype
02623 * speedJoint(qd: table, a: number, t: number) -> nil
02624 *
02625 * @par Lua example
02626 * speedJoint({0.2,0,0,0,0,0}, 1.5,10)
02627 *
02628 * @par JSON-RPC request example
02629 * {"jsonrpc":"2.0","method":"rob1.MotionControl.speedJoint","params":[[0.2,0,0,0,0,0],1.5,100],"id":1}
02630 *
02631 * @par JSON-RPC response example
02632 * {"id":1,"jsonrpc":"2.0","result":0}
02633 *
02634 * \endenglish
02635 */
02636 int speedJoint(const std::vector<double> &qd, double a, double t);
02637
02638 /**
02639 * @ingroup MotionControl
02640 * \chinese
02641 * ( , )
02642 *
02643 *
02644 *
02645 * @param qd , rad/s
02646 * @param a , rad/s^2
02647 * @param t , s
02648 * t = 0
02649 * t
02650 *
02651 *
02652 * @return 0;
02653 * AUBO_BUSY
02654 * -AUBO_INVL_ARGUMENT
02655 * -AUBO_BAD_STATE
02656 *

```

```

02657 * @throws arcs::common_interface::AuboException
02658 *
02659 * @par Python
02660 * resumeSpeedJoint(self: pyaubo_sdk.MotionControl, arg0: List[float], arg1:
02661 * float, arg2: float) -> int
02662 *
02663 * @par Lua
02664 * resumeSpeedJoint(q: table, a: number, t: number) -> nil
02665 *
02666 * @par Lua
02667 * resumeSpeedJoint({0.2,0,0,0,0,0},1.5,10)
02668 *
02669 * @par JSON-RPC
02670 * {"jsonrpc":"2.0","method":"rob1.MotionControl.resumeSpeedJoint","params":[[0.2,0,0,0,0,0],1.5,100],"id":1}
02671 *
02672 * @par JSON-RPC
02673 * {"id":1,"jsonrpc":"2.0","result":-1}
02674 * \endchinese
02675 * \english
02676 * Joint space velocity following (used to move to a safe position after a
02677 * collision during process execution)
02678 *
02679 * When the robot arm has not yet reached the target velocity, giving a new
02680 * target velocity will cause the robot arm to immediately reach the new
02681 * target velocity.
02682 *
02683 * @param qd Target joint velocity, unit: rad/s
02684 * @param a Main axis acceleration, unit: rad/s^2
02685 * @param t Time required for the function to return, unit: s
02686 * If t = 0, the function returns when the target velocity is reached;
02687 * otherwise, the function returns after t seconds, regardless of whether
02688 * the target velocity is reached. If the target velocity is not reached, it
02689 * will decelerate to zero. If the target velocity is reached, it will move
02690 * at a constant speed.
02691 * @return Returns 0 on success; otherwise returns error code
02692 * AUBO_BUSY
02693 * -AUBO_INVL_ARGUMENT
02694 * -AUBO_BAD_STATE
02695 *
02696 * @throws arcs::common_interface::AuboException
02697 *
02698 * @par Python function prototype
02699 * resumeSpeedJoint(self: pyaubo_sdk.MotionControl, arg0: List[float], arg1:
02700 * float, arg2: float) -> int
02701 *
02702 * @par Lua function prototype
02703 * resumeSpeedJoint(q: table, a: number, t: number) -> nil
02704 *
02705 * @par Lua example
02706 * resumeSpeedJoint({0.2,0,0,0,0,0},1.5,10)
02707 *
02708 * @par JSON-RPC request example
02709 * {"jsonrpc":"2.0","method":"rob1.MotionControl.resumeSpeedJoint","params":[[0.2,0,0,0,0,0],1.5,100],"id":1}
02710 *
02711 * @par JSON-RPC response example
02712 * {"id":1,"jsonrpc":"2.0","result":-1}
02713 * \endenglish
02714 */
02715 int resumeSpeedJoint(const std::vector<double> &qd, double a, double t);
02716
02717 /**
02718 * @ingroup MotionControl
02719 * \chinese
02720 *
02721 *
02722 *
02723 *
02724 * @param xd , m/s
02725 * @param a , m/s^2
02726 * @param t , s \n
02727 * t = 0
02728 * t
02729 *
02730 *
02731 * @retval 0
02732 * @retval AUBO_BAD_STATE(1)
02733 * Normal ReducedMode Recovery
02734 * @retval AUBO_BUSY(3)
02735 * @retval -AUBO_BAD_STATE(-1)
02736 * task_id
02737 * Running
02738 * @retval -AUBO_TIMEOUT(-4)
02739 *
02740 * @throws arcs::common_interface::AuboException
02741 *
02742 * @par Python
02743 * speedLine(self: pyaubo_sdk.MotionControl, arg0: List[float], arg1: float,

```

```

02744 * arg2: float) -> int
02745 *
02746 * @par Lua
02747 * speedLine(pose: table, a: number, t: number) -> nil
02748 *
02749 * @par Lua
02750 * speedLine({0.25,0,0,0,0,0},1.2,100)
02751 *
02752 * @par JSON-RPC
02753 * {"jsonrpc":"2.0","method":"rob1.MotionControl.speedLine","params":[[0.25,0,0,0,0,0],1.2,100],"id":1}
02754 *
02755 * @par JSON-RPC
02756 * {"id":1,"jsonrpc":"2.0","result":0}
02757 *
02758 * \endchinese
02759 * \english
02760 * Cartesian space velocity following
02761 *
02762 * When the robot arm has not yet reached the target velocity, giving a new
02763 * target velocity will cause the robot arm to immediately reach the new
02764 * target velocity.
02765 *
02766 * @param xd Tool velocity, unit: m/s
02767 * @param a Tool position acceleration, unit: m/s^2
02768 * @param t Time required for the function to return, unit: s \n
02769 * If t = 0, the function returns when the target velocity is reached;
02770 * otherwise, the function returns after t seconds, regardless of whether
02771 * the target velocity is reached. If the target velocity is not reached, it
02772 * will decelerate to zero. If the target velocity is reached, it will move
02773 * at a constant speed.
02774 * @retval 0 Success
02775 * @retval AUBO_BAD_STATE(1) The current safety mode is not Normal,
02776 * ReducedMode, or Recovery
02777 * @retval AUBO_BUSY(3) The previous instruction is being executed
02778 * @retval -AUBO_BAD_STATE(-1)
02779 * Possible reasons include but are not limited to: thread has been
02780 * detached, thread terminated, task_id not found, or the current robot mode
02781 * is not Running
02782 * @retval -AUBO_TIMEOUT(-4) Interface call timeout
02783 *
02784 * @throws arcs::common_interface::AuboException
02785 *
02786 * @par Python function prototype
02787 * speedLine(self: pyaubo_sdk.MotionControl, arg0: List[float], arg1: float,
02788 * arg2: float) -> int
02789 *
02790 * @par Lua function prototype
02791 * speedLine(pose: table, a: number, t: number) -> nil
02792 *
02793 * @par Lua example
02794 * speedLine({0.25,0,0,0,0,0},1.2,100)
02795 *
02796 * @par JSON-RPC request example
02797 * {"jsonrpc":"2.0","method":"rob1.MotionControl.speedLine","params":[[0.25,0,0,0,0,0],1.2,100],"id":1}
02798 *
02799 * @par JSON-RPC response example
02800 * {"id":1,"jsonrpc":"2.0","result":0}
02801 *
02802 * \endenglish
02803 */
02804 int speedLine(const std::vector<double> &xd, double a, double t);
02805
02806 /**
02807 * @ingroup MotionControl
02808 * \chinese
02809 * ( , )
02810 *
02811 *
02812 *
02813 * @param xd , m/s
02814 * @param a , m/s^2
02815 * @param t , s \n
02816 * t = 0
02817 * t
02818 *
02819 *
02820 * @return 0;
02821 * -AUBO_BAD_STATE
02822 *
02823 * @throws arcs::common_interface::AuboException
02824 *
02825 * @par Python
02826 * resumeSpeedLine(self: pyaubo_sdk.MotionControl, arg0: List[float], arg1:
02827 * float, arg2: float) -> int
02828 *
02829 * @par Lua
02830 * resumeSpeedLine(pose: table, a: number, t: number) -> nil

```

```

02831 *
02832 * @par Lua
02833 * resumeSpeedLine({0.25,0,0,0,0},1.2,100)
02834 *
02835 * @par JSON-RPC
02836 * {"jsonrpc":"2.0","method":"rob1.MotionControl.resumeSpeedLine","params":[[0.25,0,0,0,0],1.2,100],"id":1}
02837 *
02838 * @par JSON-RPC
02839 * {"id":1,"jsonrpc":"2.0","result":-1}
02840 * \endchinese
02841 * \english
02842 * Cartesian space velocity following (used to move to a safe position after
02843 * a collision during process execution)
02844 *
02845 * When the robot arm has not yet reached the target velocity, giving a new
02846 * target velocity will cause the robot arm to immediately reach the new
02847 * target velocity.
02848 *
02849 * @param xd Tool velocity, unit: m/s
02850 * @param a Tool position acceleration, unit: m/s^2
02851 * @param t Time required for the function to return, unit: s \n
02852 * If t = 0, the function returns when the target velocity is reached;
02853 * otherwise, the function returns after t seconds, regardless of whether
02854 * the target velocity is reached. If the target velocity is not reached, it
02855 * will decelerate to zero. If the target velocity is reached, it will move
02856 * at a constant speed.
02857 * @return Returns 0 on success; otherwise returns error code
02858 * -AUBO_BAD_STATE
02859 *
02860 * @throws arcs::common_interface::AuboException
02861 *
02862 * @par Python function prototype
02863 * resumeSpeedLine(self: pyaubo_sdk.MotionControl, arg0: List[float], arg1:
02864 * float, arg2: float) -> int
02865 *
02866 * @par Lua function prototype
02867 * resumeSpeedLine(pose: table, a: number, t: number) -> nil
02868 *
02869 * @par Lua example
02870 * resumeSpeedLine({0.25,0,0,0,0},1.2,100)
02871 *
02872 * @par JSON-RPC request example
02873 * {"jsonrpc":"2.0","method":"rob1.MotionControl.resumeSpeedLine","params":[[0.25,0,0,0,0],1.2,100],"id":1}
02874 *
02875 * @par JSON-RPC response example
02876 * {"id":1,"jsonrpc":"2.0","result":-1}
02877 * \endenglish
02878 */
02879 int resumeSpeedLine(const std::vector<double> &xd, double a, double t);
02880
02881 /**
02882 * @ingroup MotionControl
02883 * \chinese
02884 *
02885 *
02886 * @param q 0
02887 * @param a , rad/s^2,
02888 * RobotConfig getJointMaxAccelerations()
02889 * @param v rad/s,
02890 * RobotConfig getJointMaxSpeeds()
02891 * @param duration
02892 * @return 0;
02893 * AUBO_BUSY
02894 * AUBO_BAD_STATE
02895 * -AUBO_BAD_STATE
02896 *
02897 * @throws arcs::common_interface::AuboException
02898 *
02899 * @par Python
02900 * moveSpline(self: pyaubo_sdk.MotionControl, arg0: List[float], arg1:
02901 * float, arg2: float, arg3: float) -> int
02902 *
02903 * @par Lua
02904 * moveSpline(q: table, a: number, v: number, duration: number) -> nil
02905 *
02906 * @par Lua
02907 * moveSpline({-2.05177,-0.400292, 1.19625, 0.0285152, 1.57033,
02908 * -2.28774},1.3,1.0,0)
02909 *
02910 * @par JSON-RPC
02911 * {"jsonrpc":"2.0","method":"rob1.MotionControl.moveSpline","params":[[0,0,0,0,0],1,1,0],"id":1}
02912 *
02913 * @par JSON-RPC
02914 * {"id":1,"jsonrpc":"2.0","result":0}
02915 * \endchinese
02916 * \english
02917 * Perform spline interpolation in joint space

```



```

02918 *
02919 * @param q Joint angles. If the input vector is of size 0, it indicates the
02920 * end of the spline motion.
02921 * @param a Acceleration, unit: rad/s^2. The maximum value can be obtained
02922 * via RobotConfig::getJointMaxAccelerations().
02923 * @param v Velocity, unit: rad/s. The maximum value can be obtained via
02924 * RobotConfig::getJointMaxSpeeds().
02925 * @param duration
02926 * @return Returns 0 on success; otherwise returns an error code:
02927 * AUBO_BUSY
02928 * AUBO_BAD_STATE
02929 * -AUBO_BAD_STATE
02930 *
02931 * @throws arcs::common_interface::AuboException
02932 *
02933 * @par Python function prototype
02934 * moveSpline(self: pyaubo_sdk.MotionControl, arg0: List[float], arg1:
02935 * float, arg2: float, arg3: float) -> int
02936 *
02937 * @par Lua function prototype
02938 * moveSpline(q: table, a: number, v: number, duration: number) -> nil
02939 *
02940 * @par Lua example
02941 * moveSpline({-2.05177,-0.400292, 1.19625, 0.0285152, 1.57033,
02942 * -2.28774},1.3,1.0,0)
02943 *
02944 * @par JSON-RPC request example
02945 * {"jsonrpc":"2.0","method":"rob1.MotionControl.moveSpline","params":[[0,0,0,0,0],1,1,0],"id":1}
02946 *
02947 * @par JSON-RPC response example
02948 * {"id":1,"jsonrpc":"2.0","result":0}
02949 * \endenglish
02950 */
02951 int moveSpline(const std::vector<double> &q, double a, double v,
02952               double duration);
02953
02954 /**
02955 * @ingroup MotionControl
02956 * \chinese
02957 *
02958 *
02959 * @param q , rad
02960 * @param a , rad/s^2.
02961 * RobotConfig getJointMaxAccelerations()
02962 * @param v rad/s,
02963 * RobotConfig getJointMaxSpeeds()
02964 * @param blend_radius , m
02965 * @param duration s \n
02966 *
02967 * duration
02968 * duration \n
02969 * duration = 0
02970 *
02971 * duration 0 a v
02972 * @retval 0
02973 * @retval AUBO_BAD_STATE(1)
02974 * Normal ReducedMode Recovery
02975 * @retval AUBO_QUEUE_FULL(2)
02976 * @retval AUBO_BUSY(3)
02977 * @retval -AUBO_BAD_STATE(-1)
02978 * task_id
02979 * Running
02980 * @retval -AUBO_TIMEOUT(-4)
02981 * @retval -AUBO_INVL_ARGUMENT(-5) q
02982 * @retval AUBO_REQUEST_IGNORE(13) q
02983 *
02984 * @throws arcs::common_interface::AuboException
02985 *
02986 * @par Python
02987 * moveJoint(self: pyaubo_sdk.MotionControl, arg0: List[float], arg1: float,
02988 * arg2: float, arg3: float, arg4: float) -> int
02989 *
02990 * @par Lua
02991 * moveJoint(q: table, a: number, v: number, blend_radius: number, duration:
02992 * number) -> nil
02993 *
02994 * @par Lua
02995 * moveJoint({-2.05177,-0.400292, 1.19625, 0.0285152, 1.57033,
02996 * -2.28774},0.3,0.3,0,0)
02997 *
02998 * @par JSON-RPC
02999 * {"jsonrpc":"2.0","method":"rob1.MotionControl.moveJoint","params":[[-2.05177,
03000 * -0.400292, 1.19625, 0.0285152, 1.57033, -2.28774],0.3,0.3,0,0],"id":1}
03001 *
03002 * @par JSON-RPC
03003 * {"id":1,"jsonrpc":"2.0","result":0}
03004 *

```

```

03005 * \endchinese
03006 * \english
03007 * Add joint motion
03008 *
03009 * @param q Joint angles, unit: rad
03010 * @param a Acceleration, unit: rad/s2,
03011 * The maximum value can be obtained via
03012 * RobotConfig::getJointMaxAccelerations()
03013 * @param v Velocity, unit: rad/s,
03014 * The maximum value can be obtained via RobotConfig::getJointMaxSpeeds()
03015 * @param blend_radius Blend radius, unit: m
03016 * @param duration Execution time, unit: s \n
03017 * Normally, when speed and acceleration are given, the trajectory duration
03018 * can be determined. If you want to extend the trajectory duration, set the
03019 * duration parameter. Duration can extend the trajectory time, but cannot
03020 * shorten it.\n When duration = 0, it means the execution time is not
03021 * specified, and the time will be calculated based on speed and
03022 * acceleration. If duration is not 0, a and v values will be ignored.
03023 * @retval 0 Success
03024 * @retval AUBO_BAD_STATE(1) The current safety mode is not Normal,
03025 * ReducedMode, or Recovery
03026 * @retval AUBO_QUEUE_FULL(2) Planning queue is full
03027 * @retval AUBO_BUSY(3) The previous instruction is being executed
03028 * @retval -AUBO_BAD_STATE(-1)
03029 * Possible reasons include but are not limited to: thread has been
03030 * detached, thread terminated, task_id not found, or the current robot mode
03031 * is not Running
03032 * @retval -AUBO_TIMEOUT(-4) Interface call timeout
03033 * @retval -AUBO_INVL_ARGUMENT(-5) The length of q is less than the degrees
03034 * of freedom of the current robot arm
03035 * @retval AUBO_REQUEST_IGNORE(13) The length of q is too short and there is
03036 * no need to stop at that point
03037 *
03038 * @throws arcs::common_interface::AuboException
03039 *
03040 * @par Python function prototype
03041 * moveJoint(self: pyaubo_sdk.MotionControl, arg0: List[float], arg1: float,
03042 * arg2: float, arg3: float, arg4: float) -> int
03043 *
03044 * @par Lua function prototype
03045 * moveJoint(q: table, a: number, v: number, blend_radius: number, duration:
03046 * number) -> nil
03047 *
03048 * @par Lua example
03049 * moveJoint({-2.05177,-0.400292, 1.19625, 0.0285152, 1.57033,
03050 * -2.28774},0.3,0.3,0,0)
03051 *
03052 * @par JSON-RPC request example
03053 * {"jsonrpc":"2.0","method":"rob1.MotionControl.moveJoint","params":[[-2.05177,
03054 * -0.400292, 1.19625, 0.0285152, 1.57033, -2.28774],0.3,0.3,0,0],"id":1}
03055 *
03056 * @par JSON-RPC response example
03057 * {"id":1,"jsonrpc":"2.0","result":0}
03058 *
03059 * \endenglish
03060 */
03061 int moveJoint(const std::vector<double> &q, double a, double v,
03062 double blend_radius, double duration);
03063
03064 /**
03065 * @ingroup MotionControl
03066 * \chinese
03067 *
03068 *
03069 * @param group_name
03070 * @param q
03071 * @param a
03072 * @param v
03073 * @param blend_radius
03074 * @param duration
03075 * @return
03076 * \endchinese
03077 * \english
03078 * Synchronous motion of robot and external axes
03079 *
03080 * @param group_name
03081 * @param q
03082 * @param a
03083 * @param v
03084 * @param blend_radius
03085 * @param duration
03086 * @return
03087 * \endenglish
03088 */
03089 int moveJointWithAxisGroup(const std::vector<double> &q, double a, double v,
03090 double blend_radius, double duration,
03091 const std::string &group_name,

```

```

03092         const std::vector<double> &extq);
03093
03094     /**
03095     * @ingroup MotionControl
03096     * \chinese
03097     *
03098     *
03099     * @param q      , rad
03100     * @param a      , rad/s^2
03101     * @param v      , rad/s
03102     * @param duration      s \n
03103     *
03104     *      duration
03105     * duration      \n
03106     * duration = 0
03107     *
03108     * duration 0 a v
03109     * @return 0
03110     * -AUBO_INVL_ARGUMENT
03111     * -AUBO_BAD_STATE
03112     *
03113     * @throws arcs::common_interface::AuboException
03114     *
03115     * @par Python
03116     * resumeMoveJoint(self: pyaubo_sdk.MotionControl, arg0: List[float], arg1:
03117     * float, arg2: float, arg3: float) -> int
03118     *
03119     * @par Lua
03120     * resumeMoveJoint(q: table, a: number, v: number, duration: number) -> nil
03121     *
03122     * @par Lua
03123     * resumeMoveJoint({-2.05177,-0.400292, 1.19625, 0.0285152, 1.57033,
03124     * -2.28774},1.3,1.0,0)
03125     *
03126     * @par JSON-RPC
03127     * {"jsonrpc":"2.0","method":"rob1.MotionControl.resumeMoveJoint","params":[[-2.05177,
03128     * -0.400292, 1.19625, 0.0285152, 1.57033, -2.28774],0.3,0.3,0],"id":1}
03129     *
03130     * @par JSON-RPC
03131     * {"id":1,"jsonrpc":"2.0","result":-1}
03132     * \endchinese
03133     * \english
03134     * Move to the pause point using joint motion.
03135     *
03136     * @param q Joint angles, unit: rad
03137     * @param a Acceleration, unit: rad/s^2
03138     * @param v Velocity, unit: rad/s
03139     * @param duration Execution time, unit: s \n
03140     * Normally, when speed and acceleration are given, the trajectory duration
03141     * can be determined. If you want to extend the trajectory duration, set the
03142     * duration parameter. Duration can extend the trajectory time, but cannot
03143     * shorten it.\n When duration = 0, it means the execution time is not
03144     * specified, and the time will be calculated based on speed and
03145     * acceleration. If duration is not 0, a and v values will be ignored.
03146     * @return Returns 0 on success; otherwise returns error code
03147     * -AUBO_INVL_ARGUMENT
03148     * -AUBO_BAD_STATE
03149     *
03150     * @throws arcs::common_interface::AuboException
03151     *
03152     * @par Python function prototype
03153     * resumeMoveJoint(self: pyaubo_sdk.MotionControl, arg0: List[float], arg1:
03154     * float, arg2: float, arg3: float) -> int
03155     *
03156     * @par Lua function prototype
03157     * resumeMoveJoint(q: table, a: number, v: number, duration: number) -> nil
03158     *
03159     * @par Lua example
03160     * resumeMoveJoint({-2.05177,-0.400292, 1.19625, 0.0285152, 1.57033,
03161     * -2.28774},1.3,1.0,0)
03162     *
03163     * @par JSON-RPC request example
03164     * {"jsonrpc":"2.0","method":"rob1.MotionControl.resumeMoveJoint","params":[[-2.05177,
03165     * -0.400292, 1.19625, 0.0285152, 1.57033, -2.28774],0.3,0.3,0],"id":1}
03166     *
03167     * @par JSON-RPC response example
03168     * {"id":1,"jsonrpc":"2.0","result":-1}
03169     * \endenglish
03170     */
03171     int resumeMoveJoint(const std::vector<double> &q, double a, double v,
03172         double duration);
03173
03174     /**
03175     * @ingroup MotionControl
03176     * \chinese
03177     *
03178     *

```

```

03179 * @param pose
03180 * @param a ( 1mm, 1e-4
03181 * rad, , rad/s^2. , m/s^2)
03182 * RobotConfig::getTcpMaxAccelerations()
03183 * @param v ( 1mm, 1e-4
03184 * rad, , rad/s. , m/s)
03185 * RobotConfig::getTcpMaxSpeeds()
03186 * @param blend_radius m 0.001 1
03187 * @param duration s \n
03188 *
03189 * duration
03190 * duration \n
03191 * duration = 0
03192 *
03193 * duration 0 a v
03194 * @retval 0
03195 * @retval AUBO_BAD_STATE(1)
03196 * Normal ReducedMode Recovery
03197 * @retval AUBO_QUEUE_FULL(2)
03198 * @retval AUBO_BUSY(3)
03199 * @retval -AUBO_BAD_STATE(-1)
03200 * task_id
03201 * Running
03202 * @retval -AUBO_TIMEOUT(-4)
03203 * @retval -AUBO_INVL_ARGUMENT(-5) pose
03204 *
03205 * @retval AUBO_REQUEST_IGNORE(13) pose
03206 *
03207 *
03208 * @throws arcs::common_interface::AuboException
03209 *
03210 * @par Python
03211 * moveLine(self: pyaubo_sdk.MotionControl, arg0: List[float], arg1: float,
03212 * arg2: float, arg3: float, arg4: float) -> int
03213 *
03214 * @par Lua
03215 * moveLine(pose: table, a: number, v: number, blend_radius: number,
03216 * duration: number) -> nil
03217 *
03218 * @par Lua
03219 * moveLine({0.58815,0.0532,0.62391,2.46,0.479,1.619},1.3,1.0,0)
03220 *
03221 * @par JSON-RPC
03222 * {"json-
rpc":"2.0","method":"rob1.MotionControl.moveLine","params":[{"0.58815,0.0532,0.62391,2.46,0.479,1.619],1.2,0.25,0,0},"id":1}
03223 *
03224 * @par JSON-RPC
03225 * {"id":1,"jsonrpc":"2.0","result":0}
03226 *
03227 * \endchinese
03228 * \english
03229 * Add linear motion
03230 *
03231 * @param pose Target pose
03232 * @param a Acceleration (if the position change is less than 1mm and the
03233 * posture change is greater than 1e-4 rad, this acceleration is treated as
03234 * angular acceleration in rad/s^2; otherwise, as linear acceleration in
03235 * m/s^2). The maximum value can be obtained via
03236 * RobotConfig::getTcpMaxAccelerations().
03237 * @param v Velocity (if the position change is less than 1mm and the
03238 * posture change is greater than 1e-4 rad, this velocity is treated as
03239 * angular velocity in rad/s; otherwise, as linear velocity in m/s). The
03240 * maximum value can be obtained via RobotConfig::getTcpMaxSpeeds().
03241 * @param blend_radius Blend radius, unit: m, value between 0.001 and 1
03242 * @param duration Execution time, unit: s \n
03243 * Normally, when speed and acceleration are given, the trajectory duration
03244 * can be determined. If you want to extend the trajectory duration, set the
03245 * duration parameter. Duration can extend the trajectory time, but cannot
03246 * shorten it.\n When duration = 0, it means the execution time is not
03247 * specified, and the time will be calculated based on speed and
03248 * acceleration. If duration is not 0, a and v values will be ignored.
03249 * @retval 0 Success
03250 * @retval AUBO_BAD_STATE(1) The current safety mode is not Normal,
03251 * ReducedMode, or Recovery
03252 * @retval AUBO_QUEUE_FULL(2) Trajectory queue is full
03253 * @retval AUBO_BUSY(3) The previous instruction is being executed
03254 * @retval -AUBO_BAD_STATE(-1) Possible reasons include but are not limited
03255 * to: thread has been detached, thread terminated, task_id not found, or
03256 * the current robot mode is not Running
03257 * @retval -AUBO_TIMEOUT(-4) Interface call timeout
03258 * @retval -AUBO_INVL_ARGUMENT(-5) The length of the pose array is less than
03259 * the degrees of freedom of the current robot arm
03260 * @retval AUBO_REQUEST_IGNORE(13) The length of the pose array is too short
03261 * and there is no need to stop at that point
03262 *
03263 * @throws arcs::common_interface::AuboException
03264 *

```

```

03265 * @par Python function prototype
03266 * moveLine(self: pyaubo_sdk.MotionControl, arg0: List[float], arg1: float,
03267 * arg2: float, arg3: float, arg4: float) -> int
03268 *
03269 * @par Lua function prototype
03270 * moveLine(pose: table, a: number, v: number, blend_radius: number,
03271 * duration: number) -> nil
03272 *
03273 * @par Lua example
03274 * moveLine({0.58815,0.0532,0.62391,2.46,0.479,1.619},1.3,1.0,0)
03275 *
03276 * @par JSON-RPC request example
03277 * {"json-
rpc":"2.0","method":"rob1.MotionControl.moveLine","params":[[0.58815,0.0532,0.62391,2.46,0.479,1.619],1.2,0.25,0,0],"id":1}
03278 *
03279 * @par JSON-RPC response example
03280 * {"id":1,"jsonrpc":"2.0","result":0}
03281 *
03282 * \endenglish
03283 */
03284 int moveLine(const std::vector<double> &pose, double a, double v,
03285             double blend_radius, double duration);
03286
03287 /**
03288 * @ingroup MotionControl
03289 * \chinese
03290 *
03291 *
03292 * @param group_name
03293 * @param pose
03294 * @param a
03295 * @param v
03296 * @param blend_radius
03297 * @param duration
03298 * @return
03299 * \endchinese
03300 * \english
03301 * Linear motion synchronized with external axes
03302 *
03303 * @param group_name Name of the external axis group
03304 * @param pose Target pose
03305 * @param a Acceleration
03306 * @param v Velocity
03307 * @param blend_radius Blend radius
03308 * @param duration Execution time
03309 * @return
03310 * \endenglish
03311 */
03312 int moveLineWithAxisGroup(const std::vector<double> &pose, double a,
03313                          double v, double blend_radius, double duration,
03314                          const std::string &group_name,
03315                          const std::vector<double> &extq);
03316
03317 /**
03318 * @ingroup MotionControl
03319 * \chinese
03320 *
03321 *
03322 * @param pose
03323 * @param a
03324 * @param v
03325 * @param blend_radius
03326 * @return 0
03327 * AUBO_BAD_STATE
03328 * AUBO_BUSY
03329 * -AUBO_INVL_ARGUMENT
03330 * -AUBO_BAD_STATE
03331 *
03332 * @throws arcs::common_interface::AuboException
03333 *
03334 * @par Lua
03335 * moveProcess(pose: table, a: number, v: number, blend_radius: number,
03336 * duration: number) -> nil
03337 *
03338 * @par Lua
03339 * moveProcess({0.58815,0.0532,0.62391,2.46,0.479,1.619},1.2,0.25,0)
03340 *
03341 * @par JSON-RPC
03342 * {"json-
rpc":"2.0","method":"rob1.MotionControl.moveProcess","params":[[0.58815,0.0532,0.62391,2.46,0.479,1.619],1.2,0.25,0],"id":1}
03343 *
03344 * @par JSON-RPC
03345 * {"id":1,"jsonrpc":"2.0","result":0}
03346 * \endchinese
03347 * \english
03348 * Add process motion
03349 *

```

```

03350 * @param pose
03351 * @param a
03352 * @param v
03353 * @param blend_radius
03354 * @return Returns 0 on success; otherwise returns an error code:
03355 * AUBO_BAD_STATE
03356 * AUBO_BUSY
03357 * -AUBO_INVL_ARGUMENT
03358 * -AUBO_BAD_STATE
03359 *
03360 * @throws arcs::common_interface::AuboException
03361 *
03362 * @par Lua function prototype
03363 * moveProcess(pose: table, a: number, v: number, blend_radius: number,
03364 * duration: number) -> nil
03365 *
03366 * @par Lua example
03367 * moveProcess({0.58815,0.0532,0.62391,2.46,0.479,1.619},1.2,0.25,0)
03368 *
03369 * @par JSON-RPC request example
03370 * {"json-
rpc":"2.0","method":"rob1.MotionControl.moveProcess","params":[[0.58815,0.0532,0.62391,2.46,0.479,1.619],1.2,0.25,0],"id":1}
03371 *
03372 * @par JSON-RPC response example
03373 * {"id":1,"jsonrpc":"2.0","result":0}
03374 * \endenglish
03375 */
03376 int moveProcess(const std::vector<double> &pose, double a, double v,
03377                 double blend_radius);
03378
03379 /**
03380 * @ingroup MotionControl
03381 * \chinese
03382 *
03383 *
03384 * @param pose
03385 * @param a , m/s^2
03386 * @param v , m/s
03387 * @param duration s \n
03388 *
03389 * duration
03390 * duration \n
03391 * duration = 0
03392 *
03393 * duration 0 a v
03394 * @return 0
03395 * -AUBO_INVL_ARGUMENT
03396 * -AUBO_BAD_STATE
03397 *
03398 * @throws arcs::common_interface::AuboException
03399 *
03400 * @par Python
03401 * resumeMoveLine(self: pyaubo_sdk.MotionControl, arg0: List[float], arg1:
03402 * float, arg2: float, arg3: float) -> int
03403 *
03404 * @par Lua
03405 * resumeMoveLine(pose: table, a: number, v: number,duration: number) -> nil
03406 *
03407 * @par Lua
03408 * resumeMoveLine({0.58815,0.0532,0.62391,2.46,0.479,1.619},1.2,0.25,0)
03409 *
03410 * @par JSON-RPC
03411 * {"json-
rpc":"2.0","method":"rob1.MotionControl.resumeMoveLine","params":[[0.58815,0.0532,0.62391,2.46,0.479,1.619],1.2,0.25,0],"id":1}
03412 *
03413 * @par JSON-RPC
03414 * {"id":1,"jsonrpc":"2.0","result":0}
03415 *
03416 * \endchinese
03417 * \english
03418 * Move to the pause point using linear motion.
03419 *
03420 * @param pose Target pose
03421 * @param a Acceleration, unit: m/s^2
03422 * @param v Velocity, unit: m/s
03423 * @param duration Execution time, unit: s \n
03424 * Normally, when speed and acceleration are given, the trajectory duration
03425 * can be determined. If you want to extend the trajectory duration, set the
03426 * duration parameter. Duration can extend the trajectory time, but cannot
03427 * shorten it.\n When duration = 0, it means the execution time is not
03428 * specified, and the time will be calculated based on speed and
03429 * acceleration. If duration is not 0, a and v values will be ignored.
03430 * @return Returns 0 on success; otherwise returns error code
03431 * -AUBO_INVL_ARGUMENT
03432 * -AUBO_BAD_STATE
03433 *
03434 * @throws arcs::common_interface::AuboException

```

```

03435  *
03436  * @par Python function prototype
03437  * resumeMoveLine(self: pyaubo_sdk.MotionControl, arg0: List[float], arg1:
03438  * float, arg2: float, arg3: float) -> int
03439  *
03440  * @par Lua function prototype
03441  * resumeMoveLine(pose: table, a: number, v: number, duration: number) ->
03442  * nil
03443  *
03444  * @par Lua example
03445  * resumeMoveLine({0.58815,0.0532,0.62391,2.46,0.479,1.619},1.2,0.25,0)
03446  *
03447  * @par JSON-RPC request example
03448  * {"json-
rpc":"2.0","method":"rob1.MotionControl.resumeMoveLine","params":[[0.58815,0.0532,0.62391,2.46,0.479,1.619],1.2,0.25,0],"id":1}
03449  *
03450  * @par JSON-RPC response example
03451  * {"id":1,"jsonrpc":"2.0","result":0}
03452  *
03453  * \endenglish
03454  */
03455  int resumeMoveLine(const std::vector<double> &pose, double a, double v,
03456                    double duration);
03457
03458  /**
03459  * @ingroup MotionControl
03460  * \chinese
03461  *
03462  *
03463  * @todo
03464  *
03465  * @param via_pose
03466  * @param end_pose
03467  * @param a ( via_pose      1mm,      1e-4
03468  * rad,      , rad/s^2.      , m/s^2)
03469  * @param v ( via_pose      1mm,      1e-4
03470  * rad,      , rad / s.      , m / s)
03471  * @param blend_radius : m
03472  * @param duration : s \n
03473  *
03474  *      duration
03475  * duration      \n
03476  * duration = 0
03477  *
03478  * duration 0 a v
03479  * @return 0
03480  * AUBO_BAD_STATE
03481  * AUBO_BUSY
03482  * -AUBO_INVL_ARGUMENT
03483  * -AUBO_BAD_STATE
03484  *
03485  * @throws arcs::common_interface::AuboException
03486  *
03487  * @par Python
03488  * moveCircle(self: pyaubo_sdk.MotionControl, arg0: List[float], arg1:
03489  * List[float], arg2: float, arg3: float, arg4: float, arg5: float) -> int
03490  *
03491  * @par Lua
03492  * moveCircle(via_pose: table, end_pose: table, a: number, v: number,
03493  * blend_radius: number, duration: number) -> nil
03494  *
03495  * @par Lua
03496  *
03497  * moveCircle({0.440164,-0.00249391,0.398658,2.45651,0,1.5708},{0.440164,0.166256,0.297599,2.45651,0,1.5708},1.2,0.25,0)
03498  *
03499  * @par JSON-RPC
03500  * {"jsonrpc":"2.0","method":"rob1.MotionControl.moveCircle","params":[[0.440164,-
0.00249391,0.398658,2.45651,0,1.5708],[0.440164,0.166256,0.297599,2.45651,0,1.5708],1.2,0.25,0,0],"id":1}
03501  *
03502  * @par JSON-RPC
03503  * {"id":1,"jsonrpc":"2.0","result":0}
03504  * \endchinese
03505  * \english
03506  * Add circular arc motion
03507  *
03508  * @todo Can be composed of multiple arcs to form a circular motion
03509  *
03510  * @param via_pose The pose of the via point during the arc motion
03511  * @param end_pose The pose of the end point of the arc motion
03512  * @param a Acceleration (if the position change between via_pose and the
03513  * previous waypoint is less than 1mm and the posture change is greater than
03514  * 1e-4 rad, this acceleration is treated as angular acceleration in
03515  * rad/s^2; otherwise, as linear acceleration in m/s^2)
03516  * @param v Velocity (if the position change between via_pose and the
03517  * previous waypoint is less than 1mm and the posture change is greater than
03518  * 1e-4 rad, this velocity is treated as angular velocity in rad/s;
03519  * otherwise, as linear velocity in m/s)

```



```

03519 * @param blend_radius Blend radius, unit: m
03520 * @param duration Execution time, unit: s \n
03521 * Normally, when speed and acceleration are given, the trajectory duration
03522 * can be determined. If you want to extend the trajectory duration, set the
03523 * duration parameter. Duration can extend the trajectory time, but cannot
03524 * shorten it.\n When duration = 0, it means the execution time is not
03525 * specified, and the time will be calculated based on speed and
03526 * acceleration. If duration is not 0, a and v values will be ignored.
03527 * @return Returns 0 on success; otherwise returns error code
03528 * AUBO_BAD_STATE
03529 * AUBO_BUSY
03530 * -AUBO_INVL_ARGUMENT
03531 * -AUBO_BAD_STATE
03532 *
03533 * @throws arcs::common_interface::AuboException
03534 *
03535 * @par Python function prototype
03536 * moveCircle(self: pyaubo_sdk.MotionControl, arg0: List[float], arg1:
03537 * List[float], arg2: float, arg3: float, arg4: float, arg5: float) -> int
03538 *
03539 * @par Lua function prototype
03540 * moveCircle(via_pose: table, end_pose: table, a: number, v: number,
03541 * blend_radius: number, duration: number) -> nil
03542 *
03543 * @par Lua example
03544 *
moveCircle({0.440164,-0.00249391,0.398658,2.45651,0,1.5708},{0.440164,0.166256,0.297599,2.45651,0,1.5708},1.2,0.25,0)
03545 *
03546 * @par JSON-RPC request example
03547 * {"jsonrpc":"2.0","method":"rob1.MotionControl.moveCircle","params":[[0.440164,-
0.00249391,0.398658,2.45651,0,1.5708],[0.440164,0.166256,0.297599,2.45651,0,1.5708],1.2,0.25,0,0],"id":1}
03548 *
03549 * @par JSON-RPC response example
03550 * {"id":1,"jsonrpc":"2.0","result":0}
03551 * \endenglish
03552 */
03553 int moveCircle(const std::vector<double> &via_pose,
03554               const std::vector<double> &end_pose, double a, double v,
03555               double blend_radius, double duration);
03556
03557 /**
03558 * @ingroup MotionControl
03559 * \chinese
03560 * moveCircle
03561 *
03562 * @param group_name
03563 * @param via_pose
03564 * @param end_pose
03565 * @param a
03566 * @param v
03567 * @param blend_radius
03568 * @param duration
03569 * @return
03570 * \endchinese
03571 * \english
03572 * moveCircle with external axes synchronized motion
03573 *
03574 * @param group_name Name of the external axis group
03575 * @param via_pose The pose of the via point during the arc motion
03576 * @param end_pose The pose of the end point of the arc motion
03577 * @param a Acceleration
03578 * @param v Velocity
03579 * @param blend_radius Blend radius
03580 * @param duration Execution time
03581 * @return
03582 * \endenglish
03583 */
03584 int moveCircleWithAxisGroup(const std::vector<double> &via_pose,
03585                             const std::vector<double> &end_pose, double a,
03586                             double v, double blend_radius, double duration,
03587                             const std::string &group_name,
03588                             const std::vector<double> &extq);
03589
03590 /**
03591 * @ingroup MotionControl
03592 * \chinese
03593 *
03594 *
03595 * @param mode \n
03596 * 0: \n
03597 * 1: , , \n
03598 * 2: ,
03599 * @return 0
03600 * AUBO_BAD_STATE
03601 * AUBO_BUSY
03602 * -AUBO_INVL_ARGUMENT
03603 * -AUBO_BAD_STATE

```



```

03604 *
03605 * @throws arcs::common_interface::AuboException
03606 *
03607 * @par Python
03608 * setCirclePathMode(self: pyaubo_sdk.MotionControl, arg0: int) -> int
03609 *
03610 * @par Lua
03611 * setCirclePathMode(mode: number) -> nil
03612 *
03613 * @par Lua
03614 * setCirclePathMode(1)
03615 *
03616 * @par JSON-RPC
03617 * {"jsonrpc": "2.0", "method": "rob1.MotionControl.setCirclePathMode", "params": [0], "id": 1}
03618 *
03619 * @par JSON-RPC
03620 * {"id": 1, "jsonrpc": "2.0", "result": 0}
03621 *
03622 * \endchinese
03623 * \english
03624 * Set circle path mode
03625 *
03626 * @param mode Mode \n
03627 * 0: Tool orientation remains unchanged relative to the arc path point
03628 * coordinate system \n 1: Orientation changes linearly, rotating around a
03629 * fixed axis in space, from the start orientation to the target orientation
03630 * \n 2: Orientation changes from the start orientation, through the via
03631 * point orientation, to the target orientation
03632 * @return Returns 0 on success; otherwise returns an error code
03633 * AUBO_BAD_STATE
03634 * AUBO_BUSY
03635 * -AUBO_INVL_ARGUMENT
03636 * -AUBO_BAD_STATE
03637 *
03638 * @throws arcs::common_interface::AuboException
03639 *
03640 * @par Python function prototype
03641 * setCirclePathMode(self: pyaubo_sdk.MotionControl, arg0: int) -> int
03642 *
03643 * @par Lua function prototype
03644 * setCirclePathMode(mode: number) -> nil
03645 *
03646 * @par Lua example
03647 * setCirclePathMode(1)
03648 *
03649 * @par JSON-RPC request example
03650 * {"jsonrpc": "2.0", "method": "rob1.MotionControl.setCirclePathMode", "params": [0], "id": 1}
03651 *
03652 * @par JSON-RPC response example
03653 * {"id": 1, "jsonrpc": "2.0", "result": 0}
03654 *
03655 * \endenglish
03656 */
03657 int setCirclePathMode(int mode);
03658
03659 /**
03660 * @ingroup MotionControl
03661 * \chinese
03662 *
03663 *
03664 * @param param
03665 * @return 0
03666 * AUBO_BAD_STATE
03667 * AUBO_BUSY
03668 * -AUBO_INVL_ARGUMENT
03669 * -AUBO_BAD_STATE
03670 *
03671 * @throws arcs::common_interface::AuboException
03672 *
03673 * @par Python
03674 * moveCircle2(self: pyaubo_sdk.MotionControl, arg0:
03675 * arcs::common_interface::CircleParameters) -> int
03676 *
03677 * @par Lua
03678 * moveCircle2(param: table) -> nil
03679 *
03680 * @par Lua
03681 * moveCircle2({0.440164,-
0.00249391,0.398658,2.45651,0,1.570},{0.440164,0.166256,0.297599,2.45651,0,1.5708},1.2,0.25,0,0,0,0,0)
03682 *
03683 * @par JSON-RPC
03684 * {"jsonrpc": "2.0", "method": "rob1.MotionControl.moveCircle2", "params": [{"pose_via": [0.440164,-
0.00249391,0.398658,2.45651,0,1.570],
03685 * "pose_to": [0.440164,0.166256,0.297599,2.45651,0,1.5708], "a": 1.2, "v": 0.25, "blend_radius": 0, "duration": 0, "helix": 0,
03686 * "spiral": 0, "direction": 0, "loop_times": 0}], "id": 1}
03687 *
03688 * @par JSON-RPC

```

```

03689 * {"id":1,"jsonrpc":"2.0","result":0}
03690 * \endchinese
03691 * \english
03692 * Advanced arc or circular motion
03693 *
03694 * @param param Circular motion parameters
03695 * @return Returns 0 on success; otherwise returns an error code:
03696 * AUBO_BAD_STATE
03697 * AUBO_BUSY
03698 * -AUBO_INVL_ARGUMENT
03699 * -AUBO_BAD_STATE
03700 *
03701 * @throws arcs::common_interface::AuboException
03702 *
03703 * @par Python function prototype
03704 * moveCircle2(self: pyaubo_sdk.MotionControl, arg0:
03705 * arcs::common_interface::CircleParameters) -> int
03706 *
03707 * @par Lua function prototype
03708 * moveCircle2(param: table) -> nil
03709 *
03710 * @par Lua example
03711 * moveCircle2({0.440164,-
03712 * 0.00249391,0.398658,2.45651,0,1.570},{0.440164,0.166256,0.297599,2.45651,0,1.5708},1.2,0.25,0,0,0,0,0)
03713 *
03714 * @par JSON-RPC request example
03715 * {"jsonrpc":"2.0","method":"rob1.MotionControl.moveCircle2","params":[{"pose_via":[0.440164,-
03716 * 0.00249391,0.398658,2.45651,0,1.5708],
03717 * "pose_to":[0.440164,0.166256,0.297599,2.45651,0,1.5708],"a":1.2,"v":0.25,"blend_radius":0,"duration":0,"helix":0,
03718 * "spiral":0,"direction":0,"loop_times":0}], "id":1}
03719 *
03720 * @par JSON-RPC response example
03721 * {"id":1,"jsonrpc":"2.0","result":0}
03722 * \endenglish
03723 * /
03724 int moveCircle2(const CircleParameters &param);
03725 /**
03726 * @ingroup MotionControl
03727 * \chinese
03728 *
03729 * @param name
03730 * @param type \n
03731 * 1: toppra \n
03732 * 2: cubic_spline( ) \n
03733 * 3: B
03734 * @param size
03735 * @return AUBO_OK(0)
03736 *
03737 * @throws arcs::common_interface::AuboException
03738 *
03739 * @par Python
03740 * pathBufferAlloc(self: pyaubo_sdk.MotionControl, arg0: str, arg1: int,
03741 * arg2: int) -> int
03742 *
03743 * @par Lua
03744 * pathBufferAlloc(name: string, type: number, size: number) -> nil
03745 *
03746 * @par Lua
03747 * pathBufferAlloc("traje_name",5,100)
03748 *
03749 * @par JSON-RPC
03750 * {"jsonrpc":"2.0","method":"rob1.MotionControl.pathBufferAlloc","params":["rec",2,3],"id":1}
03751 *
03752 * @par JSON-RPC
03753 * {"id":1,"jsonrpc":"2.0","result":0}
03754 *
03755 * \endchinese
03756 * \english
03757 * Create a new path point buffer
03758 *
03759 * @param name Name of the path
03760 * @param type Type of the path \n
03761 * 1: toppra time-optimal path planning \n
03762 * 2: cubic_spline (recorded trajectory) \n
03763 * 3: Joint B-spline interpolation, at least three points
03764 * @param size Buffer size
03765 * @return Returns AUBO_OK(0) on success
03766 *
03767 * @throws arcs::common_interface::AuboException
03768 *
03769 * @par Python function prototype
03770 * pathBufferAlloc(self: pyaubo_sdk.MotionControl, arg0: str, arg1: int,
03771 * arg2: int) -> int
03772 *
03773 * @par Lua function prototype

```

```

03774 * pathBufferAlloc(name: string, type: number, size: number) -> nil
03775 *
03776 * @par Lua exmaple
03777 * pathBufferAlloc("traje_name",5,100)
03778 *
03779 * @par JSON-RPC request example
03780 * {"jsonrpc":"2.0","method":"rob1.MotionControl.pathBufferAlloc","params":["rec",2,3],"id":1}
03781 *
03782 * @par JSON-RPC response example
03783 * {"id":1,"jsonrpc":"2.0","result":0}
03784 *
03785 * \endenglish
03786 */
03787 int pathBufferAlloc(const std::string &name, int type, int size);
03788
03789 /**
03790 * @ingroup MotionControl
03791 * \chinese
03792 *
03793 *
03794 * @param name
03795 * @param waypoints
03796 * @return 0
03797 * -AUBO_INVL_ARGUMENT
03798 * -AUBO_BAD_STATE
03799 *
03800 * @throws arcs::common_interface::AuboException
03801 *
03802 * @par Python
03803 * pathBufferAppend(self: pyaubo_sdk.MotionControl, arg0: str, arg1:
03804 * List[List[float]]) -> int
03805 *
03806 * @par Lua
03807 * pathBufferAppend(name: string, waypoints: table) -> nil
03808 *
03809 * @par Lua
03810 * pathBufferAppend("traje_name",{ {-0.000000,0.000009,-0.000001,0.000002,0.000002,0.000000},{-0.000001,0.000010,-
0.000002,0.000000,0.000003,0.000002}})
03811 *
03812 * @par JSON-RPC
03813 * {"jsonrpc":"2.0","method":"rob1.MotionControl.pathBufferAppend","params":["rec",[{-0.000000,0.000009,-
0.000001,0.000002,0.000000,0.000000],[-0.000001,0.000010,-0.000002,0.000000,0.000003,0.000002]]],"id":1}
03814 *
03815 * @par JSON-RPC
03816 * {"id":1,"jsonrpc":"2.0","result":0}
03817 * \endchinese
03818 * \english
03819 * Add waypoints to the path buffer
03820 *
03821 * @param name Name of the path buffer
03822 * @param waypoints Waypoints
03823 * @return Returns 0 on success; otherwise returns an error code
03824 * -AUBO_INVL_ARGUMENT
03825 * -AUBO_BAD_STATE
03826 *
03827 * @throws arcs::common_interface::AuboException
03828 *
03829 * @par Python function prototype
03830 * pathBufferAppend(self: pyaubo_sdk.MotionControl, arg0: str, arg1:
03831 * List[List[float]]) -> int
03832 *
03833 * @par Lua function prototype
03834 * pathBufferAppend(name: string, waypoints: table) -> nil
03835 *
03836 * @par Lua example
03837 * pathBufferAppend("traje_name",{ {-0.000000,0.000009,-0.000001,0.000002,0.000002,0.000000},{-0.000001,0.000010,-
0.000002,0.000000,0.000003,0.000002}})
03838 *
03839 * @par JSON-RPC request example
03840 * {"jsonrpc":"2.0","method":"rob1.MotionControl.pathBufferAppend","params":["rec",[{-0.000000,0.000009,-
0.000001,0.000002,0.000002,0.000000],[-0.000001,0.000010,-0.000002,0.000000,0.000003,0.000002]]],"id":1}
03841 *
03842 * @par JSON-RPC response example
03843 * {"id":1,"jsonrpc":"2.0","result":0}
03844 * \endenglish
03845 */
03846 int pathBufferAppend(const std::string &name,
03847                     const std::vector<std::vector<double> &waypoints);
03848
03849 /**
03850 * @ingroup MotionControl
03851 * \chinese
03852 *
03853 *
03854 * @param name pathBufferAlloc
03855 * @param a , , rad/s^2
03856 * @param v , , rad/s

```

```

03857 * @param t \n
03858 * pathBufferAlloc
03859 * pathBufferAlloc : \n
03860 * 1, \n
03861 * 2, \n
03862 * 3:t \n
03863 * t=0, ( ) \n
03864 * t!=0, t * (points *
03865 * interval,interval >= 0.01)
03866 * @return 0
03867 * -AUBO_INVL_ARGUMENT
03868 * -AUBO_BAD_STATE
03869 *
03870 * @throws arcs::common_interface::AuboException
03871 *
03872 * @par Python
03873 * pathBufferEval(self: pyaubo_sdk.MotionControl, arg0: str, arg1:
03874 * List[float], arg2: List[float], arg3: float) -> int
03875 *
03876 * @par Lua
03877 * pathBufferEval(name: string, a: table, v: table, t: number) -> nil
03878 *
03879 * @par Lua
03880 * pathBufferEval("traje_name",{1,1,1,1,1,1},{1,1,1,1,1,1},0.02)
03881 *
03882 * @par JSON-RPC
03883 * {"jsonrpc":"2.0","method":"rob1.MotionControl.pathBufferEval","params":["rec",[1,1,1,1,1,1],[1,1,1,1,1,1],0.02],"id":1}
03884 *
03885 * @par JSON-RPC
03886 * {"id":1,"jsonrpc":"2.0","result":0}
03887 * \endchinese
03888 * \english
03889 * Perform computation and optimization (time-consuming operations). If the
03890 * input parameters are the same, the calculation will not be repeated.
03891 *
03892 * @param name Name of the path point buffer created by pathBufferAlloc
03893 * @param a Joint acceleration limits, must match the robot DOF, unit:
03894 * rad/s^2
03895 * @param v Joint velocity limits, must match the robot DOF, unit: rad/s
03896 * @param t Time \n
03897 * When allocating memory with pathBufferAlloc, the type must be specified.
03898 * According to the type in pathBufferAlloc:\n
03899 * Type 1: t means motion duration\n
03900 * Type 2: t means sampling interval\n
03901 * Type 3: t means motion duration\n
03902 * If t=0, the internal default time is used (recommended)\n
03903 * If t!=0, t should be set to number_of_points *
03904 * interval_between_points (interval >= 0.01)
03905 * @return Returns 0 on success; otherwise returns an error code
03906 * -AUBO_INVL_ARGUMENT
03907 * -AUBO_BAD_STATE
03908 *
03909 * @throws arcs::common_interface::AuboException
03910 *
03911 * @par Python function prototype
03912 * pathBufferEval(self: pyaubo_sdk.MotionControl, arg0: str, arg1:
03913 * List[float], arg2: List[float], arg3: float) -> int
03914 *
03915 * @par Lua function prototype
03916 * pathBufferEval(name: string, a: table, v: table, t: number) -> nil
03917 *
03918 * @par Lua example
03919 * pathBufferEval("traje_name",{1,1,1,1,1,1},{1,1,1,1,1,1},0.02)
03920 *
03921 * @par JSON-RPC request example
03922 * {"jsonrpc":"2.0","method":"rob1.MotionControl.pathBufferEval","params":["rec",[1,1,1,1,1,1],[1,1,1,1,1,1],0.02],"id":1}
03923 *
03924 * @par JSON-RPC response example
03925 * {"id":1,"jsonrpc":"2.0","result":0}
03926 * \endenglish
03927 */
03928 int pathBufferEval(const std::string &name, const std::vector<double> &a,
03929 const std::vector<double> &v, double t);
03930
03931 /**
03932 * @ingroup MotionControl
03933 * \chinese
03934 * buffer
03935 *
03936 * buffer : \n
03937 * 1 buffer \n
03938 * 2 buffer buffer \n
03939 * 3 pathBufferEval
03940 *
03941 * @param name buffer
03942 * @return true; false
03943

```

```

03944 * @throws arcs::common_interface::AuboException
03945 *
03946 * @par Python
03947 * pathBufferValid(self: pyaubo_sdk.MotionControl, arg0: str) -> bool
03948 *
03949 * @par Lua
03950 * pathBufferValid(name: string) -> boolean
03951 *
03952 * @par Lua
03953 * buffer_status = pathBufferValid("traje_name")
03954 *
03955 * @par JSON-RPC
03956 * {"jsonrpc":"2.0","method":"rob1.MotionControl.pathBufferValid","params":["rec"],"id":1}
03957 *
03958 * @par JSON-RPC
03959 * {"id":1,"jsonrpc":"2.0","result":false}
03960 * \endchinese
03961 * \english
03962 * Whether the buffer with the specified name is valid
03963 *
03964 * The buffer must meet three conditions to be valid: \n
03965 * 1. The buffer exists and memory has been allocated \n
03966 * 2. The number of points passed into the buffer must be greater than or
03967 * equal to the buffer size \n
03968 * 3. pathBufferEval must be executed once
03969 *
03970 * @param name Name of the buffer
03971 * @return Returns true if valid; false otherwise
03972 *
03973 * @throws arcs::common_interface::AuboException
03974 *
03975 * @par Python function prototype
03976 * pathBufferValid(self: pyaubo_sdk.MotionControl, arg0: str) -> bool
03977 *
03978 * @par Lua function prototype
03979 * pathBufferValid(name: string) -> boolean
03980 *
03981 * @par Lua example
03982 * buffer_status = pathBufferValid("traje_name")
03983 *
03984 * @par JSON-RPC request example
03985 * {"jsonrpc":"2.0","method":"rob1.MotionControl.pathBufferValid","params":["rec"],"id":1}
03986 *
03987 * @par JSON-RPC response example
03988 * {"id":1,"jsonrpc":"2.0","result":false}
03989 * \endenglish
03990 */
03991 bool pathBufferValid(const std::string &name);
03992
03993 /**
03994 * @ingroup MotionControl
03995 * \chinese
03996 *
03997 *
03998 * @param name
03999 * @return 0
04000 * -AUBO_INVL_ARGUMENT
04001 * -AUBO_BAD_STATE
04002 *
04003 * @throws arcs::common_interface::AuboException
04004 *
04005 * @par Python
04006 * pathBufferFree(self: pyaubo_sdk.MotionControl, arg0: str) -> int
04007 *
04008 * @par Lua
04009 * pathBufferFree(name: string) -> nil
04010 *
04011 * @par Lua
04012 * pathBufferFree("traje_name")
04013 *
04014 * @par JSON-RPC
04015 * {"jsonrpc":"2.0","method":"rob1.MotionControl.pathBufferFree","params":["rec"],"id":1}
04016 *
04017 * @par JSON-RPC
04018 * {"id":1,"jsonrpc":"2.0","result":-5}
04019 *
04020 * \endchinese
04021 * \english
04022 * Release path buffer
04023 *
04024 * @param name Name of the path buffer
04025 * @return Returns 0 on success; otherwise returns an error code
04026 * -AUBO_INVL_ARGUMENT
04027 * -AUBO_BAD_STATE
04028 *
04029 * @throws arcs::common_interface::AuboException
04030 *

```

```

04031 * @par Python function prototype
04032 * pathBufferFree(self: pyaubo_sdk.MotionControl, arg0: str) -> int
04033 *
04034 * @par Lua function prototype
04035 * pathBufferFree(name: string) -> nil
04036 *
04037 * @par Lua example
04038 * pathBufferFree("traje_name")
04039 *
04040 * @par JSON-RPC request example
04041 * {"jsonrpc":"2.0","method":"rob1.MotionControl.pathBufferFree","params":["rec"],"id":1}
04042 *
04043 * @par JSON-RPC response example
04044 * {"id":1,"jsonrpc":"2.0","result":-5}
04045 *
04046 * \endenglish
04047 */
04048 int pathBufferFree(const std::string &name);
04049
04050 /**
04051 * @ingroup MotionControl
04052 * \chinese
04053 *
04054 *
04055 * @brief pathBufferFilter
04056 *
04057 * @param name
04058 * @param order ( 2)
04059 * @param fd ( 3-20)
04060 * @param fs ( 20-500)
04061 *
04062 * @return 0
04063 *
04064 * @throws arcs::common_interface::AuboException
04065 *
04066 * @par Python
04067 * pathBufferFilter(self: pyaubo_sdk.MotionControl, arg0: str, arg1: int,
04068 * arg2: float, arg3: float) -> int
04069 *
04070 * @par Lua
04071 * pathBufferFilter(name: string, order: number, fd: number, fs:number) ->
04072 * nil
04073 *
04074 * @par Lua
04075 * pathBufferFilter("traje_name",2,5,125)
04076 *
04077 * \endchinese
04078 * \english
04079 * Joint space path filter
04080 *
04081 * @brief pathBufferFilter
04082 *
04083 * @param name Name of the path buffer
04084 * @param order Filter order (usually 2)
04085 * @param fd Cutoff frequency, the smaller the smoother but with more delay
04086 * (usually 3-20)
04087 * @param fs Sampling frequency of discrete data (usually 20-500)
04088 *
04089 * @return Returns 0 on success
04090 *
04091 * @throws arcs::common_interface::AuboException
04092 *
04093 * @par Python function prototype
04094 * pathBufferFilter(self: pyaubo_sdk.MotionControl, arg0: str, arg1: int,
04095 * arg2: float, arg3: float) -> int
04096 *
04097 * @par Lua function prototype
04098 * pathBufferFilter(name: string, order: number, fd: number, fs:number) ->
04099 * nil
04100 *
04101 * @par Lua example
04102 * pathBufferFilter("traje_name",2,5,125)
04103 *
04104 * \endenglish
04105 */
04106 int pathBufferFilter(const std::string &name, int order, double fd,
04107 double fs);
04108
04109 /**
04110 * @ingroup MotionControl
04111 * \chinese
04112 *
04113 *
04114 * @return
04115 *
04116 * @throws arcs::common_interface::AuboException
04117 *

```

```

04118 * @par Python
04119 * pathBufferList(self: pyaubo_sdk.MotionControl) -> List[str]
04120 *
04121 * @par Lua
04122 * pathBufferList() -> table
04123 *
04124 * @par Lua
04125 * Buffer_table = pathBufferList()
04126 *
04127 * @par JSON-RPC
04128 * {"jsonrpc": "2.0", "method": "rob1.MotionControl.pathBufferList", "params": [], "id": 1}
04129 *
04130 * @par JSON-RPC
04131 * {"id": 1, "jsonrpc": "2.0", "result": []}
04132 *
04133 * \endchinese
04134 * \english
04135 * List all cached path names
04136 *
04137 * @return Returns all cached path names
04138 *
04139 * @throws arcs::common_interface::AuboException
04140 *
04141 * @par Python function prototype
04142 * pathBufferList(self: pyaubo_sdk.MotionControl) -> List[str]
04143 *
04144 * @par Lua function prototype
04145 * pathBufferList() -> table
04146 *
04147 * @par Lua example
04148 * Buffer_table = pathBufferList()
04149 *
04150 * @par JSON-RPC request example
04151 * {"jsonrpc": "2.0", "method": "rob1.MotionControl.pathBufferList", "params": [], "id": 1}
04152 *
04153 * @par JSON-RPC response example
04154 * {"id": 1, "jsonrpc": "2.0", "result": []}
04155 *
04156 * \endenglish
04157 */
04158 std::vector<std::string> pathBufferList();
04159
04160 /**
04161 * @ingroup MotionControl
04162 * \chinese
04163 *
04164 *
04165 * @param name
04166 * @return 0
04167 * -AUBO_INVL_ARGUMENT
04168 * -AUBO_BAD_STATE
04169 *
04170 * @throws arcs::common_interface::AuboException
04171 *
04172 * @par Python
04173 * movePathBuffer(self: pyaubo_sdk.MotionControl, arg0: str) -> int
04174 *
04175 * @par Lua
04176 * movePathBuffer(name: string) -> nil
04177 *
04178 * @par Lua
04179 * movePathBuffer("traje_name")
04180 *
04181 * @par JSON-RPC
04182 * {"jsonrpc": "2.0", "method": "rob1.MotionControl.movePathBuffer", "params": ["rec"], "id": 1}
04183 *
04184 * @par JSON-RPC
04185 * {"id": 1, "jsonrpc": "2.0", "result": 0}
04186 * \endchinese
04187 * \english
04188 * Execute the cached path
04189 *
04190 * @param name Name of the cached path
04191 * @return Returns 0 on success; otherwise returns an error code
04192 * -AUBO_INVL_ARGUMENT
04193 * -AUBO_BAD_STATE
04194 *
04195 * @throws arcs::common_interface::AuboException
04196 *
04197 * @par Python function prototype
04198 * movePathBuffer(self: pyaubo_sdk.MotionControl, arg0: str) -> int
04199 *
04200 * @par Lua function prototype
04201 * movePathBuffer(name: string) -> nil
04202 *
04203 * @par Lua example
04204 * movePathBuffer("traje_name")

```

```

04205  *
04206  * @par JSON-RPC request example
04207  * {"jsonrpc":"2.0","method":"rob1.MotionControl.movePathBuffer","params":["rec"],"id":1}
04208  *
04209  * @par JSON-RPC response example
04210  * {"id":1,"jsonrpc":"2.0","result":0}
04211  * \endenglish
04212  */
04213 int movePathBuffer(const std::string &name);
04214
04215 /**
04216  * @ingroup MotionControl
04217  * \chinese
04218  *
04219  *
04220  * @param pose ( , :
04221  *              )
04222  * p1: ,
04223  * p2: ,
04224  * p3: ,
04225  * @param v
04226  * @param a
04227  * @param main_pipe_radius
04228  * @param sub_pipe_radius
04229  * @param normal_distance
04230  * @param normal_alpha
04231  *
04232  * @return 0
04233  * AUBO_BUSY
04234  * AUBO_BAD_STATE
04235  * -AUBO_INVL_ARGUMENT
04236  * -AUBO_BAD_STATE
04237  *
04238  * @throws arcs::common_interface::AuboException
04239  *
04240  * @par Python
04241  * moveIntersection(self: pyaubo_sdk.MotionControl, arg0: List[float], arg1:
04242  * float, arg2: float, arg3: float, arg4: float, arg5: float, arg6: float)
04243  * -> int
04244  *
04245  * @par Lua
04246  * moveIntersection(poses: table, a: number, v: number,
04247  * main_pipe_radius: number, sub_pipe_radius: number, normal_distance:
04248  * number, normal_alpha: number) -> nil \endchinese \english Intersection
04249  * interface
04250  *
04251  * @param poses Consists of three taught poses (first, move to the starting
04252  * point. The starting point requirement: the intersection of the plane
04253  * passing through the main cylinder axis and parallel to the sub-cylinder
04254  * axis with the sub-cylinder at the bottom) p1: Intersection of the plane
04255  * passing through the sub-cylinder axis and parallel to the main cylinder
04256  * axis with the sub-cylinder at the left top p2: Intersection of the plane
04257  * passing through the sub-cylinder axis and parallel to the main cylinder
04258  * axis with the main cylinder at the left bottom p3: Intersection of the
04259  * plane passing through the sub-cylinder axis and parallel to the main
04260  * cylinder axis with the main cylinder at the right bottom
04261  * @param v Velocity
04262  * @param a Acceleration
04263  * @param main_pipe_radius Main cylinder radius
04264  * @param sub_pipe_radius Sub cylinder radius
04265  * @param normal_distance Distance between the two cylinder axes
04266  * @param normal_alpha Angle between the two cylinder axes
04267  *
04268  * @return Returns 0 on success; otherwise returns an error code
04269  * AUBO_BUSY
04270  * AUBO_BAD_STATE
04271  * -AUBO_INVL_ARGUMENT
04272  * -AUBO_BAD_STATE
04273  *
04274  * @throws arcs::common_interface::AuboException
04275  *
04276  * @par Python function prototype
04277  * moveIntersection(self: pyaubo_sdk.MotionControl, arg0: List[float], arg1:
04278  * float, arg2: float, arg3: float, arg4: float, arg5: float, arg6: float)
04279  * -> int
04280  *
04281  * @par Lua function prototype
04282  * moveIntersection(poses: table, a: number, v: number,
04283  * main_pipe_radius: number, sub_pipe_radius: number, normal_distance:
04284  * number, normal_alpha: number) -> nil \endenglish
04285  */
04286 int moveIntersection(const std::vector<std::vector<double> &poses,
04287 double a, double v, double main_pipe_radius,
04288 double sub_pipe_radius, double normal_distance,
04289 double normal_alpha);
04290
04291 /**
04292  * @ingroup MotionControl

```



```

04292 * \chinese
04293 *
04294 *
04295 * @param acc      : rad/s^2
04296 * @return 0
04297 * AUBO_BUSY
04298 * AUBO_BAD_STATE
04299 * -AUBO_INVL_ARGUMENT
04300 * -AUBO_BAD_STATE
04301 *
04302 * @throws arcs::common_interface::AuboException
04303 *
04304 * @par Python
04305 * stopJoint(self: pyaubo_sdk.MotionControl, arg0: float) -> int
04306 *
04307 * @par Lua
04308 * stopJoint(acc: number) -> nil
04309 *
04310 * @par Lua
04311 * stopJoint(2)
04312 *
04313 * @par JSON-RPC
04314 * {"jsonrpc":"2.0","method":"rob1.MotionControl.stopJoint","params":[31],"id":1}
04315 *
04316 * @par JSON-RPC
04317 * {"id":1,"jsonrpc":"2.0","result":0}
04318 *
04319 * \endchinese
04320 * \english
04321 * Stop motion in joint space
04322 *
04323 * @param acc Joint acceleration, unit: rad/s^2
04324 * @return Returns 0 on success; otherwise returns an error code
04325 * AUBO_BUSY
04326 * AUBO_BAD_STATE
04327 * -AUBO_INVL_ARGUMENT
04328 * -AUBO_BAD_STATE
04329 *
04330 * @throws arcs::common_interface::AuboException
04331 *
04332 * @par Python function prototype
04333 * stopJoint(self: pyaubo_sdk.MotionControl, arg0: float) -> int
04334 *
04335 * @par Lua function prototype
04336 * stopJoint(acc: number) -> nil
04337 *
04338 * @par Lua example
04339 * stopJoint(2)
04340 *
04341 * @par JSON-RPC request example
04342 * {"jsonrpc":"2.0","method":"rob1.MotionControl.stopJoint","params":[31],"id":1}
04343 *
04344 * @par JSON-RPC response example
04345 * {"id":1,"jsonrpc":"2.0","result":0}
04346 *
04347 * \endenglish
04348 */
04349 int stopJoint(double acc);
04350
04351 /**
04352 * @ingroup MotionControl
04353 * \chinese
04354 * (          , resumeSpeedJoint          )
04355 *
04356 * @param acc      : rad/s^2
04357 * @return 0
04358 * AUBO_BUSY
04359 * AUBO_BAD_STATE
04360 * -AUBO_INVL_ARGUMENT
04361 * -AUBO_BAD_STATE
04362 *
04363 * @throws arcs::common_interface::AuboException
04364 *
04365 * @par Python
04366 * resumeStopJoint(self: pyaubo_sdk.MotionControl, arg0: float) -> int
04367 *
04368 * @par Lua
04369 * resumeStopJoint(acc: number) -> nil
04370 *
04371 * @par Lua
04372 * resumeStopJoint(31)
04373 *
04374 * @par JSON-RPC
04375 * {"jsonrpc":"2.0","method":"rob1.MotionControl.resumeStopJoint","params":[31],"id":1}
04376 *
04377 * @par JSON-RPC
04378 * {"id":1,"jsonrpc":"2.0","result":-1}

```

```

04379 *
04380 * \endchinese
04381 * \english
04382 * Stop motion in joint space (used after moving to a safe position via
04383 * resumeSpeedJoint following a collision during process execution)
04384 *
04385 * @param acc Joint acceleration, unit: rad/s^2
04386 * @return Returns 0 on success; otherwise returns an error code
04387 * AUBO_BUSY
04388 * AUBO_BAD_STATE
04389 * -AUBO_INVL_ARGUMENT
04390 * -AUBO_BAD_STATE
04391 *
04392 * @throws arcs::common_interface::AuboException
04393 *
04394 * @par Python function prototype
04395 * resumeStopJoint(self: pyaubo_sdk.MotionControl, arg0: float) -> int
04396 *
04397 * @par Lua function prototype
04398 * resumeStopJoint(acc: number) -> nil
04399 *
04400 * @par Lua example
04401 * resumeStopJoint(31)
04402 *
04403 * @par JSON-RPC request example
04404 * {"jsonrpc": "2.0", "method": "rob1.MotionControl.resumeStopJoint", "params": [31], "id": 1}
04405 *
04406 * @par JSON-RPC response example
04407 * {"id": 1, "jsonrpc": "2.0", "result": -1}
04408 *
04409 * \endenglish
04410 */
04411 int resumeStopJoint(double acc);
04412
04413 /**
04414 * @ingroup MotionControl
04415 * \chinese
04416 *   moveLine/moveCircle
04417 *
04418 * @param acc      , : m/s^2
04419 * @param acc_rot
04420 * @return 0
04421 * AUBO_BUSY
04422 * AUBO_BAD_STATE
04423 * -AUBO_INVL_ARGUMENT
04424 * -AUBO_BAD_STATE
04425 *
04426 * @throws arcs::common_interface::AuboException
04427 *
04428 * @par Python
04429 * stopLine(self: pyaubo_sdk.MotionControl, arg0: float, arg1: float) -> int
04430 *
04431 * @par Lua
04432 * stopLine(acc: number, acc_rot: number) -> nil
04433 *
04434 * @par Lua
04435 * stopLine(10,10)
04436 *
04437 * @par JSON-RPC
04438 * {"jsonrpc": "2.0", "method": "rob1.MotionControl.stopLine", "params": [10,10], "id": 1}
04439 *
04440 * @par JSON-RPC
04441 * {"id": 1, "jsonrpc": "2.0", "result": 0}
04442 *
04443 * \endchinese
04444 * \english
04445 * Stop motions in Cartesian space such as moveLine/moveCircle.
04446 *
04447 * @param acc Tool acceleration, unit: m/s^2
04448 * @param acc_rot
04449 * @return Returns 0 on success; otherwise returns an error code
04450 * AUBO_BUSY
04451 * AUBO_BAD_STATE
04452 * -AUBO_INVL_ARGUMENT
04453 * -AUBO_BAD_STATE
04454 *
04455 * @throws arcs::common_interface::AuboException
04456 *
04457 * @par Python function prototype
04458 * stopLine(self: pyaubo_sdk.MotionControl, arg0: float, arg1: float) -> int
04459 *
04460 * @par Lua function prototype
04461 * stopLine(acc: number, acc_rot: number) -> nil
04462 *
04463 * @par Lua example
04464 * stopLine(10,10)
04465 *

```

```

04466 * @par JSON-RPC request example
04467 * {"jsonrpc":"2.0","method":"rob1.MotionControl.stopLine","params":[10,10],"id":1}
04468 *
04469 * @par JSON-RPC response example
04470 * {"id":1,"jsonrpc":"2.0","result":0}
04471 *
04472 * \endenglish
04473 */
04474 int stopLine(double acc, double acc_rot);
04475
04476 /**
04477 * @ingroup MotionControl
04478 * \chinese
04479 * (          , resumeSpeedLine          )
04480 *
04481 * @param acc      : m/s^2
04482 * @param acc_rot  : rad/s^2
04483 * @return 0
04484 * AUBO_BUSY
04485 * AUBO_BAD_STATE
04486 * -AUBO_INVL_ARGUMENT
04487 * -AUBO_BAD_STATE
04488 *
04489 * @throws arcs::common_interface::AuboException
04490 *
04491 * @par Python
04492 * resumeStopLine(self: pyaubo_sdk.MotionControl, arg0: float, arg1: float)
04493 * -> int
04494 *
04495 * @par Lua
04496 * resumeStopLine(acc: number, acc_rot: number) -> nil
04497 *
04498 * @par Lua
04499 * resumeStopLine(10,10)
04500 *
04501 * @par JSON-RPC
04502 * {"jsonrpc":"2.0","method":"rob1.MotionControl.resumeStopLine","params":[10,10],"id":1}
04503 *
04504 * @par JSON-RPC
04505 * {"id":1,"jsonrpc":"2.0","result":-1}
04506 *
04507 * \endchinese
04508 * \english
04509 * Stop motion in Cartesian space (used after moving to a safe position via
04510 * resumeSpeedLine following a collision during process execution)
04511 *
04512 * @param acc Position acceleration, unit: m/s^2
04513 * @param acc_rot Orientation acceleration, unit: rad/s^2
04514 * @return Returns 0 on success; otherwise returns an error code
04515 * AUBO_BUSY
04516 * AUBO_BAD_STATE
04517 * -AUBO_INVL_ARGUMENT
04518 * -AUBO_BAD_STATE
04519 *
04520 * @throws arcs::common_interface::AuboException
04521 *
04522 * @par Python function prototype
04523 * resumeStopLine(self: pyaubo_sdk.MotionControl, arg0: float, arg1: float)
04524 * -> int
04525 *
04526 * @par Lua function prototype
04527 * resumeStopLine(acc: number, acc_rot: number) -> nil
04528 *
04529 * @par Lua example
04530 * resumeStopLine(10,10)
04531 *
04532 * @par JSON-RPC request example
04533 * {"jsonrpc":"2.0","method":"rob1.MotionControl.resumeStopLine","params":[10,10],"id":1}
04534 *
04535 * @par JSON-RPC response example
04536 * {"id":1,"jsonrpc":"2.0","result":-1}
04537 *
04538 * \endenglish
04539 */
04540 int resumeStopLine(double acc, double acc_rot);
04541
04542 /**
04543 * @ingroup MotionControl
04544 * \chinese
04545 *
04546 * \warning weaveStart RuntimeMachine start
04547 *
04548 * weaveStart weaveEnd
04549 * moveLine/moveCircle/moveProcess params
04550 *
04551 * @param params Json
04552 *

```

```

04553 * @par v0.29
04554 * - "type": <string>          "SINE" |
04555 * - "SPIRAL" | "TRIANGLE" | "TRAPEZOIDAL"
04556 * - "step": <number>          _____
04557 * - "amplitude": [<number>,<number>] _____ m
04558 * - "hold_distance": [<number>,<number>] _____
04559 * - "hold_time": [<number>,<number>] _____ s
04560 * - "angle": <number> _____ rad
04561 * - "direction": <integer> _____
04562 * - "movep_uniform_vel": <bool> _____ moveProcess
04563 *
04564 * @par v0.31
04565 * - v0.29
04566 *
04567 * @par v0.32 /
04568 * - "angle": [<number>,<number>] _____ rad
04569 * - "frequency": <number> _____ Hz [0.1, 5]
04570 * - "ori_weave_range": [<number>,<number>,<number>] _____
04571 * [x,y,z] rad
04572 * - "ori_weave_frequency": [<number>,<number>,<number>] _____
04573 * [x,y,z] m^-1 [0, 1]
04574 * - "adjust_cycle_num": <integer> _____
04575 * - "azimuth": <number> _____ rad
04576 * - "ridge_height": <number> _____ m
04577 *
04578 * \warning **frequency = vel / step** vel vel
04579 * m · s^-1
04580 * \warning **hold_time** **SINE**
04581 * \warning "hold_distance"/"hold_time"
04582 *
04583 * \note
04584 * - "amplitude"/"hold_distance"/"hold_time"
04585 * [left,right] "ori_weave_" [x,y,z]
04586 * - rad
04587 *
04588 * @return 0;
04589 * - AUBO_BUSY
04590 * - AUBO_BAD_STATE
04591 * - AUBO_INVL_ARGUMENT
04592 * - AUBO_BAD_STATE
04593 *
04594 * @throws arcs::common_interface::AuboException
04595 *
04596 * @par v0.29/v0.31
04597 * \code{.json}
04598 * params = {
04599 *   "type": "SINE",
04600 *   "step": 0.005,
04601 *   "amplitude": [0.01, 0.01],
04602 *   "hold_distance": [0.001, 0.001],
04603 *   "hold_time": [0, 0],
04604 *   "angle": 0,
04605 *   "direction": 0,
04606 *   "movep_uniform_vel": false
04607 * }
04608 * \endcode
04609 *
04610 * @par v0.32
04611 * \code{.json}
04612 * params = {
04613 *   "type": "SINE", // "SINE" "SPIRAL" "TRIANGLE" "SAWTOOTH" "CRESCENT"
04614 *   "step": 0.0, // frequency = vel / step vel: m · s^-1
04615 *   "frequency": 0.0, // [0.1, 5] Hz
04616 *   "amplitude": [0.01, 0.01], // m
04617 *   "hold_distance": [0.001, 0.001], // m
04618 *   "hold_time": [0, 0], // s SINE
04619 *   "angle": [0, 0], // rad
04620 *   "direction": 0,
04621 *   "ori_weave_range": [0.001, 0.001, 0.001], // rad
04622 *   "ori_weave_frequency": [0.001, 0.001, 0.001], // m^-1
04623 *   "adjust_cycle_num": 0,
04624 *   "azimuth": 0, // rad
04625 *   "ridge_height": 0 // m
04626 * }
04627 * \endcode
04628 *
04629 * @par Python
04630 * \code{.py}
04631 * weaveStart(self: pyaubo_sdk.MotionControl, arg0: str) -> int
04632 * \endcode

```

```

04640 *
04641 * @par Python
04642 * \code{.py}
04643 * robot_name = rpc_cli.getRobotNames()[0]
04644 * robot_interface = rpc_cli.getRobotInterface(robot_name)
04645 * robot_interface.getMotionControl().weaveStart(params)
04646 * \endcode
04647 *
04648 * @par Lua
04649 * \code{.lua}
04650 * weaveStart(params: string) -> nil
04651 * \endcode
04652 *
04653 * @par Lua v0.29/0.31
04654 * \code{.lua}
04655 *
04656 weaveStart("{\type\":"SINE\","step\:0.005,\"amplitude\":[0.01,0.01],\"hold_distance\":[0.001,0.001],\"hold_time\":[0,0],\"angle\":0,\"direction\":"X\"}")
04657 * \endcode
04658 * @par Lua v0.32
04659 * \code{.lua}
04660 *
04661 weaveStart("{\type\":"SINE\","step\:0.0,\"frequency\:2.0,\"amplitude\":[0.01,0.01],\"hold_distance\":[0.001,0.001],\"hold_time\":[0,0],\"angle\":"0\"}")
04662 * \endcode
04663 * @par JSON-RPC v0.29/0.31
04664 * \code{.json}
04665 *
04666 {"jsonrpc":"2.0","method":"rob1.MotionControl.weaveStart","params":[{"\type\":"SINE\","step\:0.005,\"amplitude\":[0.01,0.01],\"hold_distance\":[0.001,0.001],\"hold_time\":[0,0],\"angle\":"0\"}]}
04667 * \endcode
04668 * @par JSON-RPC v0.32
04669 * \code{.json}
04670 *
04671 {"jsonrpc":"2.0","method":"rob1.MotionControl.weaveStart","params":[{"\type\":"SINE\","step\:0.0,\"frequency\:2.0,\"amplitude\":[0.01,0.01],\"hold_distance\":[0.001,0.001],\"hold_time\":[0,0],\"angle\":"0\"}]}
04672 * \endcode
04673 * @par JSON-RPC
04674 * \code{.json}
04675 * {"id":1,"jsonrpc":"2.0","result":0}
04676 * \endcode
04677 * \endchinese
04678 *
04679 * \english
04680 * Start weaving: between weaveStart and weaveEnd,
04681 * moveLine/moveCircle/moveProcess follows params.
04682 *
04683 * \warning Start RuntimeMachine before calling weaveStart.
04684 *
04685 * @param params Json string.
04686 *
04687 * @par Since v0.29
04688 * - "type": <string> — Waveform: "SINE" | "SPIRAL"
04689 * | "TRIANGLE" | "TRAPEZOIDAL" (case-sensitive).
04690 * - "step": <number> — Arc-length sampling step,
04691 * m.
04692 * - "amplitude": [<number>,<number>] — Left/right weave amplitude,
04693 * m.
04694 * - "hold_distance": [<number>,<number>] — Dwell distance at peak
04695 * amplitude along path, m.
04696 * - "hold_time": [<number>,<number>] — Dwell time at peak
04697 * amplitude, s.
04698 * - "angle": <number> — Angle to normal plane, rad.
04699 * - "direction": <integer> — Initial direction: 0=above
04700 * path, 1=below path.
04701 * - "movep_uniform_vel": <bool> — Uniform-velocity planning
04702 * (boolean value).
04703 *
04704 * @par Since v0.31
04705 * - Same as v0.29.
04706 *
04707 * @par New/changed in v0.32
04708 * - "frequency": <number> — Weave frequency, Hz (range
04709 * [0.1, 5]).
04710 * - "ori_weave_range": [<number>,<number>,<number>] — Attitude weave
04711 * range [x,y,z], rad; often used with Archimedean spiral for force-guided
04712 * hole finding.
04713 * - "ori_weave_frequency": [<number>,<number>,<number>] — Attitude weave
04714 * frequency [x,y,z], m^-1 (per unit arc; range [0, 1]).
04715 * - "adjust_cycle_num": <integer> — Number of transition
04716 * cycles.
04717 * - "azimuth": <number> — Waveform azimuth, rad.
04718 * - "ridge_height": <number> — Center raise height, m.
04719 *
04720 * \warning **hold_time** is supported for **SINE** only and may be
04721 * asymmetric. \warning You may relate **frequency** = vel / step** when an
04722 * external path speed vel (m · s^-1) is available.

```

```

04723 *
04724 * \note
04725 * - Two-element arrays are [left,right]; three-element arrays are attitude
04726 * [x,y,z].
04727 * - All angles are in radians (rad).
04728 *
04729 * @return Returns 0 on success; otherwise returns an error code:
04730 * - AUBO_BUSY
04731 * - AUBO_BAD_STATE
04732 * - AUBO_INVL_ARGUMENT
04733 * - AUBO_BAD_STATE
04734 *
04735 * @throws arcs::common_interface::AuboException
04736 *
04737 * @par Example v0.29/v0.31
04738 * \code{.json}
04739 * params = {
04740 *   "type": "SINE",
04741 *   "step": 0.005,
04742 *   "amplitude": [0.01, 0.01],
04743 *   "hold_distance": [0.001, 0.001],
04744 *   "hold_time": [0, 0],
04745 *   "angle": 0,
04746 *   "direction": 0,
04747 *   "movep_uniform_vel": false
04748 * }
04749 * \endcode
04750 *
04751 * @par Example v0.32
04752 * \code{.json}
04753 * params = {
04754 *   "type": "SINE", // "SINE" "SPIRAL" "TRIANGLE" "TRAPEZOIDAL"
04755 *   "step": 0.0, // frequency = vel / step (vel: m · s-1)
04756 *   "frequency": 0.0, // [0.1, 5] Hz
04757 *   "amplitude": [0.01, 0.01], // m
04758 *   "hold_distance": [0.001, 0.001], // m
04759 *   "hold_time": [0, 0], // s (SINE only)
04760 *   "angle": 0, // rad
04761 *   "direction": 0,
04762 *   "ori_weave_range": [0.001, 0.001, 0.001], // rad
04763 *   "ori_weave_frequency": [0.001, 0.001, 0.001], // m-1
04764 *   "adjust_cycle_num": 0,
04765 *   "azimuth": 0, // rad
04766 *   "ridge_height": 0 // m
04767 * }
04768 * \endcode
04769 *
04770 * @par Python function prototype
04771 * \code{.py}
04772 * weaveStart(self: pyaubo_sdk.MotionControl, arg0: str) -> int
04773 * \endcode
04774 *
04775 * @par Python example
04776 * \code{.py}
04777 * robot_name = rpc_cli.getRobotNames()[0]
04778 * robot_interface = rpc_cli.getRobotInterface(robot_name)
04779 * robot_interface.getMotionControl().weaveStart(params)
04780 * \endcode
04781 *
04782 * @par Lua function prototype
04783 * \code{.lua}
04784 * weaveStart(params: string) -> nil
04785 * \endcode
04786 *
04787 * @par Lua example (v0.29/0.31)
04788 * \code{.lua}
04789 *
04790 * weaveStart("{ \"type\": \"SINE\", \"step\": 0.005, \"amplitude\": [0.01, 0.01], \"hold_distance\": [0.001, 0.001], \"hold_time\": [0, 0], \"angle\": 0, \"direction\": 0 }")
04791 * \endcode
04792 *
04793 * @par Lua example (v0.32)
04794 * \code{.lua}
04795 *
04796 * weaveStart("{ \"type\": \"SINE\", \"step\": 0.0, \"frequency\": 2.0, \"amplitude\": [0.01, 0.01], \"hold_distance\": [0.001, 0.001], \"hold_time\": [0, 0], \"angle\": 0, \"direction\": 0 }")
04797 * \endcode
04798 *
04799 * @par JSON-RPC request example (v0.29/0.31)
04800 * \code{.json}
04801 *
04802 * @par JSON-RPC request example (v0.32)
04803 * \code{.json}
04804 *
04805 * {"jsonrpc": "2.0", "method": "rob1.MotionControl.weaveStart", "params": [{" \"type\": \"SINE\", \"step\": 0.005, \"amplitude\": [0.01, 0.01], \"hold_distance\": [0.001, 0.001], \"hold_time\": [0, 0], \"angle\": 0, \"direction\": 0 }"]}
04806 * \endcode

```

```

04806     *
04807     * @par JSON-RPC response example
04808     * \code{.json}
04809     * {"id":1,"jsonrpc":"2.0","result":0}
04810     * \endcode
04811     * \endenglish
04812     */
04813     int weaveStart(const std::string &params);
04814
04815     /**
04816     * @ingroup MotionControl
04817     * \chinese
04818     *
04819     *
04820     * @return 0
04821     * AUBO_BUSY
04822     * AUBO_BAD_STATE
04823     * -AUBO_BAD_STATE
04824     *
04825     * @throws arcs::common_interface::AuboException
04826     *
04827     * @par JSON-RPC
04828     * {"jsonrpc":"2.0","method":"rob1.MotionControl.weaveEnd","params":[],"id":1}
04829     *
04830     * @par JSON-RPC
04831     * {"id":1,"jsonrpc":"2.0","result":0}
04832     *
04833     * \endchinese
04834     * \english
04835     * End weaving
04836     *
04837     * @return Returns 0 on success; otherwise returns an error code
04838     * AUBO_BUSY
04839     * AUBO_BAD_STATE
04840     * -AUBO_BAD_STATE
04841     *
04842     * @throws arcs::common_interface::AuboException
04843     *
04844     * @par JSON-RPC request example
04845     * {"jsonrpc":"2.0","method":"rob1.MotionControl.weaveEnd","params":[],"id":1}
04846     *
04847     * @par JSON-RPC response example
04848     * {"id":1,"jsonrpc":"2.0","result":0}
04849     *
04850     * \endenglish
04851     */
04852     int weaveEnd();
04853
04854     /**
04855     * @ingroup MotionControl
04856     * \chinese
04857     *
04858     *
04859     * @param sample_time      : m/s^2
04860     * @return 0
04861     * AUBO_BUSY
04862     * AUBO_BAD_STATE
04863     * -AUBO_INVL_ARGUMENT
04864     * -AUBO_BAD_STATE
04865     *
04866     * @throws arcs::common_interface::AuboException
04867     * \endchinese
04868     * \english
04869     * Set the sampling interval for points on the future path
04870     *
04871     * @param sample_time Sampling interval, unit: m/s^2
04872     * @return Returns 0 on success; otherwise returns an error code
04873     * AUBO_BUSY
04874     * AUBO_BAD_STATE
04875     * -AUBO_INVL_ARGUMENT
04876     * -AUBO_BAD_STATE
04877     *
04878     * @throws arcs::common_interface::AuboException
04879     * \endenglish
04880     */
04881     int setFuturePointSamplePeriod(double sample_time);
04882
04883     /**
04884     * @ingroup MotionControl
04885     * \chinese
04886     *
04887     *
04888     * @return (100ms * 10)
04889     *
04890     * @throws arcs::common_interface::AuboException
04891     *
04892     * @par JSON-RPC

```

```

04893 * {"jsonrpc":"2.0","method":"rob1.MotionControl.getFuturePathPointsJoint","params":[],"id":1}
04894 *
04895 * @par JSON-RPC
04896 * {"id":1,"jsonrpc":"2.0","result":[]}
04897 *
04898 * \endchinese
04899 * \english
04900 * Get trajectory points on the future path
04901 *
04902 * @return Waypoints (100ms * 10)
04903 *
04904 * @throws arcs::common_interface::AuboException
04905 *
04906 * @par JSON-RPC request example
04907 * {"jsonrpc":"2.0","method":"rob1.MotionControl.getFuturePathPointsJoint","params":[],"id":1}
04908 *
04909 * @par JSON-RPC response example
04910 * {"id":1,"jsonrpc":"2.0","result":[]}
04911 *
04912 * \endenglish
04913 */
04914 std::vector<std::vector<double>> getFuturePathPointsJoint();
04915
04916 /**
04917 * @ingroup MotionControl
04918 * \chinese
04919 *
04920 *
04921 * @param encoder_id
04922 * @param tick_per_meter
04923 * ==>
04924 * ==>
04925 * @return
04926 *
04927 * @throws arcs::common_interface::AuboException
04928 *
04929 * @par Python
04930 * setConveyorTrackEncoder(self: pyaubo_sdk.MotionControl) -> int
04931 *
04932 * @par Lua
04933 * setConveyorTrackEncoder(encoder_id: bumber,tick_per_meter: number) -> int
04934 *
04935 * @par Lua
04936 * num = setConveyorTrackEncoder(1,40000)
04937 *
04938 * @par JSON-RPC
04939 * {"jsonrpc":"2.0","method":"rob1.MotionControl.setConveyorTrackEncoder","params":[1,40000],"id":1}
04940 *
04941 * @par JSON-RPC
04942 * {"id":1,"jsonrpc":"2.0","result":0}
04943 *
04944 * \endchinese
04945 * \english
04946 * Set conveyor encoder parameters
04947 *
04948 * @param encoder_id Reserved
04949 * @param tick_per_meter
04950 * Linear conveyor: pulses per meter
04951 * Circular conveyor: pulses per revolution
04952 * @return
04953 *
04954 * @throws arcs::common_interface::AuboException
04955 *
04956 * @par Python function prototype
04957 * setConveyorTrackEncoder(self: pyaubo_sdk.MotionControl) -> int
04958 *
04959 * @par Lua function prototype
04960 * setConveyorTrackEncoder(encoder_id: bumber,tick_per_meter: number) -> int
04961 *
04962 * @par Lua example
04963 * num = setConveyorTrackEncoder(1,40000)
04964 *
04965 * @par JSON-RPC request example
04966 * {"jsonrpc":"2.0","method":"rob1.MotionControl.setConveyorTrackEncoder","params":[1,40000],"id":1}
04967 *
04968 * @par JSON-RPC response example
04969 * {"id":1,"jsonrpc":"2.0","result":0}
04970 *
04971 * \endenglish
04972 */
04973 int setConveyorTrackEncoder(int encoder_id, int tick_per_meter);
04974 /**
04975 * @ingroup MotionControl
04976 * \chinese
04977 *
04978 *
04979 * @note

```



```

04980  *
04981  * @param encoder_id
04982  * 0-
04983  * @param rotate_tool
04984  * @return
04985  *
04986  * @throws arcs::common_interface::AuboException
04987  * \endchinese
04988  * \english
04989  * Circular conveyor tracking
04990  *
04991  * @note Not implemented yet
04992  *
04993  * @param encoder_id
04994  * 0 - integrated sensor
04995  * @param rotate_tool
04996  * @return
04997  *
04998  * @throws arcs::common_interface::AuboException
04999  * \endenglish
05000  */
05001 int conveyorTrackCircle(int encoder_id, const std::vector<double> &center,
05002                          bool rotate_tool);
05003
05004 /**
05005  * @ingroup MotionControl
05006  * \chinese
05007  *
05008  *
05009  * @param encoder_id
05010  * @param direction
05011  * @return
05012  *
05013  * @throws arcs::common_interface::AuboException
05014  *
05015  * @par Python
05016  * conveyorTrackLine(self: pyaubo_sdk.MotionControl) -> int
05017  *
05018  * @par Lua
05019  * conveyorTrackLine -> int
05020  *
05021  * @par JSON-RPC
05022  * {"jsonrpc":"2.0","method":"rob1.MotionControl.conveyorTrackLine","params":[1,[1.0,0.0,0.0,0.0,0.0,0.0]],"id":1}
05023  *
05024  * @par JSON-RPC
05025  * {"id":1,"jsonrpc":"2.0","result":0}
05026  *
05027  * \endchinese
05028  * \english
05029  * Linear conveyor tracking
05030  *
05031  * @param encoder_id Reserved
05032  * @param direction The movement direction of the conveyor relative to the
05033  * robot base coordinate system
05034  * @return
05035  *
05036  * @throws arcs::common_interface::AuboException
05037  *
05038  * @par Python function prototype
05039  * conveyorTrackLine(self: pyaubo_sdk.MotionControl) -> int
05040  *
05041  * @par Lua function prototype
05042  * conveyorTrackLine -> int
05043  *
05044  * @par JSON-RPC request example
05045  * {"jsonrpc":"2.0","method":"rob1.MotionControl.conveyorTrackLine","params":[1,[1.0,0.0,0.0,0.0,0.0,0.0]],"id":1}
05046  *
05047  * @par JSON-RPC response example
05048  * {"id":1,"jsonrpc":"2.0","result":0}
05049  *
05050  * \endenglish
05051  */
05052 int conveyorTrackLine(int encoder_id, const std::vector<double> &direction);
05053
05054 /**
05055  * @ingroup MotionControl
05056  * \chinese
05057  *
05058  *
05059  * @param encoder_id
05060  * @param a
05061  * @return
05062  *
05063  * @throws arcs::common_interface::AuboException
05064  *
05065  * @par Python
05066  * conveyorTrackStop(self: pyaubo_sdk.MotionControl) -> int

```

```

05067 *
05068 * @par Lua
05069 * conveyorTrackStop -> int
05070 *
05071 * @par JSON-RPC
05072 * {"jsonrpc":"2.0","method":"rob1.MotionControl.conveyorTrackStop","params":[0,1.0],"id":1}
05073 *
05074 * @par JSON-RPC
05075 * {"id":1,"jsonrpc":"2.0","result":0}
05076 * \endchinese
05077 * \english
05078 * Stop conveyor tracking
05079 *
05080 * @param encoder_id Reserved
05081 * @param a Reserved
05082 * @return
05083 *
05084 * @throws arcs::common_interface::AuboException
05085 *
05086 * @par Python function prototype
05087 * conveyorTrackStop(self: pyaubo_sdk.MotionControl) -> int
05088 *
05089 * @par Lua function prototype
05090 * conveyorTrackStop -> int
05091 *
05092 * @par JSON-RPC request example
05093 * {"jsonrpc":"2.0","method":"rob1.MotionControl.conveyorTrackStop","params":[0,1.0],"id":1}
05094 *
05095 * @par JSON-RPC response example
05096 * {"id":1,"jsonrpc":"2.0","result":0}
05097 * \endenglish
05098 */
05099 int conveyorTrackStop(int encoder_id, double a);
05100
05101 /**
05102 * @ingroup MotionControl
05103 * \chinese
05104 *
05105 *         true
05106 *         false
05107 *
05108 * @return
05109 *
05110 * @throws arcs::common_interface::AuboException
05111 *
05112 * @param encoder_id
05113 *
05114 * @par Python
05115 * conveyorTrackSwitch(self: pyaubo_sdk.MotionControl) -> bool
05116 *
05117 * @par Lua
05118 * conveyorTrackSwitch() -> boolean
05119 *
05120 * @par JSON-RPC
05121 * {"jsonrpc":"2.0","method":"rob1.MotionControl.conveyorTrackSwitch","params":[0],"id":1}
05122 *
05123 * @par JSON-RPC
05124 * {"id":1,"jsonrpc":"2.0","result":true}
05125 *
05126 * \endchinese
05127 * \english
05128 * Switch conveyor tracking item.
05129 * If the current item is being tracked, it will be dequeued and no longer
05130 * tracked, returning true. If no item is currently being tracked, returns
05131 * false.
05132 *
05133 * @return
05134 *
05135 * @throws arcs::common_interface::AuboException
05136 *
05137 * @param encoder_id Reserved
05138 *
05139 * @par Python function prototype
05140 * conveyorTrackSwitch(self: pyaubo_sdk.MotionControl) -> bool
05141 *
05142 * @par Lua function prototype
05143 * conveyorTrackSwitch() -> boolean
05144 *
05145 * @par JSON-RPC request example
05146 * {"jsonrpc":"2.0","method":"rob1.MotionControl.conveyorTrackSwitch","params":[0],"id":1}
05147 *
05148 * @par JSON-RPC response example
05149 * {"id":1,"jsonrpc":"2.0","result":true}
05150 *
05151 * \endenglish
05152 */
05153 bool conveyorTrackSwitch(int encoder_id);

```

```

05154
05155 /**
05156  * @ingroup MotionControl
05157  * \chinese
05158  *
05159  * @param encoder_id
05160  * @return
05161  *         true
05162  *
05163  *         true
05164  *         false
05165  *
05166  * @throws arcs::common_interface::AuboException
05167  *
05168  * @par Python
05169  * hasItemOnConveyorToTrack(self: pyaubo_sdk.MotionControl) -> bool
05170  *
05171  * @par Lua
05172  * hasItemOnConveyorToTrack() -> boolean
05173  *
05174  * @par JSON-RPC
05175  * {"jsonrpc":"2.0","method":"rob1.MotionControl.hasItemOnConveyorToTrack","params":[0],"id":1}
05176  *
05177  * @par JSON-RPC
05178  * {"id":1,"jsonrpc":"2.0","result":{"true"}}
05179  * \endchinese
05180  * \english
05181  * Whether there is an item on the conveyor that can be tracked
05182  * @param encoder_id Reserved
05183  * @return
05184  * If the first item in the queue is in tracking state, returns true.
05185  * If the first item is not in tracking state, checks the rest of the queue
05186  * for items within the start window. Items outside the start window are
05187  * dequeued until an item is found within the window, which is then set to
05188  * tracking state and returns true. If no item in the queue is within the
05189  * start window, returns false.
05190  *
05191  * @throws arcs::common_interface::AuboException
05192  *
05193  * @par Python function prototype
05194  * hasItemOnConveyorToTrack(self: pyaubo_sdk.MotionControl) -> bool
05195  *
05196  * @par Lua function prototype
05197  * hasItemOnConveyorToTrack() -> boolean
05198  *
05199  * @par JSON-RPC request example
05200  * {"jsonrpc":"2.0","method":"rob1.MotionControl.hasItemOnConveyorToTrack","params":[0],"id":1}
05201  *
05202  * @par JSON-RPC response example
05203  * {"id":1,"jsonrpc":"2.0","result":{"true"}}
05204  * \endenglish
05205  */
05206 bool hasItemOnConveyorToTrack(int encoder_id);
05207
05208 /**
05209  * @ingroup MotionControl
05210  * \chinese
05211  *
05212  *
05213  * @param encoder_id
05214  * @param item_id ID
05215  * @param offset
05216  * @return
05217  *
05218  * @throws arcs::common_interface::AuboException
05219  *
05220  * @par Python
05221  * conveyorTrackCreatItem(self: pyaubo_sdk.MotionControl, arg0: int,
05222  * arg1:int, arg2: List[float]) -> int
05223  *
05224  * @par Lua
05225  * conveyorTrackCreatItem(encoder_id: number, item_id: number, offset:
05226  * table) -> number
05227  *
05228  * @par JSON-RPC
05229  * {"jsonrpc":"2.0","method":"rob1.MotionControl.conveyorTrackCreatItem","params":[0,2,
05230  * {0.0,0.0,0.0,0.0,0.0,0.0}], "id":1}
05231  *
05232  * @par JSON-RPC
05233  * {"id":1,"jsonrpc":"2.0","result":0.0}
05234  * \endchinese
05235  * \english
05236  * Add an item to the conveyor queue
05237  *
05238  * @param encoder_id Reserved
05239  * @param item_id Item ID
05240  * @param offset Offset of the current item position relative to the

```

```

05241 * template item position
05242 * @return
05243 *
05244 * @throws arcs::common_interface::AuboException
05245 *
05246 * @par Python function prototype
05247 * conveyorTrackCreatItem(self: pyaubo_sdk.MotionControl, arg0: int,
05248 * arg1:int, arg2: List[float]) -> int
05249 *
05250 * @par Lua function prototype
05251 * conveyorTrackCreatItem(encoder_id: number, item_id: number, offset:
05252 * table) -> number
05253 *
05254 * @par JSON-RPC request example
05255 * {"jsonrpc":"2.0","method":"rob1.MotionControl.conveyorTrackCreatItem","params":[0,2,
05256 * {0.0,0.0,0.0,0.0,0.0,0.0}], "id":1}
05257 *
05258 * @par JSON-RPC response example
05259 * {"id":1,"jsonrpc":"2.0","result":0.0}
05260 * \endenglish
05261 */
05262 int conveyorTrackCreatItem(int encoder_id, int item_id,
05263                             const std::vector<double> &offset);
05264 /**
05265 * @ingroup MotionControl
05266 * \chinese
05267 *
05268 *
05269 * @param encoder_id
05270 * @param comp
05271 * @return
05272 *
05273 * @throws arcs::common_interface::AuboException
05274 *
05275 * @par Python
05276 * setConveyorTrackCompensate(self: pyaubo_sdk.MotionControl, arg0: int,
05277 * arg1: float) -> int
05278 *
05279 * @par Lua
05280 * setConveyorTrackCompensate(comp: number) -> number
05281 *
05282 * @par JSON-RPC
05283 * {"jsonrpc":"2.0","method":"rob1.MotionControl.setConveyorTrackCompensate","params":[0,
05284 * 0.1], "id":1}
05285 *
05286 * @par JSON-RPC
05287 * {"id":1,"jsonrpc":"2.0","result":0}
05288 * \endchinese
05289 * \english
05290 * Set the compensation value for conveyor tracking
05291 *
05292 * @param encoder_id Reserved
05293 * @param comp Conveyor compensation value
05294 * @return
05295 *
05296 * @throws arcs::common_interface::AuboException
05297 *
05298 * @par Python function prototype
05299 * setConveyorTrackCompensate(self: pyaubo_sdk.MotionControl, arg0: int,
05300 * arg1: float) -> int
05301 *
05302 * @par Lua function prototype
05303 * setConveyorTrackCompensate(comp: number) -> number
05304 *
05305 * @par JSON-RPC request example
05306 * {"jsonrpc":"2.0","method":"rob1.MotionControl.setConveyorTrackCompensate","params":[0,
05307 * 0.1], "id":1}
05308 *
05309 * @par JSON-RPC response example
05310 * {"id":1,"jsonrpc":"2.0","result":0}
05311 * \endenglish
05312 */
05313 int setConveyorTrackCompensate(int encoder_id, double comp);
05314 /**
05315 * @ingroup MotionControl
05316 * \chinese
05317 *
05318 *
05319 *
05320 * @param encoder_id
05321 * @return
05322 *
05323 * @throws arcs::common_interface::AuboException
05324 *
05325 * @par Python
05326 * isConveyorTrackSync(self: pyaubo_sdk.MotionControl, arg0: int) -> bool
05327 *

```

```

05328 * @par Lua
05329 * isConveyorTrackSync(encoder_id: number) -> bool
05330 *
05331 * @par JSON-RPC
05332 * {"jsonrpc":"2.0","method":"rob1.MotionControl.isConveyorTrackSync","params":[0],"id":1}
05333 *
05334 * @par JSON-RPC
05335 * {"id":1,"jsonrpc":"2.0","result":false}
05336 *
05337 * \endchinese
05338 * \english
05339 * Determine whether the conveyor and the robot arm have reached relative
05340 * rest
05341 *
05342 * @param encoder_id Reserved
05343 * @return
05344 *
05345 * @throws arcs::common_interface::AuboException
05346 *
05347 * @par Python function prototype
05348 * isConveyorTrackSync(self: pyaubo_sdk.MotionControl, arg0: int) -> bool
05349 *
05350 * @par Lua function prototype
05351 * isConveyorTrackSync(encoder_id: number) -> bool
05352 *
05353 * @par JSON-RPC request example
05354 * {"jsonrpc":"2.0","method":"rob1.MotionControl.isConveyorTrackSync","params":[0],"id":1}
05355 *
05356 * @par JSON-RPC response example
05357 * {"id":1,"jsonrpc":"2.0","result":false}
05358 *
05359 * \endenglish
05360 */
05361 bool isConveyorTrackSync(int encoder_id);
05362
05363 /**
05364 * @ingroup MotionControl
05365 * \chinese
05366 *
05367 *
05368 * @param encoder_id
05369 * @param limit
05370 * ,
05371 * @return
05372 *
05373 * @throws arcs::common_interface::AuboException
05374 *
05375 * @par Python
05376 * setConveyorTrackLimit(self: pyaubo_sdk.MotionControl, arg0: int, arg1:
05377 * double) -> int
05378 *
05379 * @par Lua
05380 * setConveyorTrackLimit(encoder_id: number, limit: number) -> int
05381 *
05382 * @par JSON-RPC
05383 * {"jsonrpc":"2.0","method":"rob1.MotionControl.setConveyorTrackLimit","params":[0,
05384 * 1.5],"id":1}
05385 *
05386 * @par JSON-RPC
05387 * {"id":1,"jsonrpc":"2.0","result":0}
05388 * \endchinese
05389 * \english
05390 * Set the maximum distance limit for conveyor tracking
05391 *
05392 * @param encoder_id Reserved
05393 * @param limit
05394 * Maximum distance limit for conveyor tracking, unit: meter
05395 * @return
05396 *
05397 * @throws arcs::common_interface::AuboException
05398 *
05399 * @par Python function prototype
05400 * setConveyorTrackLimit(self: pyaubo_sdk.MotionControl, arg0: int, arg1:
05401 * double) -> int
05402 *
05403 * @par Lua function prototype
05404 * setConveyorTrackLimit(encoder_id: number, limit: number) -> int
05405 *
05406 * @par JSON-RPC request example
05407 * {"jsonrpc":"2.0","method":"rob1.MotionControl.setConveyorTrackLimit","params":[0,
05408 * 1.5],"id":1}
05409 *
05410 * @par JSON-RPC response example
05411 * {"id":1,"jsonrpc":"2.0","result":0}
05412 * \endenglish
05413 */
05414 int setConveyorTrackLimit(int encoder_id, double limit);

```

```

05415
05416 /**
05417  * @ingroup MotionControl
05418  * \chinese
05419  *
05420  *
05421  * @param encoder_id
05422  * @param min_window
05423  *
05424  * @param max_window
05425  *
05426  * @return
05427  *
05428  * @throws arcs::common_interface::AuboException
05429  *
05430  * @par Python
05431  * setConveyorTrackStartWindow(self: pyaubo_sdk.MotionControl, arg0: int,
05432  * arg1: double, arg2: double) -> int
05433  *
05434  * @par Lua
05435  * setConveyorTrackStartWindow(encoder_id: number, min_window: number,
05436  * max_window: number) -> int
05437  *
05438  * @par JSON-RPC
05439  * {"jsonrpc":"2.0","method":"rob1.MotionControl.setConveyorTrackStartWindow","params":[0,
05440  * 0.2, 1.0],"id":1}
05441  *
05442  * @par JSON-RPC
05443  * {"id":1,"jsonrpc":"2.0","result":0}
05444  *
05445  * \endchinese
05446  * \english
05447  * Set the start window for conveyor tracking
05448  *
05449  * @param encoder_id Reserved
05450  * @param min_window
05451  * Start position of the window, unit: meter
05452  * @param max_window
05453  * End position of the window, unit: meter
05454  * @return
05455  *
05456  * @throws arcs::common_interface::AuboException
05457  *
05458  * @par Python function prototype
05459  * setConveyorTrackStartWindow(self: pyaubo_sdk.MotionControl, arg0: int,
05460  * arg1: double, arg2: double) -> int
05461  *
05462  * @par Lua function prototype
05463  * setConveyorTrackStartWindow(encoder_id: number, min_window: number,
05464  * max_window: number) -> int
05465  *
05466  * @par JSON-RPC request example
05467  * {"jsonrpc":"2.0","method":"rob1.MotionControl.setConveyorTrackStartWindow","params":[0,
05468  * 0.2, 1.0],"id":1}
05469  *
05470  * @par JSON-RPC response example
05471  * {"id":1,"jsonrpc":"2.0","result":0}
05472  *
05473  * \endenglish
05474  */
05475 int setConveyorTrackStartWindow(int encoder_id, double window_min,
05476                                double window_max);
05477
05478 /**
05479  * @ingroup MotionControl
05480  * \chinese
05481  *
05482  *
05483  * @param encoder_id
05484  * @param offset
05485  *
05486  * @return
05487  *
05488  * @throws arcs::common_interface::AuboException
05489  *
05490  * @par Python
05491  * setConveyorTrackSensorOffset(self: pyaubo_sdk.MotionControl, arg0: int,
05492  * arg1: double) -> int
05493  *
05494  * @par Lua
05495  * setConveyorTrackSensorOffset(encoder_id: number, offset: number) -> int
05496  *
05497  * @par JSON-RPC
05498  * {"jsonrpc":"2.0","method":"rob1.MotionControl.setConveyorTrackSensorOffset","params":[0,
05499  * 0.2],"id":1}
05500  *
05501  * @par JSON-RPC

```

```

05502 * {"id":1,"jsonrpc":"2.0","result":0}
05503 *
05504 * \endchinese
05505 * \english
05506 * Set the distance from the conveyor teaching position to the sync switch
05507 *
05508 * @param encoder_id Reserved
05509 * @param offset
05510 * Distance from the conveyor teaching position to the sync switch, unit:
05511 * meter
05512 * @return
05513 *
05514 * @throws arcs::common_interface::AuboException
05515 *
05516 * @par Python function prototype
05517 * setConveyorTrackSensorOffset(self: pyaubo_sdk.MotionControl, arg0: int,
05518 * arg1: double) -> int
05519 *
05520 * @par Lua function prototype
05521 * setConveyorTrackSensorOffset(encoder_id: number, offset: number) -> int
05522 *
05523 * @par JSON-RPC request example
05524 * {"jsonrpc":"2.0","method":"rob1.MotionControl.setConveyorTrackSensorOffset","params":[0,
05525 * 0.2],"id":1}
05526 *
05527 * @par JSON-RPC response example
05528 * {"id":1,"jsonrpc":"2.0","result":0}
05529 *
05530 * \endenglish
05531 */
05532 int setConveyorTrackSensorOffset(int encoder_id, double offset);
05533
05534 /**
05535 * @ingroup MotionControl
05536 * \chinese
05537 *
05538 *
05539 * @param encoder_id
05540 * @param distance
05541 *
05542 * @param time
05543 *
05544 *
05545 * distance time 0
05546 * distance time distance
05547 * @return
05548 *
05549 * @throws arcs::common_interface::AuboException
05550 *
05551 * @par Python
05552 * setConveyorTrackSyncSeparation(self: pyaubo_sdk.MotionControl, arg0: int,
05553 * arg1: double, arg2: double) -> int
05554 *
05555 * @par Lua
05556 * setConveyorTrackSyncSeparation(encoder_id: number, distance: number,
05557 * time: number) -> int
05558 *
05559 * @par JSON-RPC
05560 * {"jsonrpc":"2.0","method":"rob1.MotionControl.setConveyorTrackSyncSeparation","params":[0,
05561 * 0.05, 0.2],"id":1}
05562 *
05563 * @par JSON-RPC
05564 * {"id":1,"jsonrpc":"2.0","result":0}
05565 * \endchinese
05566 * \english
05567 * Set conveyor sync separation, used to filter out unwanted signals from
05568 * the sync switch.
05569 *
05570 * @param encoder_id Reserved
05571 * @param distance
05572 * The minimum distance to travel after a sync signal appears before
05573 * accepting a new sync signal as a valid object, unit: meter
05574 * @param time
05575 * The minimum time to elapse after a sync signal appears before accepting a
05576 * new sync signal as a valid object, unit: second
05577 *
05578 * If distance and time are both set greater than 0, the setting takes
05579 * effect. If both distance and time are set, distance takes precedence.
05580 * @return
05581 *
05582 * @throws arcs::common_interface::AuboException
05583 *
05584 * @par Python function prototype
05585 * setConveyorTrackSyncSeparation(self: pyaubo_sdk.MotionControl, arg0: int,
05586 * arg1: double, arg2: double) -> int
05587 *
05588 * @par Lua function prototype

```

```

05589     * setConveyorTrackSyncSeparation(encoder_id: number, distance: number,
05590     * time: number) -> int
05591     *
05592     * @par JSON-RPC request example
05593     * {"jsonrpc":"2.0","method":"rob1.MotionControl.setConveyorTrackSyncSeparation","params":[0,
05594     * 0.05, 0.2],"id":1}
05595     *
05596     * @par JSON-RPC response example
05597     * {"id":1,"jsonrpc":"2.0","result":0}
05598     * \endenglish
05599     */
05600 int setConveyorTrackSyncSeparation(int encoder_id, double distance,
05601                                   double time);
05602
05603 /**
05604  * @ingroup MotionControl
05605  * \chinese
05606  *
05607  *
05608  * @param encoder_id
05609  * @return
05610  * true :
05611  * false :
05612  *
05613  * @throws arcs::common_interface::AuboException
05614  *
05615  * @par Python
05616  * isConveyorTrackExceed(self: pyaubo_sdk.MotionControl, arg0: int) -> bool
05617  *
05618  * @par Lua
05619  * isConveyorTrackExceed(encoder_id: number) -> bool
05620  *
05621  * @par JSON-RPC
05622  * {"jsonrpc":"2.0","method":"rob1.MotionControl.isConveyorTrackExceed","params":[0],"id":1}
05623  *
05624  * @par JSON-RPC
05625  * {"id":1,"jsonrpc":"2.0","result":false}
05626  * \endchinese
05627  * \english
05628  * Whether the workpiece on the conveyor has moved beyond the maximum limit
05629  *
05630  * @param encoder_id Reserved
05631  * @return
05632  * true : The movement distance exceeds the maximum limit
05633  * false : The movement distance does not exceed the maximum limit
05634  *
05635  * @throws arcs::common_interface::AuboException
05636  *
05637  * @par Python function prototype
05638  * isConveyorTrackExceed(self: pyaubo_sdk.MotionControl, arg0: int) -> bool
05639  *
05640  * @par Lua function prototype
05641  * isConveyorTrackExceed(encoder_id: number) -> bool
05642  *
05643  * @par JSON-RPC request example
05644  * {"jsonrpc":"2.0","method":"rob1.MotionControl.isConveyorTrackExceed","params":[0],"id":1}
05645  *
05646  * @par JSON-RPC response example
05647  * {"id":1,"jsonrpc":"2.0","result":false}
05648  * \endenglish
05649  */
05650 bool isConveyorTrackExceed(int encoder_id);
05651
05652 /**
05653  * @ingroup MotionControl
05654  * \chinese
05655  *
05656  *
05657  * @param encoder_id
05658  * @return
05659  *
05660  * @throws arcs::common_interface::AuboException
05661  *
05662  * @par Python
05663  * conveyorTrackClearItems(self: pyaubo_sdk.MotionControl, arg0: int) -> int
05664  *
05665  * @par Lua
05666  * conveyorTrackClearItems(encoder_id: number) -> int
05667  *
05668  * @par JSON-RPC
05669  * {"jsonrpc":"2.0","method":"rob1.MotionControl.conveyorTrackClearItems","params":[0],"id":1}
05670  *
05671  * @par JSON-RPC
05672  * {"id":1,"jsonrpc":"2.0","result":0}
05673  *
05674  * \endchinese
05675  * \english

```



```

05676 * Clear all items in the conveyor queue
05677 *
05678 * @param encoder_id Reserved
05679 * @return
05680 *
05681 * @throws arcs::common_interface::AuboException
05682 *
05683 * @par Python function prototype
05684 * conveyorTrackClearItems(self: pyaubo_sdk.MotionControl, arg0: int) -> int
05685 *
05686 * @par Lua function prototype
05687 * conveyorTrackClearItems(encoder_id: number) -> int
05688 *
05689 * @par JSON-RPC request example
05690 * {"jsonrpc": "2.0", "method": "rob1.MotionControl.conveyorTrackClearItems", "params": [0], "id": 1}
05691 *
05692 * @par JSON-RPC response example
05693 * {"id": 1, "jsonrpc": "2.0", "result": 0}
05694 *
05695 * \endenglish
05696 */
05697 int conveyorTrackClearItems(int encoder_id);
05698
05699 /**
05700 * @ingroup MotionControl
05701 * \chinese
05702 *
05703 *
05704 * @param encoder_id
05705 * @return
05706 *
05707 * @throws arcs::common_interface::AuboException
05708 *
05709 * @par Python
05710 * getConveyorTrackQueue(self: pyaubo_sdk.MotionControl, arg0: int) ->
05711 * List[int]
05712 * @par Lua
05713 * getConveyorTrackQueue(encoder_id: number) -> table
05714 *
05715 * @par JSON-RPC
05716 * {"jsonrpc": "2.0", "method": "rob1.MotionControl.getConveyorTrackQueue", "params": [0], "id": 1}
05717 *
05718 * @par JSON-RPC
05719 * {"id": 1, "jsonrpc": "2.0", "result": {[-500,-200,150,-50]}}
05720 * \endchinese
05721 * \english
05722 * Get encoder values of the conveyor queue
05723 *
05724 * @param encoder_id Reserved
05725 * @return
05726 *
05727 * @throws arcs::common_interface::AuboException
05728 *
05729 * @par Python function prototype
05730 * getConveyorTrackQueue(self: pyaubo_sdk.MotionControl, arg0: int) ->
05731 * List[int]
05732 * @par Lua function prototype
05733 * getConveyorTrackQueue(encoder_id: number) -> table
05734 *
05735 * @par JSON-RPC request example
05736 * {"jsonrpc": "2.0", "method": "rob1.MotionControl.getConveyorTrackQueue", "params": [0], "id": 1}
05737 *
05738 * @par JSON-RPC response example
05739 * {"id": 1, "jsonrpc": "2.0", "result": {[-500,-200,150,-50]}}
05740 * \endenglish
05741 */
05742 std::vector<int> getConveyorTrackQueue(int encoder_id);
05743 /**
05744 * \chinese
05745 * id
05746 *
05747 * @param encoder_id
05748 * @return
05749 * id next item -1
05750 *
05751 * @throws arcs::common_interface::AuboException
05752 *
05753 * @par Python
05754 * getConveyorTrackNextItem(self: pyaubo_sdk.MotionControl, arg0: int) ->
05755 * int
05756 * @par Lua
05757 * getConveyorTrackNextItem(encoder_id: number) -> int
05758 *
05759 * @par JSON-RPC
05760 * {"jsonrpc": "2.0", "method": "rob1.MotionControl.getConveyorTrackNextItem", "params": [0], "id": 1}
05761 *
05762 * @par JSON-RPC

```

```

05763 * {"id":1,"jsonrpc":"2.0","result":{10}}
05764 * \endchinese
05765 * \english
05766 * Get the ID of the next item on the conveyor to be tracked
05767 *
05768 * @param encoder_id Reserved
05769 * @return
05770 * Returns the item ID; returns -1 if there is no next item
05771 *
05772 * @throws arcs::common_interface::AuboException
05773 *
05774 * @par Python function prototype
05775 * getConveyorTrackNextItem(self: pyaubo_sdk.MotionControl, arg0: int) ->
05776 * int
05777 * @par Lua function prototype
05778 * getConveyorTrackNextItem(encoder_id: number) -> int
05779 *
05780 * @par JSON-RPC Request Example
05781 * {"jsonrpc":"2.0","method":"rob1.MotionControl.getConveyorTrackNextItem","params":[0],"id":1}
05782 *
05783 * @par JSON-RPC Response Example
05784 * {"id":1,"jsonrpc":"2.0","result":{10}}
05785 * \endenglish
05786 */
05787
05788 int getConveyorTrackNextItem(int encoder_id);
05789
05790 /**
05791 * @ingroup MotionControl
05792 * \chinese
05793 *
05794 *
05795 * @param param
05796 * @param blend_radius
05797 * @param v
05798 * @param a
05799 * @param t
05800 * @return 0
05801 * AUBO_BUSY
05802 * AUBO_BAD_STATE
05803 * -AUBO_INVL_ARGUMENT
05804 * -AUBO_BAD_STATE
05805 *
05806 * @throws arcs::common_interface::AuboException
05807 *
05808 * @par JSON-RPC
05809 * {"jsonrpc":"2.0","method":"rob1.MotionControl.moveSpiral",
05810 * "params":[{"spiral":0.005,"helix":0.005,"angle":18.84,"plane":0,"frame":[0,0,0,0,0]},0,0.25,1.2,0],"id":1}
05811 *
05812 * @par JSON-RPC
05813 * {"id":1,"jsonrpc":"2.0","result":0}
05814 * \endchinese
05815 * \english
05816 * Spiral motion
05817 *
05818 * @param param Encapsulated parameters
05819 * @param blend_radius
05820 * @param v
05821 * @param a
05822 * @param t
05823 * @return Returns 0 on success; otherwise returns an error code
05824 * AUBO_BUSY
05825 * AUBO_BAD_STATE
05826 * -AUBO_INVL_ARGUMENT
05827 * -AUBO_BAD_STATE
05828 *
05829 * @throws arcs::common_interface::AuboException
05830 *
05831 * @par JSON-RPC request example
05832 * {"jsonrpc":"2.0","method":"rob1.MotionControl.moveSpiral",
05833 * "params":[{"spiral":0.005,"helix":0.005,"angle":18.84,"plane":0,"frame":[0,0,0,0,0]},0,0.25,1.2,0],"id":1}
05834 *
05835 * @par JSON-RPC response example
05836 * {"id":1,"jsonrpc":"2.0","result":0}
05837 * \endenglish
05838 */
05839 int moveSpiral(const SpiralParameters &param, double blend_radius, double v,
05840               double a, double t);
05841
05842 /**
05843 * @ingroup MotionControl
05844 * \chinese
05845 *
05846 *
05847 * @return
05848 *
05849 * @throws arcs::common_interface::AuboException

```

```

05850 *
05851 * @par Python
05852 * getLookAheadSize(self: pyaubo_sdk.MotionControl) -> int
05853 *
05854 * @par Lua
05855 * getLookAheadSize() -> number
05856 *
05857 * @par JSON-RPC
05858 * {"jsonrpc":"2.0","method":"rob1.MotionControl.getLookAheadSize","params":[],"id":1}
05859 *
05860 * @par JSON-RPC
05861 * {"id":1,"jsonrpc":"2.0","result":1}
05862 *
05863 * \endchinese
05864 * \english
05865 * Get look-ahead segment size
05866 *
05867 * @return Returns the look-ahead segment size
05868 *
05869 * @throws arcs::common_interface::AuboException
05870 *
05871 * @par Python function prototype
05872 * getLookAheadSize(self: pyaubo_sdk.MotionControl) -> int
05873 *
05874 * @par Lua function prototype
05875 * getLookAheadSize() -> number
05876 *
05877 * @par JSON-RPC request example
05878 * {"jsonrpc":"2.0","method":"rob1.MotionControl.getLookAheadSize","params":[],"id":1}
05879 *
05880 * @par JSON-RPC response example
05881 * {"id":1,"jsonrpc":"2.0","result":1}
05882 *
05883 * \endenglish
05884 */
05885 int getLookAheadSize();
05886
05887 /**
05888 * @ingroup MotionControl
05889 * \chinese
05890 *
05891 * 1.      ,   ,   ,   ,   ,   ;
05892 * 2.      ,   ,   ,   ,   .
05893 *
05894 * @return 0
05895 *
05896 * @throws arcs::common_interface::AuboException
05897 *
05898 * @par Python
05899 * setLookAheadSize(self: pyaubo_sdk.MotionControl, arg0: int) -> int
05900 *
05901 * @par Lua
05902 * setLookAheadSize(eqradius: number) -> number
05903 *
05904 * @par JSON-RPC
05905 * {"jsonrpc":"2.0","method":"rob1.MotionControl.setLookAheadSize","params":[1],"id":1}
05906 *
05907 * @par JSON-RPC
05908 * {"id":1,"jsonrpc":"2.0","result":0}
05909 * \endchinese
05910 * \english
05911 * Set look-ahead segment size
05912 * 1. For tasks requiring high speed smoothness, such as CNC machining,
05913 *    gluing, welding, etc., a longer look-ahead segment size can provide
05914 *    better speed planning and smoother motion.
05915 * 2. For fast-response tasks such as picking, a shorter look-ahead segment
05916 *    size is preferred to improve response speed, but may cause large speed
05917 *    fluctuations due to insufficiently timely path feeding.
05918 *
05919 * @return Returns 0 on success
05920 *
05921 * @throws arcs::common_interface::AuboException
05922 *
05923 * @par Python function prototype
05924 * setLookAheadSize(self: pyaubo_sdk.MotionControl, arg0: int) -> int
05925 *
05926 * @par Lua function prototype
05927 * setLookAheadSize(eqradius: number) -> number
05928 *
05929 * @par JSON-RPC request example
05930 * {"jsonrpc":"2.0","method":"rob1.MotionControl.setLookAheadSize","params":[1],"id":1}
05931 *
05932 * @par JSON-RPC response example
05933 * {"id":1,"jsonrpc":"2.0","result":0}
05934 * \endenglish
05935 */
05936 int setLookAheadSize(int size);

```

```

05937
05938 /**
05939  * @ingroup MotionControl
05940  * \chinese
05941  *
05942  *
05943  * @param params Json
05944  * {
05945  *   "frequency": <num>,
05946  *   "amplitude": {<num>,<num>}
05947  * }
05948  *
05949  * @return 0
05950  *
05951  * @throws arcs::common_interface::AuboException
05952  * \endchinese
05953  * \english
05954  * Update frequency and amplitude during weaving process
05955  *
05956  * @param params Json string used to define weaving parameters
05957  * {
05958  *   "frequency": <num>,
05959  *   "amplitude": {<num>,<num>}
05960  * }
05961  *
05962  * @return Returns 0 on success
05963  *
05964  * @throws arcs::common_interface::AuboException
05965  * \endenglish
05966  */
05967 int weaveUpdateParameters(const std::string &params);
05968
05969 /**
05970  * @ingroup MotionControl
05971  * \chinese
05972  *
05973  *
05974  * @param stiffness [0 -> 1]
05975  * @return
05976  *
05977  * @throws arcs::common_interface::AuboException
05978  *
05979  * @par Python
05980  * enableJointSoftServo(self: pyaubo_sdk.RobotConfig, arg0: List[float]) ->
05981  * int
05982  *
05983  * @par Lua
05984  * enableJointSoftServo(stiffness: table) -> int
05985  *
05986  * @par JSON-RPC
05987  * {"jsonrpc":"2.0","method":"rob1.RobotConfig.enableJointSoftServo","params":[[0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0]],"id":1}
05988  *
05989  * @par JSON-RPC
05990  * {"id":1,"jsonrpc":"2.0","result":0}
05991  * \endchinese
05992  * \english
05993  * Set joint current loop stiffness coefficient
05994  *
05995  * @param stiffness Stiffness coefficient for each joint, as a percentage [0
05996  * -> 1]. The larger the value, the stiffer the joint.
05997  * @return
05998  *
05999  * @throws arcs::common_interface::AuboException
06000  *
06001  * @par Python function prototype
06002  * enableJointSoftServo(self: pyaubo_sdk.RobotConfig, arg0: List[float]) ->
06003  * int
06004  *
06005  * @par Lua function prototype
06006  * enableJointSoftServo(stiffness: table) -> int
06007  *
06008  * @par JSON-RPC request example
06009  * {"jsonrpc":"2.0","method":"rob1.RobotConfig.enableJointSoftServo","params":[[0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0]],"id":1}
06010  *
06011  * @par JSON-RPC response example
06012  * {"id":1,"jsonrpc":"2.0","result":0}
06013  * \endenglish
06014  */
06015 int enableJointSoftServo(const std::vector<double> &stiffness);
06016
06017 /**
06018  * @ingroup MotionControl
06019  * \chinese
06020  *
06021  *
06022  * @return 0
06023  *

```

```

06024 * @throws arcs::common_interface::AuboException
06025 *
06026 * @par Python
06027 * disableJointSoftServo(self: pyaubo_sdk.MotionControl) -> int
06028 *
06029 * @par Lua
06030 * disableJointSoftServo() -> number
06031 *
06032 * @par JSON-RPC
06033 * {"jsonrpc":"2.0","method":"rob1.MotionControl.disableJointSoftServo","params":[],"id":1}
06034 *
06035 * @par JSON-RPC
06036 * {"id":1,"jsonrpc":"2.0","result":1}
06037 * \endchinese
06038 * \english
06039 * Disable joint current loop stiffness coefficient
06040 *
06041 * @return Returns 0 on success, otherwise error code
06042 *
06043 * @throws arcs::common_interface::AuboException
06044 *
06045 * @par Python function prototype
06046 * disableJointSoftServo(self: pyaubo_sdk.MotionControl) -> int
06047 *
06048 * @par Lua function prototype
06049 * disableJointSoftServo() -> number
06050 *
06051 * @par JSON-RPC request example
06052 * {"jsonrpc":"2.0","method":"rob1.MotionControl.disableJointSoftServo","params":[],"id":1}
06053 *
06054 * @par JSON-RPC response example
06055 * {"id":1,"jsonrpc":"2.0","result":1}
06056 * \endenglish
06057 */
06058 int disableJointSoftServo();
06059
06060 /**
06061 * @ingroup MotionControl
06062 * \chinese
06063 *
06064 *
06065 * @return true false
06066 *
06067 * @throws arcs::common_interface::AuboException
06068 *
06069 * @par Python
06070 * isJointSoftServoEnabled(self: pyaubo_sdk.MotionControl) -> bool
06071 *
06072 * @par Lua
06073 * isJointSoftServoEnabled() -> boolean
06074 *
06075 * @par JSON-RPC
06076 * {"jsonrpc":"2.0","method":"rob1.MotionControl.isJointSoftServoEnabled","params":[],"id":1}
06077 *
06078 * @par JSON-RPC
06079 * {"id":1,"jsonrpc":"2.0","result":1}
06080 * \endchinese
06081 * \english
06082 * Determine whether the joint current loop stiffness coefficient is
06083 * enabled.
06084 *
06085 * @return Returns true if enabled, otherwise returns false.
06086 *
06087 * @throws arcs::common_interface::AuboException
06088 *
06089 * @par Python function prototype
06090 * isJointSoftServoEnabled(self: pyaubo_sdk.MotionControl) -> bool
06091 *
06092 * @par Lua function prototype
06093 * isJointSoftServoEnabled() -> boolean
06094 *
06095 * @par JSON-RPC request example
06096 * {"jsonrpc":"2.0","method":"rob1.MotionControl.isJointSoftServoEnabled","params":[],"id":1}
06097 *
06098 * @par JSON-RPC response example
06099 * {"id":1,"jsonrpc":"2.0","result":1}
06100 * \endenglish
06101 */
06102 bool isJointSoftServoEnabled();
06103
06104 /**
06105 * @ingroup MotionControl
06106 * \chinese
06107 *
06108 *
06109 * @param omega , .0
06110 * \n

```

```

06111 * @param zeta      ,      . 0
06112 * \n
06113 * @param level
06114 *      (1~3),      . 0
06115 * \n
06116 * @retval 0
06117 * @retval AUBO_BAD_STATE(1)
06118 * Normal ReducedMode Recovery
06119 * @retval AUBO_QUEUE_FULL(2)
06120 * @retval AUBO_BUSY(3)
06121 * @retval -AUBO_BAD_STATE(-1)
06122 *      task_id
06123 *      Running
06124 * @retval -AUBO_TIMEOUT(-4)
06125 *
06126 * @throws arcs::common_interface::AuboException
06127 *
06128 * @par Python
06129 * enableVibrationSuppress(self: pyaubo_sdk.MotionControl, arg0:
06130 * list[float], arg1: list[float], arg2: int) -> int
06131 *
06132 * @par Lua
06133 * enableVibrationSuppress(omega: table, zeta: table, level: number) -> nil
06134 *
06135 * @par JSON-RPC
06136 * {"json-
rpc":"2.0","method":"rob1.MotionControl.enableVibrationSuppress","params":[[0.5,0.5,0.5,0.5,0.5,0.5],[0.5,0.5,0.5,0.5,0.5,2],"id":1}
06137 *
06138 * @par JSON-RPC
06139 * {"id":1,"jsonrpc":"2.0","result":0}
06140 *
06141 * \endchinese
06142 */
06143 int enableVibrationSuppress(const std::vector<double> &omega,
06144                           const std::vector<double> &zeta, int level);
06145 /**
06146 * @ingroup MotionControl
06147 * \chinese
06148 *
06149 *
06150 * @return true false
06151 *
06152 * @throws arcs::common_interface::AuboException
06153 *
06154 * @par Python
06155 * disbaleVibrationSuppress(self: pyaubo_sdk.MotionControl) -> int
06156 *
06157 * @par Lua
06158 * disbaleVibrationSuppress() -> number
06159 *
06160 * @par JSON-RPC
06161 * {"jsonrpc":"2.0","method":"rob1.MotionControl.disbaleVibrationSuppress","params":[],"id":1}
06162 *
06163 * @par JSON-RPC
06164 * {"id":1,"jsonrpc":"2.0","result":1}
06165 * \endchinese
06166 */
06167 int disbaleVibrationSuppress();
06168 /**
06169 * @ingroup MotionControl
06170 * \chinese
06171 *
06172 *
06173 *
06174 * @param enable
06175 *
06176 * @throws arcs::common_interface::AuboException
06177 *
06178 * @par Lua
06179 * setTimeOptimalEnable() -> bool
06180 *
06181 * @par Lua
06182 * setTimeOptimalEnable(true)
06183 *
06184 * @par JSON-RPC
06185 * {"jsonrpc":"2.0","method":"rob1.MotionControl.setTimeOptimalEnable","params":[true],"id":1}
06186 *
06187 * @par JSON-RPC
06188 * {"id":1,"jsonrpc":"2.0","result":0}
06189 * \endchinese
06190 * \english
06191 *
06192 * @param enable
06193 *
06194 * @throws arcs::common_interface::AuboException
06195 *
06196 * @par Lua function prototype

```

```

06197 * setTimeOptimalEnable() -> number
06198 *
06199 * @par Lua example
06200 * setTimeOptimalEnable(true)
06201 *
06202 * @par JSON-RPC request example
06203 * {"jsonrpc":"2.0","method":"rob1.MotionControl.setTimeOptimalEnable","params":[true],"id":1}
06204 *
06205 * @par JSON-RPC response example
06206 * {"id":1,"jsonrpc":"2.0","result":0}
06207 * \endenglish
06208 */
06209 int setTimeOptimalEnable(bool enable);
06210
06211 /**
06212 * @ingroup MotionControl
06213 * \chinese
06214 *
06215 * true -
06216 * false -
06217 *
06218 * @return
06219 *
06220 * @throws arcs::common_interface::AuboException
06221 *
06222 * @par Python
06223 * isTimeOptimalEnabled(self: pyaubo_sdk.MotionControl) -> float
06224 *
06225 * @par Lua
06226 * isTimeOptimalEnabled() -> number
06227 *
06228 * @par Lua
06229 * num = isTimeOptimalEnabled()
06230 *
06231 * @par JSON-RPC
06232 * {"jsonrpc":"2.0","method":"rob1.MotionControl.isTimeOptimalEnabled","params":[],"id":1}
06233 *
06234 * @par JSON-RPC
06235 * {"id":1,"jsonrpc":"2.0","result":1.0}
06236 * \endchinese
06237 * \english
06238 * Get the time optimal algorithm state:
06239 * true - enable
06240 * false - disable
06241 *
06242 * @return Returns the time optimal algorithm state
06243 *
06244 * @throws arcs::common_interface::AuboException
06245 *
06246 * @par Python function prototype
06247 * isTimeOptimalEnabled(self: pyaubo_sdk.MotionControl) -> bool
06248 *
06249 * @par Lua function prototype
06250 * isTimeOptimalEnabled() -> boolean
06251 *
06252 * @par Lua example
06253 * flag = isTimeOptimalEnabled()
06254 *
06255 * @par JSON-RPC request example
06256 * {"jsonrpc":"2.0","method":"rob1.MotionControl.isTimeOptimalEnabled","params":[],"id":1}
06257 *
06258 * @par JSON-RPC response example
06259 * {"id":1,"jsonrpc":"2.0","result":true}
06260 * \endenglish
06261 */
06262 bool isTimeOptimalEnabled();
06263
06264 /**
06265 * @ingroup MotionControl
06266 * \chinese
06267 *
06268 * true -
06269 * false -
06270 *
06271 * @return
06272 *
06273 * @throws arcs::common_interface::AuboException
06274 *
06275 * @par Python
06276 * isTimeOptimalEnabled(self: pyaubo_sdk.MotionControl) -> bool
06277 *
06278 * @par Lua
06279 * isSupportedTimeOptimal() -> boolean
06280 *
06281 * @par Lua
06282 * num = isSupportedTimeOptimal()
06283 *

```

```

06284 * @par JSON-RPC
06285 * {"jsonrpc":"2.0","method":"rob1.MotionControl.isSupportedTimeOptimal","params":[],"id":1}
06286 *
06287 * @par JSON-RPC
06288 * {"id":1,"jsonrpc":"2.0","result":1.0}
06289 * \endchinese
06290 * \english
06291 * Check whether the time optimal algorithm is supported
06292 * true - supported
06293 * false - not supported
06294 *
06295 * @return Return whether the time-optimal algorithm is supported
06296 *
06297 * @throws arcs::common_interface::AuboException
06298 *
06299 * @par Python function prototype
06300 * isSupportedTimeOptimal(self: pyaubo_sdk.MotionControl) -> bool
06301 *
06302 * @par Lua function prototype
06303 * isSupportedTimeOptimal() -> boolean
06304 *
06305 * @par Lua example
06306 * flag = isSupportedTimeOptimal()
06307 *
06308 * @par JSON-RPC request example
06309 * {"jsonrpc":"2.0","method":"rob1.MotionControl.isSupportedTimeOptimal","params":[],"id":1}
06310 *
06311 * @par JSON-RPC response example
06312 * {"id":1,"jsonrpc":"2.0","result":true}
06313 * \endenglish
06314 */
06315 bool isSupportedTimeOptimal();
06316
06317 /**
06318 * @ingroup MotionControl
06319 * \chinese
06320 * TCP
06321 *
06322 * @param v TCP m/s /
06323 *
06324 * @throws arcs::common_interface::AuboException
06325 *
06326 * @par Lua
06327 * setTcpMaxLinearVelocity(number) -> number
06328 *
06329 * @par Lua
06330 * setTcpMaxLinearVelocity(0.5) -- TCP 0.5 /
06331 *
06332 * @par Python
06333 * setTcpMaxLinearVelocity(v: float) -> int
06334 *
06335 * @par Python
06336 * setTcpMaxLinearVelocity(0.5) -- TCP 0.5 /
06337 *
06338 * @par JSON-RPC
06339 * {"jsonrpc":"2.0","method":"rob1.MotionControl.setTcpMaxLinearVelocity","params":[0.5],"id":1}
06340 *
06341 * @par JSON-RPC
06342 * {"id":1,"jsonrpc":"2.0","result":0}
06343 * \endchinese
06344 * \english
06345 * Set TCP maximum linear velocity
06346 *
06347 * @param v TCP maximum linear velocity value, unit is m/s
06348 *
06349 * @throws arcs::common_interface::AuboException
06350 *
06351 * @par Lua function prototype
06352 * setTcpMaxLinearVelocity(number) -> number
06353 *
06354 * @par Lua example
06355 * setTcpMaxLinearVelocity(0.5) -- Set TCP maximum linear velocity to 0.5
06356 * meters per second
06357 *
06358 * @par Python function prototype
06359 * setTcpMaxLinearVelocity(v: float) -> int
06360 *
06361 * @par Python example
06362 * setTcpMaxLinearVelocity(0.5) -- Set TCP maximum linear velocity to 0.5
06363 *
06364 * @par JSON-RPC request example
06365 * {"jsonrpc":"2.0","method":"rob1.MotionControl.setTcpMaxLinearVelocity","params":[0.5],"id":1}
06366 *
06367 * @par JSON-RPC response example
06368 * {"id":1,"jsonrpc":"2.0","result":0}
06369 * \endenglish
06370 */

```



```

06371 int setTcpMaxLinearVelocity(double v);
06372
06373 /**
06374  * @ingroup MotionControl
06375  * \chinese
06376  * TCP
06377  *
06378  * @return TCP m/s /
06379  *
06380  * @throws arcs::common_interface::AuboException
06381  *
06382  * @par Lua
06383  * getTcpMaxLinearVelocity() -> number
06384  *
06385  * @par Lua
06386  * local v = getTcpMaxLinearVelocity() -- TCP
06387  *
06388  * @par Python
06389  * getTcpMaxLinearVelocity() -> float
06390  *
06391  * @par Python
06392  * v = getTcpMaxLinearVelocity() # TCP
06393  *
06394  * @par JSON-RPC
06395  * {"jsonrpc":"2.0","method":"rob1.MotionControl.getTcpMaxLinearVelocity","params":[],"id":1}
06396  *
06397  * @par JSON-RPC
06398  * {"id":1,"jsonrpc":"2.0","result":0.5}
06399  * \endchinese
06400  * \english
06401  * Get TCP maximum linear velocity
06402  *
06403  * @return Current TCP maximum linear velocity value, unit is m/s
06404  *
06405  * @throws arcs::common_interface::AuboException
06406  *
06407  * @par Lua function prototype
06408  * getTcpMaxLinearVelocity() -> number
06409  *
06410  * @par Lua example
06411  * local v = getTcpMaxLinearVelocity() -- Get TCP maximum linear velocity
06412  *
06413  * @par Python function prototype
06414  * getTcpMaxLinearVelocity() -> float
06415  *
06416  * @par Python example
06417  * v = getTcpMaxLinearVelocity() # Get TCP maximum linear velocity
06418  *
06419  * @par JSON-RPC request example
06420  * {"jsonrpc":"2.0","method":"rob1.MotionControl.getTcpMaxLinearVelocity","params":[],"id":1}
06421  *
06422  * @par JSON-RPC response example
06423  * {"id":1,"jsonrpc":"2.0","result":0.5}
06424  * \endenglish
06425  */
06426 double getTcpMaxLinearVelocity();
06427
06428 /**
06429  * @ingroup MotionControl
06430  * \chinese
06431  * TCP
06432  *
06433  * setTcpMaxLinearVelocity TCP
06434  *
06435  * @return 0
06436  *
06437  * @throws arcs::common_interface::AuboException
06438  *
06439  * @par Lua
06440  * resetTcpMaxLinearVelocity() -> number
06441  *
06442  * @par Lua
06443  * resetTcpMaxLinearVelocity() -- TCP
06444  *
06445  * @par Python
06446  * resetTcpMaxLinearVelocity() -> int
06447  *
06448  * @par Python
06449  * resetTcpMaxLinearVelocity() -- TCP
06450  *
06451  * @par JSON-RPC
06452  * {"jsonrpc":"2.0","method":"rob1.MotionControl.resetTcpMaxLinearVelocity","params":[],"id":1}
06453  *
06454  * @par JSON-RPC
06455  * {"id":1,"jsonrpc":"2.0","result":0}
06456  * \endchinese
06457  */

```

```

06458 * \english
06459 * Reset TCP maximum linear velocity
06460 *
06461 * Clear the TCP maximum linear velocity override set by
06462 * setTcpMaxLinearVelocity
06463 *
06464 * @return Returns 0 on success; error code on failure.
06465 *
06466 * @throws arcs::common_interface::AuboException
06467 *
06468 * @par Lua function prototype
06469 * resetTcpMaxLinearVelocity() -> number
06470 *
06471 * @par Lua example
06472 * resetTcpMaxLinearVelocity() -- Reset TCP maximum linear velocity to
06473 * the value in the safety configuration
06474 *
06475 * @par Python function prototype
06476 * resetTcpMaxLinearVelocity() -> int
06477 *
06478 * @par Python example
06479 * resetTcpMaxLinearVelocity() -- Reset TCP maximum linear velocity
06480 *
06481 * @par JSON-RPC request example
06482 * {"jsonrpc":"2.0","method":"rob1.MotionControl.resetTcpMaxLinearVelocity","params":[],"id":1}
06483 *
06484 * @par JSON-RPC response example
06485 * {"id":1,"jsonrpc":"2.0","result":0}
06486 * \endenglish
06487 */
06488 int resetTcpMaxLinearVelocity();
06489
06490 /**
06491 * @ingroup MotionControl
06492 * \chinese
06493 *
06494 *
06495 *          blending
06496 *
06497 *
06498 *
06499 *
06500 *
06501 * @return    0
06502 *
06503 * @throws arcs::common_interface::AuboException
06504 *
06505 * @par Lua
06506 * setEndPath() -> number
06507 *
06508 * @par Lua
06509 * setEndPath() --
06510 *
06511 * @par Python
06512 * setEndPath() -> int
06513 *
06514 * @par Python
06515 * setEndPath() #
06516 *
06517 * @par JSON-RPC
06518 * {"jsonrpc":"2.0","method":"rob1.MotionControl.setEndPath","params":[],"id":1}
06519 *
06520 * @par JSON-RPC
06521 * {"id":1,"jsonrpc":"2.0","result":0}
06522 * \endchinese
06523 *
06524 * \english
06525 * Set end path (terminate blending at current trajectory segment)
06526 *
06527 * Explicitly specifies the currently executing trajectory segment as the
06528 * boundary at which trajectory blending is terminated. The current
06529 * trajectory segment will be executed completely, but no velocity or
06530 * position blending will be applied with subsequent trajectory segments.
06531 *
06532 * This interface does not trigger an emergency stop and does not interrupt
06533 * the execution of the current trajectory segment. It only affects the
06534 * blending behavior between trajectory segments.
06535 *
06536 * @return Returns 0 on success; error code on failure.
06537 *
06538 * @throws arcs::common_interface::AuboException
06539 *
06540 * @par Lua function prototype
06541 * setEndPath() -> number
06542 *
06543 * @par Lua example
06544 * setEndPath() -- Terminate trajectory blending at the current segment

```

```

06545  *
06546  * @par Python function prototype
06547  * setEndPath() -> int
06548  *
06549  * @par Python example
06550  * setEndPath() # Terminate trajectory blending at the current segment
06551  *
06552  * @par JSON-RPC request example
06553  * {"jsonrpc":"2.0","method":"rob1.MotionControl.setEndPath","params":[],"id":1}
06554  *
06555  * @par JSON-RPC response example
06556  * {"id":1,"jsonrpc":"2.0","result":0}
06557  * \endenglish
06558  */
06559  int setEndPath();
06560
06561 protected:
06562  void *d_;
06563 };
06564 using MotionControlPtr = std::shared_ptr<MotionControl>;
06565 } // namespace common_interface
06566 } // namespace arcs
06567
06568 #endif // AUBO_SDK_MOTION_CONTROL_INTERFACE_H

```

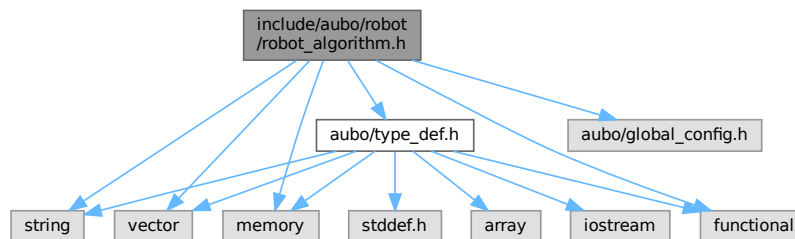
12.26 include/aubo/robot/robot_algorithm.h File Reference

```

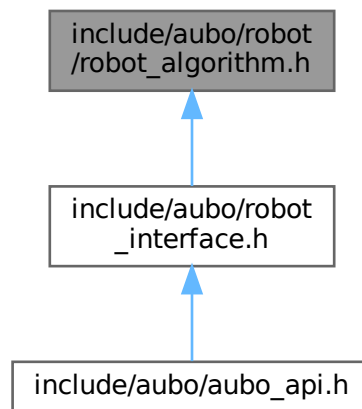
#include <string>
#include <vector>
#include <memory>
#include <functional>
#include <aubo/global_config.h>
#include <aubo/type_def.h>

```

Include dependency graph for robot_algorithm.h:



This graph shows which files directly or indirectly include this file:



Classes

- class [arcs::common_interface::RobotAlgorithm](#)

Namespaces

- namespace [arcs](#)
- namespace [arcs::common_interface](#)

Typedefs

- using [arcs::common_interface::RobotAlgorithmPtr](#) = std::shared_ptr<[RobotAlgorithm](#)>

12.26.1 Detailed Description

Definition in file [robot_algorithm.h](#).

12.27 robot_algorithm.h

[Go to the documentation of this file.](#)

```

00001 /** @file robot_algorithm.h
00002  * @brief
00003  */
00004 #ifndef AUBO_SDK_ROBOT_ALGORITHM_INTERFACE_H
00005 #define AUBO_SDK_ROBOT_ALGORITHM_INTERFACE_H
00006
00007 #include <string>
00008 #include <vector>
00009 #include <memory>
00010 #include <functional>
00011
00012 #include <aubo/global_config.h>
00013 #include <aubo/type_def.h>
00014
00015 namespace arcs {
00016 namespace common_interface {
00017
00018 // clang-format off
00019
00020 /**
00021  * @defgroup RobotAlgorithm RobotAlgorithm ( )
00022  * @ingroup RobotInterface
00023  * \chinese
00024  *
00025  * \endchinese
00026  * \english
00027  * Interfaces related to robot algorithms
00028  * \endenglish
00029  */
00030 class ARCS_ABI_EXPORT RobotAlgorithm
00031 {
00032 public:
00033     RobotAlgorithm();
00034     virtual ~RobotAlgorithm();
00035
00036     /**
00037      * @ingroup RobotAlgorithm
00038      * \chinese
00039      * ( )
00040      *
00041      * @param force
00042      * @param q
00043      * @return
00044      *
00045      * @throws arcs::common_interface::AuboException
00046      *
00047      * @par Python
00048      * calibrateTcpForceSensor(self: pyaubo_sdk.RobotAlgorithm, arg0:
00049      * List[List[float]], arg1: List[List[float]]) -> Tuple[List[float],
00050      * List[float], float, List[float])
00051      *
00052      * @par Lua
00053      * calibrateTcpForceSensor(force: table, q: table) -> table
00054      *
00055      * @par Lua
00056      * cal_table = calibrateTcpForceSensor({10.0,10.0,10.0,-1.2,-1.2,-1.2}, {3.083,1.227,1.098,0.670,-1.870,-0.397})
00057      *
00058      * \endchinese
00059      * \english
00060      * Force sensor calibration algorithm (three-point calibration method)
00061      *
00062      * @param force Force data
00063      * @param q Joint angles
00064      * @return Calibration result
00065      *
00066      * @throws arcs::common_interface::AuboException
00067      *
00068      * @par Python function prototype
00069      * calibrateTcpForceSensor(self: pyaubo_sdk.RobotAlgorithm, arg0:
00070      * List[List[float]], arg1: List[List[float]]) -> Tuple[List[float],
00071      * List[float], float, List[float])
00072      *
00073      * @par Lua function prototype
00074      * calibrateTcpForceSensor(force: table, q: table) -> table
00075      *
00076      * @par Lua example
00077      * cal_table = calibrateTcpForceSensor({10.0,10.0,10.0,-1.2,-1.2,-1.2}, {3.083,1.227,1.098,0.670,-1.870,-0.397})
00078      *
00079      * \endenglish
00080      */
00081     ForceSensorCalibResult calibrateTcpForceSensor(
00082         const std::vector<std::vector<double>> &forces,

```

```

00083         const std::vector<std::vector<double>> &poses);
00084
00085     /**
00086     * @ingroup RobotAlgorithm
00087     * \chinese
00088     * ( )
00089     * @param forces
00090     * @param poses
00091     * @return force_offset, com, mass, angle error
00092     *
00093     * @throws arcs::common_interface::AuboException
00094     * \endchinese
00095     * \english
00096     * Force sensor calibration algorithm (three-point calibration method)
00097     * @param forces
00098     * @param poses
00099     * @return force_offset, com, mass, angle error
00100     *
00101     * @throws arcs::common_interface::AuboException
00102     * \endenglish
00103     */
00104 ForceSensorCalibResultWithError calibrateTcpForceSensor2(
00105     const std::vector<std::vector<double>> &forces,
00106     const std::vector<std::vector<double>> &poses);
00107
00108     /**
00109     * @ingroup RobotAlgorithm
00110     * \chinese
00111     *
00112     *
00113     * @param force
00114     * @param poses
00115     * @param mass , : kg
00116     * @param cog , : m, (CoGx, CoGy, CoGz)
00117     * @return
00118     *
00119     * @throws arcs::common_interface::AuboException
00120     * \endchinese
00121     * \english
00122     * Force sensor offset calibration algorithm
00123     *
00124     * @param forces Force data
00125     * @param poses
00126     * @param m Mass, unit: kg
00127     * @param cog Center of gravity, unit: m, format (CoGx, CoGy, CoGz)
00128     * @return Calibration result
00129     *
00130     * @throws arcs::common_interface::AuboException
00131     * \endenglish
00132     */
00133 ResultWithError calibrateTcpForceSensor3(
00134     const std::vector<std::vector<double>> &forces,
00135     const std::vector<std::vector<double>> &poses, const double &mass,
00136     const std::vector<double>>&cog);
00137
00138     /**
00139     * @ingroup RobotAlgorithm
00140     * \chinese
00141     *
00142     *
00143     *
00144     *
00145     * @param data_file_no_payload
00146     * .csv 18 6 6 6
00147     * @param data_file_with_payload
00148     * .csv 18 6 6 6
00149     * @return
00150     *
00151     * @throws arcs::common_interface::AuboException
00152     *
00153     * @par Python
00154     * payloadIdentify(self: pyaubo_sdk.RobotAlgorithm, arg0: List[List[float]],
00155     * arg1: List[List[float]]) -> Tuple[List[float], List[float], float,
00156     * List[float]]
00157     *
00158     * @par Lua
00159     * payloadIdentify(data_with_payload: table, data_with_payload: table) ->
00160     * table
00161     * \endchinese
00162     * \english
00163     * Payload identification algorithm interface based on current
00164     *
00165     * Requires collecting position, velocity, and current data when running the excitation trajectory without load, as well as
    with load.
00166     *
00167     * @param data_file_no_payload
00168     * File path of joint data when running the excitation trajectory without load (.csv format), 18 columns in total: 6 joint

```

```

positions, 6 joint velocities, 6 joint currents
00169 * @param data_file_with_payload
00170 * File path of joint data when running the excitation trajectory with load (.csv format), 18 columns in total: 6 joint
positions, 6 joint velocities, 6 joint currents
00171 * @return Identification result
00172 *
00173 * @throws arcs::common_interface::AuboException
00174 *
00175 * @par Python function prototype
00176 * payloadIdentify(self: pyaubo_sdk.RobotAlgorithm, arg0: List[List[float]],
00177 * arg1: List[List[float]]) -> Tuple[List[float], List[float], float,
00178 * List[float]]
00179 *
00180 * @par Lua function prototype
00181 * payloadIdentify(data_with_payload: table, data_with_payload: table) ->
00182 * table
00183 * \endenglish
00184 */
00185 int payloadIdentify(const std::string &data_file_no_payload,
00186                    const std::string &data_file_with_payload);
00187
00188 /**
00189 * @ingroup RobotAlgorithm
00190 * \chinese
00191 *
00192 *
00193 *
00194 *
00195 * @param data
00196 * .csv 42 0
00197 * @return
00198 * \endchinese
00199 * \english
00200 * New version of payload identification algorithm interface based on current
00201 *
00202 * Requires collecting at least three points of position, velocity, acceleration, current, temperature, end sensor data, and
base data when running with load
00203 *
00204 * @param data
00205 * File path of joint data with load (.csv format), 42 columns in total, end sensor data and base data default to 0
00206 * @return Identification result
00207 * \endenglish
00208 */
00209 int payloadIdentify1(const std::string &file_name);
00210
00211 /**
00212 * @ingroup RobotAlgorithm
00213 * \chinese
00214 *
00215 * @return 0; 1; <0;
00216 *
00217 * @throws arcs::common_interface::AuboException
00218 *
00219 * @par JSON-RPC
00220 * {"jsonrpc":"2.0","method":"rob1.RobotAlgorithm.payloadCalculateFinished","params":[],"id":1}
00221 *
00222 * @par JSON-RPC
00223 * {"id":1,"jsonrpc":"2.0","result":0}
00224 *
00225 * \endchinese
00226 * \english
00227 * Whether payload identification calculation is finished
00228 * @return 0 if finished; 1 if in progress; <0 if failed;
00229 *
00230 * @throws arcs::common_interface::AuboException
00231 *
00232 * @par JSON-RPC request example
00233 * {"jsonrpc":"2.0","method":"rob1.RobotAlgorithm.payloadCalculateFinished","params":[],"id":1}
00234 *
00235 * @par JSON-RPC response example
00236 * {"id":1,"jsonrpc":"2.0","result":0}
00237 *
00238 * \endenglish
00239 */
00240 int payloadCalculateFinished();
00241
00242 /**
00243 * @ingroup RobotAlgorithm
00244 * \chinese
00245 *
00246 * @return
00247 *
00248 * @throws arcs::common_interface::AuboException
00249 *
00250 * @par JSON-RPC
00251 * {"jsonrpc":"2.0","method":"rob1.RobotAlgorithm.getPayloadIdentifyResult","params":[],"id":1}
00252 *

```

```

00253 * @par JSON-RPC
00254 * {"id":1,"jsonrpc":"2.0","result":[0.0,[],[],[]]}
00255 *
00256 * \endchinese
00257 * \english
00258 * Get the result of payload identification
00259 * @return
00260 *
00261 * @throws arcs::common_interface::AuboException
00262 *
00263 * @par JSON-RPC request example
00264 * {"jsonrpc":"2.0","method":"rob1.RobotAlgorithm.getPayloadIdentifyResult","params":[],"id":1}
00265 *
00266 * @par JSON-RPC response example
00267 * {"id":1,"jsonrpc":"2.0","result":[0.0,[],[],[]]}
00268 *
00269 * \endenglish
00270 */
00271 Payload getPayloadIdentifyResult();
00272
00273 /**
00274 * @ingroup RobotAlgorithm
00275 * \chinese
00276 *
00277 *
00278 * @param q
00279 * @param qd
00280 * @param qdd
00281 * @param temp
00282 * @return
00283 *
00284 * @throws arcs::common_interface::AuboException
00285 *
00286 * @par Python
00287 * frictionModelIdentify(self: pyaubo_sdk.RobotAlgorithm, arg0:
00288 * List[List[float]], arg1: List[List[float]], arg2: List[List[float]],
00289 * arg3: List[List[float]]) -> bool
00290 *
00291 * @par Lua
00292 * frictionModelIdentify(q: table, qd: table, qdd: table, temp: table) ->
00293 * boolean
00294 *
00295 * @par Lua
00296 * Identify_result = frictionModelIdentify({3.083,1.227,1.098,0.670,-1.870,-0.397},
00297 * {10.0,10.0,10.0,10.0,10.0,10.0},{20.0,20.0,20.0,20.0,20.0,20.0},{30.0,30.0,30.0,30.0,30.0,30.0})
00298 *
00299 * \endchinese
00300 * \english
00301 * Joint friction model identification algorithm interface
00302 *
00303 * @param q Joint positions
00304 * @param qd Joint velocities
00305 * @param qdd Joint accelerations
00306 * @param temp Joint temperatures
00307 * @return Whether identification succeeded
00308 *
00309 * @throws arcs::common_interface::AuboException
00310 *
00311 * @par Python function prototype
00312 * frictionModelIdentify(self: pyaubo_sdk.RobotAlgorithm, arg0:
00313 * List[List[float]], arg1: List[List[float]], arg2: List[List[float]],
00314 * arg3: List[List[float]]) -> bool
00315 *
00316 * @par Lua function prototype
00317 * frictionModelIdentify(q: table, qd: table, qdd: table, temp: table) ->
00318 * boolean
00319 *
00320 * @par Lua example
00321 * Identify_result = frictionModelIdentify({3.083,1.227,1.098,0.670,-1.870,-0.397},
00322 * {10.0,10.0,10.0,10.0,10.0,10.0},{20.0,20.0,20.0,20.0,20.0,20.0},{30.0,30.0,30.0,30.0,30.0,30.0})
00323 *
00324 * \endenglish
00325 */
00326 bool frictionModelIdentify(const std::vector<std::vector<double>> &q,
00327                           const std::vector<std::vector<double>> &qd,
00328                           const std::vector<std::vector<double>> &qdd,
00329                           const std::vector<std::vector<double>> &temp);
00330
00331 /**
00332 * @ingroup RobotAlgorithm
00333 * \chinese
00334 * ( TCP )
00335 * ( )
00336 *
00337 * @param q
00338 * @param type
00339 * @return ( )

```



```

00340 *
00341 * @throws arcs::common_interface::AuboException
00342 *
00343 * @par Python
00344 * calibWorkpieceCoordinatePara(self: pyaubo_sdk.RobotAlgorithm, arg0:
00345 * List[List[float]], arg1: int) -> Tuple[List[float], int]
00346 *
00347 * @par Lua
00348 * calibWorkpieceCoordinatePara(q: table, type: number) -> table, number
00349 *
00350 * @par Lua
00351 * coord_pose, coord_num = calibWorkpieceCoordinatePara({3.083,1.227,1.098,0.670,-1.870,-0.397},1)
00352 *
00353 * \endchinese
00354 * \english
00355 * Workpiece coordinate calibration algorithm interface (requires correct TCP offset set before calling)
00356 * Input multiple sets of joint angles and calibration type, output workpiece coordinate pose (relative to robot base)
00357 *
00358 * @param q Joint angles
00359 * @param type Calibration type
00360 * @return Calculation result (workpiece coordinate pose) and error code
00361 *
00362 * @throws arcs::common_interface::AuboException
00363 *
00364 * @par Python function prototype
00365 * calibWorkpieceCoordinatePara(self: pyaubo_sdk.RobotAlgorithm, arg0:
00366 * List[List[float]], arg1: int) -> Tuple[List[float], int]
00367 *
00368 * @par Lua function prototype
00369 * calibWorkpieceCoordinatePara(q: table, type: number) -> table, number
00370 *
00371 * @par Lua example
00372 * coord_pose, coord_num = calibWorkpieceCoordinatePara({3.083,1.227,1.098,0.670,-1.870,-0.397},1)
00373 *
00374 * \endenglish
00375 */
00376 ResultWithErrno calibWorkpieceCoordinatePara(
00377     const std::vector<std::vector<double> &q, int type);
00378
00379 /**
00380 * @ingroup RobotAlgorithm
00381 * \chinese
00382 *
00383 *
00384 * @param q
00385 * @param torqs
00386 * @return
00387 *
00388 * @throws arcs::common_interface::AuboException
00389 *
00390 * @par Python
00391 * forwardDynamics(self: pyaubo_sdk.RobotAlgorithm, arg0: List[float], arg1:
00392 * List[float]) -> Tuple[List[float], int]
00393 *
00394 * @par Lua
00395 * forwardDynamics(q: table, torqs: table) -> table, number
00396 *
00397 * @par Lua
00398 * Dynamics, fk_result = forwardDynamics({3.083,1.227,1.098,0.670,-1.870,-0.397},{0.0,0.0,0.0,0.0,0.0,0.0})
00399 *
00400 * \endchinese
00401 * \english
00402 * Forward dynamics
00403 *
00404 * @param q Joint angles
00405 * @param torqs
00406 * @return Calculation result and error code
00407 *
00408 * @throws arcs::common_interface::AuboException
00409 *
00410 * @par Python function prototype
00411 * forwardDynamics(self: pyaubo_sdk.RobotAlgorithm, arg0: List[float], arg1:
00412 * List[float]) -> Tuple[List[float], int]
00413 *
00414 * @par Lua function prototype
00415 * forwardDynamics(q: table, torqs: table) -> table, number
00416 *
00417 * @par Lua example
00418 * Dynamics, fk_result = forwardDynamics({3.083,1.227,1.098,0.670,-1.870,-0.397},{0.0,0.0,0.0,0.0,0.0,0.0})
00419 *
00420 * \endenglish
00421 */
00422 ResultWithErrno forwardDynamics(const std::vector<double> &q,
00423     const std::vector<double> &torqs);
00424
00425 /**
00426 * @ingroup RobotAlgorithm

```

```

00427 * \chinese
00428 *     TCP
00429 *
00430 * @param q
00431 * @param torqs
00432 * @param tcp_offset TCP
00433 * @return forwardDynamics
00434 *
00435 * @throws arcs::common_interface::AuboException
00436 *
00437 * @par Python
00438 * forwardDynamics(self: pyaubo_sdk.RobotAlgorithm, arg0: List[float], arg1:
00439 * List[float], arg2: List[float]) -> Tuple[List[float], int]
00440 *
00441 * @par Lua
00442 * forwardDynamics1(q: table, torqs: table, tcp_offset: table) -> table, number
00443 *
00444 * @par Lua
00445 * Dynamics, fk_result =
forwardDynamics1({3.083,1.227,1.098,0.670,-1.870,-0.397},{0.0,0.0,0.0,0.0,0.0,0.0},{0.0,0.13201,0.03879,0,0,0})
00446 *
00447 * \endchinese
00448 * \english
00449 * Forward dynamics based on the given TCP offset
00450 *
00451 * @param q Joint angles
00452 * @param torqs
00453 * @param tcp_offset TCP offset
00454 * @return Calculation result and error code, same as forwardDynamics
00455 *
00456 * @throws arcs::common_interface::AuboException
00457 *
00458 * @par Python function prototype
00459 * forwardDynamics(self: pyaubo_sdk.RobotAlgorithm, arg0: List[float], arg1:
00460 * List[float], arg2: List[float]) -> Tuple[List[float], int]
00461 *
00462 * @par Lua function prototype
00463 * forwardDynamics1(q: table, torqs: table, tcp_offset: table) -> table, number
00464 *
00465 * @par Lua example
00466 * Dynamics, fk_result =
forwardDynamics1({3.083,1.227,1.098,0.670,-1.870,-0.397},{0.0,0.0,0.0,0.0,0.0,0.0},{0.0,0.13201,0.03879,0,0,0})
00467 *
00468 * \endenglish
00469 */
00470 ResultWithErrno forwardDynamics1(const std::vector<double> &q,
00471                                const std::vector<double> &torqs,
00472                                const std::vector<double> &tcp_offset);
00473
00474 /**
00475 * @ingroup RobotAlgorithm
00476 * \chinese
00477 *     , TCP ( setTcpOffset )
00478 *     TCP
00479 *
00480 * @param q
00481 * @return TCP
00482 *
00483 * 0 -
00484 * -1 - ( )
00485 * -5 - ( )
00486 * @throws arcs::common_interface::AuboException
00487 *
00488 * @par Python
00489 * forwardKinematics(self: pyaubo_sdk.RobotAlgorithm, arg0: List[float]) ->
00490 * Tuple[List[float], int]
00491 *
00492 * @par Lua
00493 * forwardKinematics(q: table) -> table, number
00494 *
00495 * @par Lua
00496 * pose, fk_result = forwardKinematics({3.083,1.227,1.098,0.670,-1.870,-0.397})
00497 *
00498 * @par JSON-RPC
00499 *
{"jsonrpc":"2.0","method":"rob1.RobotAlgorithm.forwardKinematics","params":[[3.083688522170976,1.2273215976885394,1.098072739631141,0.
1.870715392248607,-0.39708546603119627]],"id":1}
00500 *
00501 * @par JSON-RPC
00502 *
{"id":1,"jsonrpc":"2.0","result":[[0.7137448715395925,0.08416057568819092,0.6707994191515292,2.4599818776908724,0.4789772388601265,1.6189
00503 * \endchinese
00504 * \english
00505 * Forward kinematics, based on the activated TCP offset (the most recently set via setTcpOffset)
00506 * Input joint angles, output TCP pose
00507 *
00508 * @param q Joint angles

```

```

00509     * @return TCP pose and whether the result is valid
00510     * The first parameter of the return value is the forward kinematics result, the second is the error code. Error codes are
    as follows:
00511     * 0 - Success
00512     * -1 - The status of the robot is incorrect (not initialized yet, you can try calling it again)
00513     * -5 - The input joint angles is invalid (dimension error)
00514     * @throws arcs::common_interface::AuboException
00515     *
00516     * @par Python function prototype
00517     * forwardKinematics(self: pyaubo_sdk.RobotAlgorithm, arg0: List[float]) ->
00518     * Tuple[List[float], int]
00519     *
00520     * @par Lua function prototype
00521     * forwardKinematics(q: table) -> table, number
00522     *
00523     * @par Lua example
00524     * pose, fk_result = forwardKinematics({3.083,1.227,1.098,0.670,-1.870,-0.397})
00525     *
00526     * @par JSON-RPC request example
00527     *
    {"jsonrpc":"2.0","method":"rob1.RobotAlgorithm.forwardKinematics","params":[[3.083688522170976,1.2273215976885394,1.098072739631141,0.
    1.870715392248607,-0.39708546603119627]], "id":1}
00528     *
00529     * @par JSON-RPC response example
00530     *
    {"id":1,"jsonrpc":"2.0","result":[[0.7137448715395925,0.08416057568819092,0.6707994191515292,2.4599818776908724,0.4789772388601265,1.618963
    00531     * \endenglish
00532     */
00533     ResultWithErrno forwardKinematics(const std::vector<double> &q);
00534
00535     /**
00536     * @ingroup RobotAlgorithm
00537     * \chinese
00538     *
00539     *     TCP
00540     * @param q
00541     * @param tcp_offset tcp
00542     * @return TCP
00543     *
00544     * 0 -
00545     * -1 - ( )
00546     * -5 - tcp ( )
00547     * @throws arcs::common_interface::AuboException
00548     *
00549     * @par Python
00550     * forwardKinematics(self: pyaubo_sdk.RobotAlgorithm, arg0: List[float],
00551     * arg1: List[float]) -> Tuple[List[float], int]
00552     *
00553     * @par Lua
00554     * forwardKinematics1(q: table, tcp_offset: table) -> table, number
00555     *
00556     * @par Lua
00557     * pose, fk_result = forwardKinematics1({3.083,1.227,1.098,0.670,-1.870,-0.397},{0.0,0.13201,0.03879,0.0,0})
00558     *
00559     * @par JSON-RPC
00560     *
    {"jsonrpc":"2.0","method":"rob1.RobotAlgorithm.forwardKinematics1","params":[[3.083688522170976,1.2273215976885394,1.098072739631141,0.
    1.870715392248607,-0.39708546603119627],[0.0,
00561     * 0.13201,0.03879,0.0,0]], "id":1}
00562     *
00563     * @par JSON-RPC
00564     *
    {"id":1,"jsonrpc":"2.0","result":[[0.7137636726659518,0.0837705432006433,0.6710022027216355,2.459981877690872,0.4789772388601267,1.618963
00565     *
00566     * @since 0.24.1
00567     * \endchinese
00568     * \english
00569     * Forward kinematics
00570     * Input joint angles, output TCP pose
00571     *
00572     * @param q Joint angles
00573     * @param tcp_offset TCP offset
00574     * @return TCP pose and whether the result is valid
00575     * The first parameter of the return value is the forward kinematics result, the second is the error code. Error codes are
    as follows:
00576     * 0 - Success
00577     * -1 - The status of the robot is incorrect (not initialized yet, you can try calling it again)
00578     * -5 - The input joint angles or tcp offset is invalid (dimension error)
00579     * @throws arcs::common_interface::AuboException
00580     *
00581     * @par Python function prototype
00582     * forwardKinematics(self: pyaubo_sdk.RobotAlgorithm, arg0: List[float],
00583     * arg1: List[float]) -> Tuple[List[float], int]
00584     *
00585     * @par Lua function prototype
00586     * forwardKinematics1(q: table, tcp_offset: table) -> table, number
00587     *

```

```

00588     * @par Lua example
00589     * pose, fk_result = forwardKinematics1({3.083,1.227,1.098,0.670,-1.870,-0.397},{0.0,0.13201,0.03879,0,0,0})
00590     *
00591     * @par JSON-RPC request example
00592     *
00593     {"jsonrpc":"2.0","method":"rob1.RobotAlgorithm.forwardKinematics1","params":[[3.083688522170976,1.2273215976885394,1.098072739631141,0.
00594     1.870715392248607,-0.39708546603119627],[0.0,
00595     * 0.13201,0.03879,0,0,0]],"id":1}
00596     *
00597     * @par JSON-RPC response example
00598     *
00599     {"id":1,"jsonrpc":"2.0","result":[[0.7137636726659518,0.0837705432006433,0.6710022027216355,2.459981877690872,0.4789772388601267,1.618963
00600     *
00601     * @since 0.24.1
00602     * \endenglish
00603     */
00604     ResultWithErrno forwardKinematics1(const std::vector<double> &q,
00605     const std::vector<double> &tcp_offset);
00606
00607     /**
00608     * @ingroup RobotAlgorithm
00609     * \chinese
00610     * ( TCP )
00611     *
00612     * @param q
00613     * @return
00614     *
00615     * 0 -
00616     * -1 - ( )
00617     * -5 - ( )
00618     *
00619     * @throws arcs::common_interface::AuboException
00620     *
00621     * @par Lua
00622     * forwardToolKinematics(q: table) -> table, number
00623     *
00624     * @par Lua
00625     * pose, fk_result = forwardToolKinematics({3.083,1.227,1.098,0.670,-1.870,-0.397})
00626     *
00627     * @par JSON-RPC
00628     *
00629     {"jsonrpc":"2.0","method":"rob1.RobotAlgorithm.forwardToolKinematics","params":[[3.083688522170976,1.2273215976885394,1.0980727396311
00630     1.870715392248607,-0.39708546603119627]],"id":1}
00631     *
00632     * @par JSON-RPC
00633     *
00634     {"id":1,"jsonrpc":"2.0","result":[[0.5881351149440136,0.05323734739426938,0.623922550656701,2.4599818776908724,0.4789772388601265,1.61896
00635     * \endchinese
00636     * \english
00637     * Forward kinematics (ignoring TCP offset)
00638     *
00639     * @param q Joint angles
00640     * @return Flange center pose and whether the result is valid
00641     * The first parameter of the return value is the forward kinematics result, the second is the error code. Error codes are
00642     as follows:
00643     * 0 - Success
00644     * -1 - The status of the robot is incorrect (not initialized yet, you can try calling it again)
00645     * -5 - The input joint angles is invalid (dimension error)
00646     *
00647     * @par Lua function prototype
00648     * forwardToolKinematics(q: table) -> table, number
00649     *
00650     * @par Lua example
00651     * pose, fk_result = forwardToolKinematics({3.083,1.227,1.098,0.670,-1.870,-0.397})
00652     *
00653     * @throws arcs::common_interface::AuboException
00654     *
00655     * @par JSON-RPC request example
00656     *
00657     {"jsonrpc":"2.0","method":"rob1.RobotAlgorithm.forwardToolKinematics","params":[[3.083688522170976,1.2273215976885394,1.0980727396311
00658     1.870715392248607,-0.39708546603119627]],"id":1}
00659     *
00660     * @par JSON-RPC response example
00661     *
00662     {"id":1,"jsonrpc":"2.0","result":[[0.5881351149440136,0.05323734739426938,0.623922550656701,2.4599818776908724,0.4789772388601265,1.61896
00663     * \endenglish
00664     */
00665     ResultWithErrno forwardToolKinematics(const std::vector<double> &q);
00666
00667     /**
00668     * @ingroup RobotAlgorithm
00669     * \chinese
00670     * , TCP ( setTcpOffset )
00671     *
00672     * @param q
00673     * @return

```

```

00665 *
00666 * 0 -
00667 * -1 - ( )
00668 * -5 - ( )
00669 * @throws arcs::common_interface::AuboException
00670 *
00671 * @par Python
00672 * forwardKinematicsAll(self: pyaubo_sdk.RobotAlgorithm, arg0: List[float]) ->
00673 * Tuple[List[List[float]], int]
00674 *
00675 * @par Lua
00676 * forwardKinematicsAll(q: table) -> table, number
00677 *
00678 * @par Lua
00679 * poses, fk_result = forwardKinematicsAll({3.083,1.227,1.098,0.670,-1.870,-0.397})
00680 *
00681 * @par JSON-RPC
00682 * {"jsonrpc":"2.0","method":"rob1.RobotAlgorithm.forwardKinematicsAll","params":{"q":[-7.32945e-11,-
0.261799,1.74533,0.436332,1.5708,-2.14136e-10]},"id":1}
00683 *
00684 * @par JSON-RPC
00685 * {"id":1,"jsonrpc":"2.0","result":[[[0.0010004,0.0,0.122,0.0,0.0,3.141592653589793],
00686 * [0.0010003999910823934,-0.12166795404953117,0.12205392567412496,1.5690838552993878,-
1.3089969602251492,0.0016541204887245322],
00687 * [0.10659809449330016,-0.12166124765791932,0.5161518260534821,-1.5718540206872664,0.43633111081725523,-
0.002370419930605744],
00688 * [0.4482255342315581,-0.1226390142692,0.3568490758201224,-0.000788980482890453,1.5665204619271216,-
1.5719618592940798],
00689 * [0.5519603828181741,-
0.12282266347465487,0.35639753697117343,1.5707938201843654,0.0042721909279596635,1.569044569428874],
00690 * [0.5513243939877184,-0.12401224959696328,0.2615193425222568,-
3.1349118101994096,0.004272190926529066,1.569044569643007],
00691 * [0.7494883076798231,-0.025647479212994567,-0.04023458911333461,-
3.1349118101994096,0.004272190926529066,1.569044569643007]],0]}
00692 * \endchinese
00693 * \english
00694 * Forward kinematics, based on the activated TCP offset (the most recently set via setTcpOffset)
00695 * Input joint angles, output links poses
00696 *
00697 * @param q Joint angles
00698 * @return links poses and whether the result is valid
00699 * The first parameter of the return value is the forward kinematics result, the second is the error code. Error codes are
as follows:
00700 * 0 - Success
00701 * -1 - The status of the robot is incorrect (not initialized yet, you can try calling it again)
00702 * -5 - The input joint angles is invalid (dimension error)
00703 * @throws arcs::common_interface::AuboException
00704 *
00705 * @par Python function prototype
00706 * forwardKinematicsAll(self: pyaubo_sdk.RobotAlgorithm, arg0: List[float]) ->
00707 * Tuple[List[List[float]], int]
00708 *
00709 * @par Lua function prototype
00710 * forwardKinematicsAll(q: table) -> table, number
00711 *
00712 * @par Lua example
00713 * pose, fk_result = forwardKinematicsAll({3.083,1.227,1.098,0.670,-1.870,-0.397})
00714 *
00715 * @par JSON-RPC request example
00716 * {"jsonrpc":"2.0","method":"rob1.RobotAlgorithm.forwardKinematicsAll","params":{"q":[-7.32945e-11,-
0.261799,1.74533,0.436332,1.5708,-2.14136e-10]},"id":1}
00717 *
00718 * @par JSON-RPC response example
00719 * {"id":1,"jsonrpc":"2.0","result":[[[0.0010004,0.0,0.122,0.0,0.0,3.141592653589793],
00720 * [0.0010003999910823934,-0.12166795404953117,0.12205392567412496,1.5690838552993878,-
1.3089969602251492,0.0016541204887245322],
00721 * [0.10659809449330016,-0.12166124765791932,0.5161518260534821,-1.5718540206872664,0.43633111081725523,-
0.002370419930605744],
00722 * [0.4482255342315581,-0.1226390142692,0.3568490758201224,-0.000788980482890453,1.5665204619271216,-
1.5719618592940798],
00723 * [0.5519603828181741,-
0.12282266347465487,0.35639753697117343,1.5707938201843654,0.0042721909279596635,1.569044569428874],
00724 * [0.5513243939877184,-0.12401224959696328,0.2615193425222568,-
3.1349118101994096,0.004272190926529066,1.569044569643007],
00725 * [0.7494883076798231,-0.025647479212994567,-0.04023458911333461,-
3.1349118101994096,0.004272190926529066,1.569044569643007]],0]}
00726 * \endenglish
00727 */
00728 ResultWithErrno1 forwardKinematicsAll(const std::vector<double> &q);
00729
00730 /**
00731 * @ingroup RobotAlgorithm
00732 * \chinese
00733 *
00734 * TCP
00735 *
00736 * @param qnear

```

```

00737 * @param pose TCP
00738 * @return
00739 *
00740 * 0 -
00741 * -1 - ( TCP )
00742 * -5 - TCP ( )
00743 * -23 -
00744 * -24 -
00745 * -25 -
00746 * -26 -
00747 * -27 -
00748 * -28 -
00749 * 0, qnear
00750 *
00751 * @throws arcs::common_interface::AuboException
00752 *
00753 * @par Python
00754 * inverseKinematics(self: pyaubo_sdk.RobotAlgorithm, arg0: List[float],
00755 * arg1: List[float]) -> Tuple[List[float], int]
00756 *
00757 * @par Lua
00758 * inverseKinematics(qnear: table, pose: table) -> table, int
00759 *
00760 * @par Lua
00761 * joint_ik_result = inverseKinematics({0,0,0,0,0,0},{0.81665,-0.20419,0.43873,-3.135,0.004,1.569})
00762 *
00763 * @par JSON-RPC
00764 * {"json-
rpc":"2.0","method":"rob1.RobotAlgorithm.inverseKinematics","params":[[0,0,0,0,0],[0.71374,0.08417,0.6708,2.46,0.479,1.619]],"id":1}
00765 *
00766 * @par JSON-RPC
00767 * {"id":1,"jsonrpc":"2.0","result":[[3.083688522170976,1.2273215976885394,1.098072739631141,0.6705738810610149,-
1.870715392248607,-0.39708546603119627],0]}
00768 * \endchinese
00769 * \english
00770 * Inverse kinematics
00771 * Input TCP pose and reference joint angles, output joint angles
00772 *
00773 * @param qnear Reference joint angles
00774 * @param pose TCP pose
00775 * @return Joint angles and whether the inverse kinematics result is valid
00776 * The first parameter of the return value is the inverse kinematics result, the second is the error code. Error codes are
as follows:
00777 * 0 - Success
00778 * -1 - The status of the robot is incorrect (not initialized yet, you can try calling it again)
00779 * -5 - The input reference joint angles or tcp pose is invalid (dimension error)
00780 * -23 - Inverse kinematics calculation does not converge, calculation error
00781 * -24 - Inverse kinematics calculation exceeds robot limits
00782 * -25 - Inverse kinematics input configuration error
00783 * -26 - Inverse kinematics Jacobian calculation failed
00784 * -27 - Analytical solution exists but none satisfy the selection criteria
00785 * -28 - Unknown error in inverse kinematics
00786 * If the error code is not 0, the first parameter of the return value is the input reference joint angles qnear
00787 *
00788 * @throws arcs::common_interface::AuboException
00789 *
00790 * @par Python function prototype
00791 * inverseKinematics(self: pyaubo_sdk.RobotAlgorithm, arg0: List[float],
00792 * arg1: List[float]) -> Tuple[List[float], int]
00793 *
00794 * @par Lua function prototype
00795 * inverseKinematics(qnear: table, pose: table) -> table, int
00796 *
00797 * @par Lua example
00798 * joint_ik_result = inverseKinematics({0,0,0,0,0,0},{0.81665,-0.20419,0.43873,-3.135,0.004,1.569})
00799 *
00800 * @par JSON-RPC request example
00801 * {"json-
rpc":"2.0","method":"rob1.RobotAlgorithm.inverseKinematics","params":[[0,0,0,0,0],[0.71374,0.08417,0.6708,2.46,0.479,1.619]],"id":1}
00802 *
00803 * @par JSON-RPC response example
00804 * {"id":1,"jsonrpc":"2.0","result":[[3.083688522170976,1.2273215976885394,1.098072739631141,0.6705738810610149,-
1.870715392248607,-0.39708546603119627],0]}
00805 * \endenglish
00806 */
00807 ResultWithErrno inverseKinematics(const std::vector<double> &qnear,
00808 const std::vector<double> &pose);
00809
00810 /**
00811 * @ingroup RobotAlgorithm
00812 * \chinese
00813 *
00814 * TCP
00815 *
00816 * @param qnear
00817 * @param pose TCP
00818 * @param tcp_offset TCP

```

```

00819     * @return          inverseKinematics
00820     * @throws arcs::common_interface::AuboException
00821     *
00822     * @par Python
00823     * inverseKinematics1(self: pyaubo_sdk.RobotAlgorithm, arg0: List[float],
00824     * arg1: List[float], arg2: List[float]) -> Tuple[List[float], int]
00825     *
00826     * @par Lua
00827     * inverseKinematics1(qnear: table, pose: table, tcp_offset: table) -> table, int
00828     *
00829     * @par Lua
00830     * joint_ik_result =
inverseKinematics1({0,0,0,0,0,0},{0.81665,-0.20419,0.43873,-3.135,0.004,1.569},{0.04,-0.035,0.1,0,0,0})
00831     *
00832     * @par JSON-RPC
00833     * {"json-
rpc":"2.0","method":"rob1.RobotAlgorithm.inverseKinematics1","params":[[0,0,0,0,0,0],[0.71374,0.08417,0.6708,2.46,0.479,1.619],[0.0,
00834     * 0.13201,0.03879,0,0,0]],"id":1}
00835     *
00836     * @par JSON-RPC
00837     * {"id":1,"jsonrpc":"2.0","result":[[3.084454549595208,1.2278265883747776,1.0986586440159576,0.6708221281915528,-
1.8712459848518375,-0.3965111476861782],0]}
00838     * \endchinese
00839     * \english
00840     * Inverse kinematics
00841     * Input TCP pose and reference joint angles, output joint angles
00842     *
00843     * @param qnear Reference joint angles
00844     * @param pose TCP pose
00845     * @param tcp_offset TCP offset
00846     * @return Joint angles and whether the result is valid, same as inverseKinematics
00847     * @throws arcs::common_interface::AuboException
00848     *
00849     * @par Python function prototype
00850     * inverseKinematics1(self: pyaubo_sdk.RobotAlgorithm, arg0: List[float],
00851     * arg1: List[float], arg2: List[float]) -> Tuple[List[float], int]
00852     *
00853     * @par Lua function prototype
00854     * inverseKinematics1(qnear: table, pose: table, tcp_offset: table) -> table, int
00855     *
00856     * @par Lua example
00857     * joint_ik_result =
inverseKinematics1({0,0,0,0,0,0},{0.81665,-0.20419,0.43873,-3.135,0.004,1.569},{0.04,-0.035,0.1,0,0,0})
00858     *
00859     * @par JSON-RPC request example
00860     * {"json-
rpc":"2.0","method":"rob1.RobotAlgorithm.inverseKinematics1","params":[[0,0,0,0,0,0],[0.71374,0.08417,0.6708,2.46,0.479,1.619],[0.0,
00861     * 0.13201,0.03879,0,0,0]],"id":1}
00862     *
00863     * @par JSON-RPC response example
00864     * {"id":1,"jsonrpc":"2.0","result":[[3.084454549595208,1.2278265883747776,1.0986586440159576,0.6708221281915528,-
1.8712459848518375,-0.3965111476861782],0]}
00865     * \endenglish
00866     */
00867     ResultWithErrno inverseKinematics1(const std::vector<double> &qnear,
00868     const std::vector<double> &pose,
00869     const std::vector<double> &tcp_offset);
00870
00871     /**
00872     * @ingroup RobotAlgorithm
00873     * \chinese
00874     * , TCP
00875     *
00876     * @param pose TCP
00877     * @return
00878     * inverseKinematics
00879     *
00880     * @throws arcs::common_interface::AuboException
00881     *
00882     * @par JSON-RPC
00883     * {"json-
rpc":"2.0","method":"rob1.RobotAlgorithm.inverseKinematicsAll","params":[[0.71374,0.08417,0.6708,2.46,0.479,1.619]],"id":1}
00884     *
00885     * @par JSON-RPC
00886     * {"id":1,"jsonrpc":"2.0","result":[[[3.083688522170976,1.2273215976885394,1.098072739631141,0.6705738810610149,-
1.870715392248607,-0.39708546603119627],
00887     * [3.081056801097411,0.17985038037652645,-1.0991717292664145,-0.4806460200109001,-1.869182975312333,-
0.402066016835411],
00888     * [0.4090095277807992,-
0.1623365054641728,1.081775890307679,0.26993250263224805,0.9738255833642309,0.000572556627720845],
00889     * [0.4116449425067969,-1.1931664523907126,-1.0822709833775688,-
0.8665964106161371,0.9732141569888207,0.006484919654891586]],0]}
00890     * \endchinese
00891     * \english
00892     * Solve all inverse kinematics solutions based on the activated TCP offset
00893     *
00894     * @param pose TCP pose

```



```

00895     * @return Joint angles and whether the result is valid
00896     * The returned error code is the same as inverseKinematics
00897     *
00898     * @throws arcs::common_interface::AuboException
00899     *
00900     * @par JSON-RPC request example
00901     * {"json-
rpc":"2.0","method":"rob1.RobotAlgorithm.inverseKinematicsAll","params":[[0.71374,0.08417,0.6708,2.46,0.479,1.619]],"id":1}
00902     *
00903     * @par JSON-RPC response example
00904     * {"id":1,"jsonrpc":"2.0","result":[[[3.083688522170976,1.2273215976885394,1.098072739631141,0.6705738810610149,-
1.870715392248607,-0.39708546603119627],
00905     * [3.081056801097411,0.17985038037652645,-1.0991717292664145,-0.4806460200109001,-1.869182975312333,-
0.402066016835411],
00906     * [0.4090095277807992,-
0.1623365054641728,1.081775890307679,0.26993250263224805,0.9738255833642309,0.000572556627720845],
00907     * [0.4116449425067969,-1.1931664523907126,-1.0822709833775688,-
0.8665964106161371,0.9732141569888207,0.006484919654891586]],0]}
00908     * \endenglish
00909     */
00910     ResultWithErrno1 inverseKinematicsAll(const std::vector<double> &pose);
00911
00912     /**
00913     * @ingroup RobotAlgorithm
00914     * \chinese
00915     * , TCP
00916     *
00917     * @param pose TCP
00918     * @param tcp_offset TCP
00919     * @return inverseKinematicsAll
00920     * inverseKinematics
00921     *
00922     * @throws arcs::common_interface::AuboException
00923     *
00924     * @par JSON-RPC
00925     * {"json-
rpc":"2.0","method":"rob1.RobotAlgorithm.inverseKinematicsAll1","params":[[0.71374,0.08417,0.6708,2.46,0.479,1.619],[0.0,
00926     * 0.13201,0.03879,0,0,0]],"id":1}
00927     *
00928     * @par JSON-RPC
00929     * {"id":1,"jsonrpc":"2.0","result":[[[3.084454549595208,1.2278265883747776,1.0986586440159576,0.6708221281915528,-
1.8712459848518375,-0.3965111476861782],
00930     * [3.0818224058231602,0.17980369843203092,-1.0997576631122077,-0.48102131527371267,-1.8697135490338517,-
0.40149459722060593],
00931     * [0.40972960018231047,-
0.16226026285489026,1.0823403816496,0.2700204411869427,0.9734251963887868,0.0012903686498106507],
00932     * [0.41236549588802296,-1.193621392918341,-1.0828346680836718,-
0.8671097369314354,0.972815367289568,0.007206851371073478]],0]}
00933     * \endchinese
00934     * \english
00935     * Solve all inverse kinematics solutions based on the provided TCP offset
00936     *
00937     * @param pose TCP pose
00938     * @param tcp_offset TCP offset
00939     * @return Joint angles and whether the result is valid, same as inverseKinematicsAll
00940     * The returned error code is the same as inverseKinematics
00941     *
00942     * @throws arcs::common_interface::AuboException
00943     *
00944     * @par JSON-RPC request example
00945     * {"json-
rpc":"2.0","method":"rob1.RobotAlgorithm.inverseKinematicsAll1","params":[[0.71374,0.08417,0.6708,2.46,0.479,1.619],[0.0,
00946     * 0.13201,0.03879,0,0,0]],"id":1}
00947     *
00948     * @par JSON-RPC response example
00949     * {"id":1,"jsonrpc":"2.0","result":[[[3.084454549595208,1.2278265883747776,1.0986586440159576,0.6708221281915528,-
1.8712459848518375,-0.3965111476861782],
00950     * [3.0818224058231602,0.17980369843203092,-1.0997576631122077,-0.48102131527371267,-1.8697135490338517,-
0.40149459722060593],
00951     * [0.40972960018231047,-
0.16226026285489026,1.0823403816496,0.2700204411869427,0.9734251963887868,0.0012903686498106507],
00952     * [0.41236549588802296,-1.193621392918341,-1.0828346680836718,-
0.8671097369314354,0.972815367289568,0.007206851371073478]],0]}
00953     * \endenglish
00954     */
00955     ResultWithErrno1 inverseKinematicsAll1(
00956         const std::vector<double> &pose, const std::vector<double> &tcp_offset);
00957
00958     /**
00959     * @ingroup RobotAlgorithm
00960     * \chinese
00961     * ( TCP )
00962     *
00963     * @param qnear
00964     * @param pose
00965     * @return
00966

```



```

00967 * 0 -
00968 * -1 - ( )
00969 * -5 - ( )
00970 *
00971 * @throws arcs::common_interface::AuboException
00972 *
00973 * @par Lua
00974 * inverseToolKinematics(qnear: table, pose: table) -> table, int
00975 *
00976 * @par Lua
00977 * joint, ik_result = inverseToolKinematics({0,0,0,0,0},{0.58815,0.0532,0.62391,2.46,0.479,1.619})
00978 *
00979 * @par JSON-RPC
00980 * {"json-
rpc":"2.0","method":"rob1.RobotAlgorithm.inverseToolKinematics","params":[[0,0,0,0,0],[0.58815,0.0532,0.62391,2.46,0.479,1.619]],"id":1}
00981 *
00982 * @par JSON-RPC
00983 * {"id":1,"jsonrpc":"2.0","result":[[3.083609363838651,1.22736129158332,1.098095443698268,0.6705395395487186,-
1.8706605026855632,-0.39714507002376465],0]}
00984 * \endchinese
00985 * \english
00986 * Inverse kinematics (ignoring TCP offset)
00987 *
00988 * @param qnear Reference joint angles
00989 * @param pose Flange center pose
00990 * @return Joint angles and whether the result is valid
00991 * The first parameter of the return value is the inverse kinematics result, the second is the error code. Error codes are
as follows:
00992 * 0 - Success
00993 * -1 - The status of the robot is incorrect (not initialized yet, you can try calling it again)
00994 * -5 - The input reference joint angles or pose is invalid (dimension error)
00995 * @throws arcs::common_interface::AuboException
00996 *
00997 * @par Lua function prototype
00998 * inverseToolKinematics(qnear: table, pose: table) -> table, int
00999 *
01000 * @par Lua example
01001 * joint, ik_result = inverseToolKinematics({0,0,0,0,0},{0.58815,0.0532,0.62391,2.46,0.479,1.619})
01002 *
01003 * @par JSON-RPC request example
01004 * {"json-
rpc":"2.0","method":"rob1.RobotAlgorithm.inverseToolKinematics","params":[[0,0,0,0,0],[0.58815,0.0532,0.62391,2.46,0.479,1.619]],"id":1}
01005 *
01006 * @par JSON-RPC response example
01007 * {"id":1,"jsonrpc":"2.0","result":[[3.083609363838651,1.22736129158332,1.098095443698268,0.6705395395487186,-
1.8706605026855632,-0.39714507002376465],0]}
01008 * \endenglish
01009 */
01010 ResultWithErrno inverseToolKinematics(const std::vector<double> &qnear,
const std::vector<double> &pose);
01011
01012 /**
01013 * @ingroup RobotAlgorithm
01014 * \chinese
01015 * ( TCP )
01016 *
01017 * @param qnear
01018 * @param pose
01019 * @return
01020 *
01021 * @throws arcs::common_interface::AuboException
01022 *
01023 * @par JSON-RPC
01024 * {"json-
rpc":"2.0","method":"rob1.RobotAlgorithm.inverseToolKinematicsAll","params":[[0.58815,0.0532,0.62391,2.46,0.479,1.619]],"id":1}
01025 *
01026 * @par JSON-RPC
01027 * {"id":1,"jsonrpc":"2.0","result":[[[3.083609363838651,1.22736129158332,1.098095443698268,0.6705395395487186,-
1.8706605026855632,-0.39714507002376465],
01028 * [3.0809781797426523,0.17987122696706134,-1.0991932793263717,-0.4807053707530958,-1.8691282890274434,-
0.40212516672751814],
01029 * [0.40892195618737215,-
0.16235398607358653,1.081812753177426,0.27003586475871766,0.9738744130114284,0.00048462518316674287],
01030 * [0.41155633414333076,-1.1932173012004512,-1.082306542045813,-
0.8665312056504818,0.9732632365861417,0.0063958311601771175]],0]}
01031 * \endchinese
01032 * \english
01033 * Inverse kinematics (ignoring TCP offset)
01034 *
01035 * @param qnear Reference joint angles
01036 * @param pose Flange center pose
01037 * @return Joint angles and whether the result is valid
01038 *
01039 * @throws arcs::common_interface::AuboException
01040 *
01041 * @par JSON-RPC request example
01042 * {"json-
01043

```

```

rpc":"2.0","method":"rob1.RobotAlgorithm.inverseToolKinematicsAll","params":[[0.58815,0.0532,0.62391,2.46,0.479,1.619]],"id":1}
01044 *
01045 * @par JSON-RPC response example
01046 * {"id":1,"jsonrpc":"2.0","result":[[[3.083609363838651,1.22736129158332,1.098095443698268,0.6705395395487186,-
01047 * [3.0809781797426523,0.17987122696706134,-1.0991932793263717,-0.4807053707530958,-1.8691282890274434,-
01048 * 0.40212516672751814],
01049 * [0.40892195618737215,-
01050 * 0.16235398607358653,1.081812753177426,0.27003586475871766,0.9738744130114284,0.00048462518316674287],
01051 * [0.41155633414333076,-1.1932173012004512,-1.082306542045813,-
01052 * 0.8665312056504818,0.9732632365861417,0.0063958311601771175]],0]}
01053 * \endenglish
01054 */
01055 ResultWithErrno1 inverseToolKinematicsAll(const std::vector<double> &pose);
01056
01057 /**
01058 * @ingroup RobotAlgorithm
01059 * \chinese
01060 *
01061 * - LEFT( ) / RIGHT( )
01062 * - UP( ) / DOWN( )
01063 * - FLIP( ) / NOFLIP( )
01064 *
01065 *      8   L/R + U/D + F/N
01066 *      LUF 0   LEFT/UP/FLIP
01067 *
01068 * @param q      6      6      (rad)
01069 * @return      ResultWithErrno3 std::tuple<int, int>
01070 *              (RobotConfiguration)
01071 *      -1 (NONE) -
01072 *      0 (LUF)  - LEFT+UP+FLIP
01073 *      1 (LUN)  - LEFT+UP+NOFLIP
01074 *      2 (LDF)  - LEFT+DOWN+FLIP
01075 *      3 (LDN)  - LEFT+DOWN+NOFLIP
01076 *      4 (RUF)  - RIGHT+UP+FLIP
01077 *      5 (RUN)  - RIGHT+UP+NOFLIP
01078 *      6 (RDF)  - RIGHT+DOWN+FLIP
01079 *      7 (RDN)  - RIGHT+DOWN+NOFLIP
01080 *      - int
01081 *      0 -
01082 *      -1 -
01083 *      -5 -      6
01084 * @throws arcs::common_interface::AuboException
01085 *
01086 * @par Python
01087 * getRobotConfiguration(self: pyaubo_sdk.RobotAlgorithm, arg0: List[float]) -> Tuple[int, int]
01088 *
01089 * @par Lua
01090 * getRobotConfiguration(q: table) -> number, number
01091 *
01092 * @par Lua
01093 * -- 6      rad
01094 * local joint_angles = {3.083,1.227,1.098,0.670,-1.870,-0.397}
01095 * local config_val, err_code = getRobotConfiguration(joint_angles)
01096 * if err_code == 0 then
01097 *     print("      ", config_val) -- 4 RUF
01098 * else
01099 *     print("      ", err_code)
01100 * end
01101 *
01102 * @par JSON-RPC
01103 * {"jsonrpc":"2.0","method":"rob1.RobotAlgorithm.getRobotConfiguration","params":[[3.083688522170976,1.2273215976885394,1.09807273963114,
01104 * 1.870715392248607,-0.39708546603119627]],"id":1}
01105 *
01106 * @par JSON-RPC
01107 * {"id":1,"jsonrpc":"2.0","result":[4,0]}
01108 * \endchinese
01109 * \english
01110 * Calculate and return the corresponding robot configuration based on input joint angles
01111 *
01112 * The robot configuration consists of three dimensions of states, with the following definitions for each dimension:
01113 * - Shoulder direction: LEFT (Shoulder to the left) / RIGHT (Shoulder to the right)
01114 * - Elbow direction: UP (Elbow up) / DOWN (Elbow down)
01115 * - Wrist state: FLIP (Wrist flipped) / NOFLIP (Wrist not flipped)
01116 *
01117 * The combination of the three dimensions forms 8 basic configurations (L/R + U/D + F/N). This interface parses the
01118 * current
01119 * configuration type from the input joint angles and returns its integer value corresponding to the RobotConfiguration
01120 * enumeration
01121 * and an error code (Note: The interface returns combined configuration values, e.g., LUF corresponds to 0, not the
01122 * individual
01123 * enumeration values of LEFT/UP/FLIP).

```

```

01121 * @param q Input joint angle array, 6 elements for 6-axis robot, unit: radians (rad)
01122 * @return Robot configuration result and error code (type: ResultWithErrno3, i.e., std::tuple<int, int>):
01123 *     - First int: Integer value corresponding to the RobotConfiguration enumeration, value range and meanings:
01124 *         -1 (NONE) - Invalid configuration
01125 *         0 (LUF) - LEFT+UP+FLIP (Left shoulder, elbow up, wrist flipped)
01126 *         1 (LUN) - LEFT+UP+NOFLIP (Left shoulder, elbow up, wrist not flipped)
01127 *         2 (LDF) - LEFT+DOWN+FLIP (Left shoulder, elbow down, wrist flipped)
01128 *         3 (LDN) - LEFT+DOWN+NOFLIP (Left shoulder, elbow down, wrist not flipped)
01129 *         4 (RUF) - RIGHT+UP+FLIP (Right shoulder, elbow up, wrist flipped)
01130 *         5 (RUN) - RIGHT+UP+NOFLIP (Right shoulder, elbow up, wrist not flipped)
01131 *         6 (RDF) - RIGHT+DOWN+FLIP (Right shoulder, elbow down, wrist flipped)
01132 *         7 (RDN) - RIGHT+DOWN+NOFLIP (Right shoulder, elbow down, wrist not flipped)
01133 *     - Second int: Error code with the following meanings:
01134 *         0 - Success: Configuration calculation completed, valid configuration value returned
01135 *        -1 - Abnormal robot state: Not initialized completely, try reinitializing before calling
01136 *        -5 - Invalid input parameters: Incorrect dimension of joint angle array (not 6 elements) or values out of
reasonable range
01137 *
01138 * @throws arcs::common_interface::AuboException Thrown when input parameters are illegal (e.g., empty array, wrong
number of elements)
01139 *
01140 * @par Python function prototype
01141 * getRobotConfiguration(self: pyaubo_sdk.RobotAlgorithm, arg0: List[float]) -> Tuple[int, int]
01142 *
01143 * @par Lua function prototype
01144 * getRobotConfiguration(q: table) -> number, number
01145 *
01146 * @par Lua example
01147 * -- Input 6-axis joint angles (unit: rad) to get configuration and error code
01148 * local joint_angles = {3.083,1.227,1.098,0.670,-1.870,-0.397}
01149 * local config_val, err_code = getRobotConfiguration(joint_angles)
01150 * if err_code == 0 then
01151 *     print("Robot configuration value: ", config_val) -- Example output: 4 (corresponding to RUF, Right shoulder,
elbow up, wrist flipped)
01152 * else
01153 *     print("Failed to get configuration, error code: ", err_code)
01154 * end
01155 *
01156 * @par JSON-RPC request example
01157 *
{"jsonrpc":"2.0","method":"rob1.RobotAlgorithm.getRobotConfiguration","params":[[3.083688522170976,1.2273215976885394,1.09807273963114
1.870715392248607,-0.39708546603119627]],"id":1}
01158 *
01159 * @par JSON-RPC response example
01160 * {"id":1,"jsonrpc":"2.0","result":[4,0]}
01161 * \endenglish
01162 */
01163 ResultWithErrno3 getRobotConfiguration(const std::vector<double>& q);
01164
01165 /**
01166 * \chinese
01167 * movej
01168 *
01169 * @param q1 movej
01170 * @param r1 q1
01171 * @param q2 movej
01172 * @param r2 q2
01173 * @param d
01174 * @return q1~q2 (x,y,z,rx,ry,rz)
01175 *
01176 * @throws arcs::common_interface::AuboException
01177 *
01178 * @par Python
01179 * pathMovej(self: pyaubo_sdk.RobotAlgorithm, arg0: List[float], arg1:
01180 * float, arg2: List[float], arg3: float, arg4: float) -> List[List[float]]
01181 *
01182 * @par Lua
01183 * pathMovej(q1: table, r1: number, q2: table, r2: number, d: number) ->
01184 * table, number
01185 *
01186 * @par Lua
01187 * path , num =
pathMovej({0.0,-0.2618,1.7453,0.4364,1.5711,0.0},0.25,{0.3234,-0.5405,1.5403,0.5881,1.2962,0.7435},0.03,0.2)
01188 *
01189 * \endchinese
01190 * \english
01191 * Solve the trajectory points between movej
01192 *
01193 * @param q1 Start point of movej
01194 * @param r1 Blend radius at q1
01195 * @param q2 End point of movej
01196 * @param r2 Blend radius at q2
01197 * @param d Sampling distance
01198 * @return Discrete trajectory points (x, y, z, rx, ry, rz) between q1 and q2 in Cartesian space
01199 *
01200 * @throws arcs::common_interface::AuboException
01201 *

```

```

01202 * @par Python function prototype
01203 * pathMovej(self: pyaubo_sdk.RobotAlgorithm, arg0: List[float], arg1:
01204 * float, arg2: List[float], arg3: float, arg4: float) -> List[List[float]]
01205 *
01206 * @par Lua function prototype
01207 * pathMovej(q1: table, r1: number, q2: table, r2: number, d: number) ->
01208 * table, number
01209 *
01210 * @par Lua example
01211 * path , num =
pathMovej({0.0,-0.2618,1.7453,0.4364,1.5711,0.0},0.25,{0.3234,-0.5405,1.5403,0.5881,1.2962,0.7435},0.03,0.2)
01212 *
01213 * \endenglish
01214 */
01215 std::vector<std::vector<double> > pathMovej(const std::vector<double> &q1,
01216 double r1,
01217 const std::vector<double> &q2,
01218 double r2, double d);
01219 /**
01220 * @ingroup RobotAlgorithm
01221 * \chinese
01222 *
01223 *
01224 * @param q
01225 * @param base_or_end
01226 * true: base
01227 * false:
01228 * @return
01229 *
01230 * 0.28.1-rc.21 0.29.0-alpha.25
01231 * 30082
01232 * 0 -
01233 * -23 -
01234 * -24 -
01235 * -25 -
01236 * -26 -
01237 * -27 -
01238 * -28 -
01239 * 0, qnear
01240 *
01241 * @throws arcs::common_interface::AuboException
01242 *
01243 * @par Python
01244 * calJacobian(self: pyaubo_sdk.RobotAlgorithm, arg0: List[float],
01245 * arg1: bool) -> Tuple[List[float], int]
01246 *
01247 * @par Lua
01248 * calJacobian(q: table, base_or_end: boolean) -> table
01249 *
01250 * @par Lua
01251 * calJ_result = calJacobian({0.58815,0.0532,0.62391,2.46,0.479,1.619},true)
01252 *
01253 * @par JSON-RPC
01254 * {"json-
rpc":"2.0","method":"rob1.RobotAlgorithm.calcJacobian","params":[[0.58815,0.0532,0.62391,2.46,0.479,1.619],true],"id":1}
01255 *
01256 * @par JSON-RPC
01257 * {"id":1,"jsonrpc":"2.0","result":[[0.20822779551242535,-
0.5409416184208162,0.2019786999613013,0.061264982268770196,-0.026269884327316487,
01258 * 0.10131708699859962,0.26388933410019777,-0.36074292664199115,0.1346954733416397,0.04085636647597124,-
0.07244204452918337,0.0708466286633346,
01259 * 0.0,0.10401808481666497,-0.12571344758923886,-
0.07741290545882097,0.18818543519232858,0.04628646442706299,0.0,0.5548228314607867,
01260 * -0.5548228314607868,0.5548228314607868,-0.7901273140338193,0.37230961532208007,0.0,-
0.8319685244586092,0.8319685244586091,-0.8319685244586091,
01261 * -0.5269197820578843,-0.8184088260676008,1.0,3.749399456654644e-33,-6.512048180336603e-18,1.0956823467534067e-
16,-0.31313634553301894,
01262 * 0.43771285536682175],0]}
01263 * \endchinese
01264 * \english
01265 * Calculate the Jacobian matrix at the robot end-effector
01266 *
01267 * @param q Joint angles
01268 * @param base_or_end Reference frame: base (or end-effector)
01269 * true: described in base frame
01270 * false: described in end-effector frame
01271 * @return Whether the Jacobian matrix is valid
01272 * The first parameter of the return value is the Jacobian matrix for this configuration, the second is the error code.
01273 * The error code list was updated after versions 0.28.1-rc.21 and 0.29.0-alpha.25.
01274 * Previously, inverse kinematics errors returned 30082. The updated error codes are:
01275 * 0 - Success
01276 * -23 - Inverse kinematics calculation does not converge, calculation error
01277 * -24 - Inverse kinematics calculation exceeds robot limits
01278 * -25 - Inverse kinematics input configuration error
01279 * -26 - Inverse kinematics Jacobian calculation failed
01280 * -27 - Analytical solution exists but none satisfy the selection criteria
01281 * -28 - Unknown error in inverse kinematics

```

```

01282 * If the error code is not 0, the first parameter of the return value is the input reference joint angles qnear
01283 *
01284 * @throws arcs::common_interface::AuboException
01285 *
01286 * @par Python function prototype
01287 * calJacobian(self: pyaubo_sdk.RobotAlgorithm, arg0: List[float],
01288 * arg1: bool) -> Tuple[List[float], int]
01289 *
01290 * @par Lua function prototype
01291 * calJacobian(q: table, base_or_end: boolean) -> table
01292 *
01293 * @par Lua example
01294 * calJ_result = calJacobian({0.58815,0.0532,0.62391,2.46,0.479,1.619},true)
01295 *
01296 * @par JSON-RPC request example
01297 * {"json-
rpc":"2.0","method":"rob1.RobotAlgorithm.calcJacobian","params":[[0.58815,0.0532,0.62391,2.46,0.479,1.619],true],"id":1}
01298 *
01299 * @par JSON-RPC response example
01300 * {"id":1,"jsonrpc":"2.0","result":[[0.20822779551242535,-
0.5409416184208162,0.2019786999613013,0.061264982268770196,-0.026269884327316487,
01301 * 0.10131708699859962,0.26388933410019777,-0.36074292664199115,0.1346954733416397,0.04085636647597124,-
0.07244204452918337,0.0708466286633346,
01302 * 0.0,0.10401808481666497,-0.12571344758923886,-
0.07741290545882097,0.18818543519232858,0.04628646442706299,0.0,0.5548228314607867,
01303 * -0.5548228314607868,0.5548228314607868,-0.7901273140338193,0.37230961532208007,0.0,-
0.8319685244586092,0.8319685244586091,-0.8319685244586091,
01304 * -0.5269197820578843,-0.8184088260676008,1.0,3.749399456654644e-33,-6.512048180336603e-18,1.0956823467534067e-
16,-0.31313634553301894,
01305 * 0.43771285536682175],0]}
01306 * \endenglish
01307 */
01308 ResultWithErrno calcJacobian(const std::vector<double> &q,
01309 bool base_or_end);
01310
01311 /**
01312 * @ingroup RobotAlgorithm
01313 * \chinese
01314 *
01315 *
01316 * @param type
01317 * 0-movej movej
01318 * 1-movej movel
01319 * 2-movej movej
01320 * 3-movel movel
01321 * @param q_start
01322 * @param q_via
01323 * @param q_to
01324 * @param r q_via
01325 * @param d
01326 * @return q_via (x,y,z)
01327 *
01328 * @throws arcs::common_interface::AuboException
01329 *
01330 * @par Python
01331 * pathBlend3Points(self: pyaubo_sdk.RobotAlgorithm, arg0: int, arg1:
01332 * List[float], arg2: List[float], arg3: List[float], arg4: float, arg5:
01333 * float) -> List[List[float]]
01334 *
01335 * @par Lua
01336 * pathBlend3Points(type: number, q_start: table, q_via: table, q_to: table,
01337 * r: number, d: number) -> table, number
01338 *
01339 * @par Lua
01340 * q_via , num = pathBlend3Points(1,{0.58815,0.0532,0.62391,2.46,0.479,1.619},{0.0,-0.2618,1.7453,0.4364,1.5711,0.0},
01341 * {0.3234,-0.5405,1.5403,0.5881,1.2962,0.7435},0.25,0.02)
01342 *
01343 * \endchinese
01344 * \english
01345 * Solve the blended trajectory points
01346 *
01347 * @param type
01348 * 0-movej and movej
01349 * 1-movej and movel
01350 * 2-movel and movej
01351 * 3-movel and movel
01352 * @param q_start Start point before blending
01353 * @param q_via Blending point
01354 * @param q_to End point after blending
01355 * @param r Blend radius at q_via
01356 * @param d Sampling distance
01357 * @return Discrete trajectory points (x, y, z) of the blend segment at q_via in Cartesian space
01358 *
01359 * @throws arcs::common_interface::AuboException
01360 *
01361 * @par Python function prototype
01362 * pathBlend3Points(self: pyaubo_sdk.RobotAlgorithm, arg0: int, arg1:

```

```

01363 * List[float], arg2: List[float], arg3: List[float], arg4: float, arg5:
01364 * float) -> List[List[float]]
01365 *
01366 * @par Lua function prototype
01367 * pathBlend3Points(type: number, q_start: table, q_via: table, q_to: table,
01368 * r: number, d: number) -> table, number
01369 *
01370 * @par Lua example
01371 * q_via, num = pathBlend3Points(1,{0.58815,0.0532,0.62391,2.46,0.479,1.619},{0.0,-0.2618,1.7453,0.4364,1.5711,0.0},
01372 * {0.3234,-0.5405,1.5403,0.5881,1.2962,0.7435},0.25,0.02)
01373 *
01374 * \endenglish
01375 */
01376 std::vector<std::vector<double> > pathBlend3Points(
01377     int type, const std::vector<double> &q_start,
01378     const std::vector<double> &q_via, const std::vector<double> &q_to,
01379     double r, double d);
01380
01381 /**
01382 * @ingroup RobotAlgorithm
01383 * \chinese
01384 *
01385 *     pathBufferAppend
01386 *     buffer     movePathBuffer
01387 * @param name
01388 * @param traj_conf
01389 * traj_conf.move_axis:
01390 *     traj_conf.move_axis=LoadIdentifyMoveAxis::Joint_4_6;
01391 * traj_conf.init_joint:
01392 *     , 5 traj_conf.init_joint[4] 0.3(rad) 1.57(rad)
01393 * traj_conf.lower_joint_bound, traj_conf.upper_joint_bound:
01394 *     , config.move_axis , upper_joint_bound 2 lower_joint_bound -2
01395 * config.max_velocity, config.max_acceleration:
01396 *     , config.move_axis , , max_velocity=3,max_acceleration=5
01397 *
01398 * @return 0
01399 * AUBO_BUSY
01400 * AUBO_BAD_STATE
01401 * -AUBO_INVL_ARGUMENT
01402 * -AUBO_BAD_STATE
01403 *
01404 * @throws arcs::common_interface::AuboException
01405 * \endchinese
01406 * \english
01407 * Generate excitation trajectory for payload identification
01408 * This interface internally calls pathBufferAppend
01409 * The offline trajectory is stored in the buffer, and can be executed later via movePathBuffer
01410 * @param name Trajectory name
01411 * @param traj_conf Joint trajectory constraints
01412 * traj_conf.move_axis: Moving axes
01413 * Since users may not want large multi-joint movements during payload identification, it is recommended to use
01414 * traj_conf.move_axis=LoadIdentifyMoveAxis::Joint_4_6;
01415 * traj_conf.init_joint:
01416 * Initial joint angles. To avoid singularity issues near joint 5 zero position, set abs(traj_conf.init_joint[4]) >= 0.3(rad),
01417 * preferably close to 1.57(rad). Other joints can be set arbitrarily.
01418 * traj_conf.lower_joint_bound, traj_conf.upper_joint_bound:
01419 * Joint angle limits, dimensions should match config.move_axis. Recommended: upper_joint_bound=2,
01420 * lower_joint_bound=-2
01421 * config.max_velocity, config.max_acceleration:
01422 * Joint velocity and acceleration limits, dimensions should match config.move_axis. For safety and driver performance,
01423 * recommended: max_velocity=3, max_acceleration=5
01424 *
01425 * @return Returns 0 on success; error code on failure
01426 * AUBO_BUSY
01427 * AUBO_BAD_STATE
01428 * -AUBO_INVL_ARGUMENT
01429 * -AUBO_BAD_STATE
01430 *
01431 * @throws arcs::common_interface::AuboException
01432 * \endenglish
01433 */
01434 int generatePayloadIdentifyTraj(const std::string &name,
01435     const TrajConfig &traj_conf);
01436
01437 /**
01438 * @ingroup RobotAlgorithm
01439 * \chinese
01440 *
01441 *
01442 * @return 0; 1; <0;
01443 *
01444 * @throws arcs::common_interface::AuboException
01445 *
01446 * @par JSON-RPC
01447 * {"jsonrpc":"2.0","method":"rob1.RobotAlgorithm.payloadIdentifyTrajGenFinished","params":[],"id":1}
01448 *
01449 * @par JSON-RPC
01450 * {"id":1,"jsonrpc":"2.0","result":0}

```

```

01446  *
01447  * \endchinese
01448  * \english
01449  * Whether payload identification trajectory generation is finished
01450  *
01451  * @return 0 if finished; 1 if in progress; <0 if failed;
01452  *
01453  * @throws arcs::common_interface::AuboException
01454  *
01455  * @par JSON-RPC request example
01456  * {"jsonrpc":"2.0","method":"rob1.RobotAlgorithm.payloadIdentifyTrajGenFinished","params":[],"id":1}
01457  *
01458  * @par JSON-RPC response example
01459  * {"id":1,"jsonrpc":"2.0","result":0}
01460  *
01461  * \endenglish
01462  */
01463 int payloadIdentifyTrajGenFinished();
01464
01465 /**
01466  * @ingroup RobotAlgorithm
01467  * \chinese
01468  *   moveS
01469  *
01470  * @brief pathMoveS
01471  * @param qs
01472  * @param d
01473  * @return
01474  *
01475  * @throws arcs::common_interface::AuboException
01476  * \endchinese
01477  * \english
01478  * Solve the trajectory points for moveS
01479  *
01480  * @brief pathMoveS
01481  * @param qs Spline trajectory generation point set
01482  * @param d Sampling distance
01483  * @return
01484  *
01485  * @throws arcs::common_interface::AuboException
01486  * \endenglish
01487  */
01488 std::vector<std::vector<double> > pathMoveS(
01489     const std::vector<std::vector<double> > &qs, double d);
01490
01491 /**
01492  * @ingroup RobotAlgorithm
01493  * \chinese
01494  *
01495  *
01496  * @param q
01497  * @param qd
01498  * @param target_q
01499  * @param target_qd
01500  * @param target_qdd
01501  * @param tool_offset TCP
01502  * @param omega
01503  * @param zeta
01504  * @return
01505  *
01506  * @throws arcs::common_interface::AuboException
01507  *
01508  * @par Python
01509  * calibVibrationParams(self: pyaubo_sdk.RobotAlgorithm, arg0:
01510  * List[List[float]], arg1: List[List[float]], arg2: List[List[float]], arg3: List[List[float]],
01511  * arg4: List[List[float]], arg5: List[List[float]] -> list[list[float]],int
01512  *
01513  * @par Lua
01514  * calibVibrationParams(q: table,qd: table, target_q: table, target_qd: table,
01515  * target_qdd: table, tool_offset: table, omega: table, zeta: table) -> table,number
01516  * \endchinese
01517  */
01518 ResultWithErrno1 calibVibrationParams(const std::vector<std::vector<double> > &q,
01519     const std::vector<std::vector<double> > &qd,
01520     const std::vector<std::vector<double> > &target_q,
01521     const std::vector<std::vector<double> > &target_qd,
01522     const std::vector<std::vector<double> > &target_qdd,
01523     const std::vector<double> &tool_offset);
01524
01525 /**
01526  * @ingroup RobotAlgorithm
01527  * \chinese
01528  *   1
01529  *
01530  * @param record_cache_name
01531  * @param tool_offset TCP
01532  * @return

```



```

01533 *
01534 * @throws arcs::common_interface::AuboException
01535 *
01536 * @par Python
01537 * calibVibrationParams(self: pyaubo_sdk.RobotAlgorithm, arg0:
01538 * string, arg1: List[float]) -> list[list[float]],int
01539 *
01540 * @par Lua
01541 * calibVibrationParams(record_cache_name: string, tool_offset: table) -> table,number
01542 * \endchinese
01543 */
01544 ResultWithErrno1 calibVibrationParams1(const std::string &record_cache_name, const std::vector<double>
&tool_offset);
01545
01546 /**
01547 * @ingroup RobotAlgorithm
01548 * \chinese
01549 *
01550 *
01551 * @param param1
01552 * @param param2
01553 * @param threshold 0~1
01554 * @return >0      =0    <0
01555 *
01556 * @par Lua
01557 * needVibrationRecalib(param1: table, param2: table, threshold: number) -> number
01558 *
01559 * @par JSON-RPC
01560 * {"jsonrpc":"2.0","method":"VibrationController.needVibrationRecalib",
01561 *  "params":[param1_obj, param2_obj, 0.1],"id":1}
01562 * \endchinese
01563 */
01564 int needVibrationRecalib(const VibrationRecalibrationParameter &param1,
01565                          const VibrationRecalibrationParameter &param2,
01566                          double threshold);
01567
01568 /**
01569 * @ingroup RobotAlgorithm
01570 * \chinese
01571 *
01572 *
01573 *
01574 * <->    <->
01575 *
01576 * @param type
01577 * 0 -
01578 * 1 -
01579 * @param start
01580 * @param r1      m
01581 * @param end
01582 * @param r2      m
01583 * @param d      m
01584 * @return
01585 * 0 -
01586 * -18 -      /
01587 * -21 -
01588 * -22 -
01589 * -24 -
01590 * -27 -
01591 *
01592 * @throws arcs::common_interface::AuboException
01593 *
01594 * @par Lua
01595 * validatePath(type, start, r1, end, r2, d) -> number
01596 *
01597 * @par Lua
01598 * -- 6
01599 * local start_joint = {0.0, 0.0, 90.0, 0.0, 90.0, 0.0} --
01600 * local end_joint = {30.0, 0.0, 90.0, 0.0, 90.0, 0.0}
01601 * local result = validatePath(0, start_joint, 0.01, end_joint, 0.01, 0.05)
01602 *
01603 * @par Python
01604 * validatePath(type: int, start: list[float], r1: float, end: list[float],
01605 * r2: float, d: float) -> int
01606 *
01607 * @par Python
01608 * # 6
01609 * start_joint = [0.0, 0.0, math.pi/2, 0.0, math.pi/2, 0.0]
01610 * end_pose = [0.5, 0.2, 0.8, 0.0, math.pi/2, 0.0]
01611 * result = validatePath(1, start_joint, 0.01, end_pose, 0.01, 0.05)
01612 *
01613 * @par JSON-RPC
01614 * {"jsonrpc":"2.0","method":"rob1.RobotAlgorithm.validatePath","params":[0,
01615 * [0.0,0.0,1.57,0.0,1.57,0.0], 0.01, [0.52,0.0,1.57,0.0,1.57,0.0], 0.01,
01616 * 0.05],"id":1}
01617 *
01618 * @par JSON-RPC

```



```

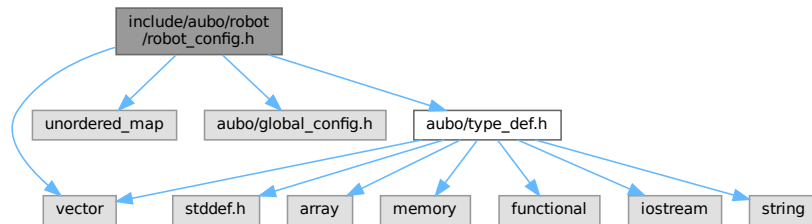
01619 * {"id":1,"jsonrpc":"2.0","result":0}
01620 * \endchinese
01621 *
01622 * \english
01623 * Validate the reachability of the robot's motion path from start point to
01624 * end point
01625 *
01626 * This interface verifies whether the specified path has unreachable
01627 * conditions such as out-of-limit, self-collision, singularity, etc., by
01628 * sampling. It supports four path type validations: Joint-Joint,
01629 * Joint-Pose, Pose-Joint, Pose-Pose.
01630 *
01631 * @param type Path type identifier, specifying the data type of start and
01632 * end points:
01633 * 0 - Start: Joint angles, End: Joint angles
01634 * 1 - Start: Pose, End: Pose
01635 * @param start Start point data of the path
01636 * - Joint angles
01637 * - Pose
01638 * @param r1 Start point blending radius, unit: m (meters), used to smooth
01639 * path sampling near the start point
01640 * @param end End point data of the path, format is consistent with start
01641 * (matching the type specified by type)
01642 * @param r2 End point blending radius, unit: m (meters), used to smooth
01643 * path sampling near the end point
01644 * @param d Path sampling interval, unit: m (meters). Smaller intervals
01645 * improve verification accuracy but increase time consumption
01646 *
01647 * @return Path reachability result code:
01648 * 0 - Reachable: Path is normal, movement is possible
01649 * -18 - Joint/ Cartesian space limits exceeded in the path
01650 * -21 - Trajectory generation failed
01651 * -22 - Self-collision of the robot body in the path
01652 * -24 - Path passing through robot singular configuration
01653 * -27 - The target point has a solution but exceeds joint limits
01654 * configuration
01655 *
01656 * @throws arcs::common_interface::AuboException Thrown when parameters are
01657 * invalid (e.g., mismatched data length, values out of reasonable range,
01658 * etc.)
01659 *
01660 * @par Lua function prototype
01661 * validatePath(type, start, r1, end, r2, d) -> integer
01662 *
01663 * @par Lua example
01664 * -- Validate reachability of Joint-Joint path for 6-axis robot
01665 * local start_joint = {0.0, 0.0, 90.0, 0.0, 90.0, 0.0}
01666 * local end_joint = {30.0, 0.0, 90.0, 0.0, 90.0, 0.0}
01667 * result = validatePath(0, start_joint, 0.01, end_joint, 0.01, 0.05)
01668 *
01669 * @par Python function prototype
01670 * validatePath(type: int, start: list[float], r1: float, end: list[float],
01671 * r2: float, d: float) -> int
01672 *
01673 * @par Python example
01674 * # Validate reachability of Joint-Pose path for 6-axis robot
01675 * start_joint = [0.0, 0.0, math.pi/2, 0.0, math.pi/2, 0.0]
01676 * end_pose = [0.5, 0.2, 0.8, 0.0, math.pi/2, 0.0]
01677 * result = validatePath(1, start_joint, 0.01, end_pose, 0.01, 0.05)
01678 *
01679 * @par JSON-RPC request example
01680 * {"jsonrpc":"2.0","method":"rob1.RobotAlgorithm.validatePath","params":[0,
01681 * [0.0,0.0,1.57,0.0,1.57,0.0], 0.01, [0.52,0.0,1.57,0.0,1.57,0.0], 0.01,
01682 * 0.005],"id":1}
01683 *
01684 * @par JSON-RPC response example {"id":1,"jsonrpc":"2.0","result":0}
01685 * \endenglish
01686 */
01687 int validatePath(int type, const std::vector<double> &start, double r1,
01688 const std::vector<double> &end, double r2, double d);
01689
01690 protected:
01691 void *d_;
01692 };
01693 using RobotAlgorithmPtr = std::shared_ptr<RobotAlgorithm>;
01694
01695 } // namespace common_interface
01696 } // namespace arcs
01697
01698 #endif // AUBO_SDK_ROBOT_ALGORITHM_INTERFACE_H

```

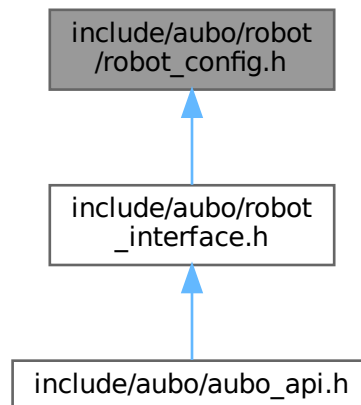
12.28 include/aubo/robot/robot_config.h File Reference

DH

```
#include <vector>
#include <unordered_map>
#include <aubo/global_config.h>
#include <aubo/type_def.h>
Include dependency graph for robot_config.h:
```



This graph shows which files directly or indirectly include this file:



Classes

- class [arcs::common_interface::RobotConfig](#)

Namespaces

- namespace [arcs](#)
- namespace [arcs::common_interface](#)

Typedefs

- using [arcs::common_interface::RobotConfigPtr](#) = std::shared_ptr<[RobotConfig](#)>

12.28.1 Detailed Description

DH

Definition in file [robot_config.h](#).

12.29 robot_config.h

[Go to the documentation of this file.](#)

```

00001 /** @file robot_config.h
00002 * @brief DH
00003 */
00004 #ifndef AUBO_SDK_ROBOT_CONFIG_H
00005 #define AUBO_SDK_ROBOT_CONFIG_H
00006
00007 #include <vector>
00008 #include <unordered_map>
00009
00010 #include <aubo/global_config.h>
00011 #include <aubo/type_def.h>
00012
00013 namespace arcs {
00014 namespace common_interface {
00015
00016 /**
00017 * @defgroup RobotConfig RobotConfig ( )
00018 * @ingroup RobotInterface
00019 * RobotConfig
00020 */
00021 class ARCS_ABI_EXPORT RobotConfig
00022 {
00023 public:
00024     RobotConfig();
00025     virtual ~RobotConfig();
00026
00027     /**
00028      * @ingroup RobotConfig
00029      * \chinese
00030      *
00031      *
00032      * @return
00033      *
00034      * @throws arcs::common_interface::AuboException
00035      *
00036      * @par Python
00037      * getName(self: pyaubo_sdk.RobotConfig) -> str
00038      *
00039      * @par Lua
00040      * getName() -> string
00041      *
00042      * @par Lua
00043      * robot_name = getName()
00044      *
00045      * @par JSON-RPC
00046      * {"jsonrpc":"2.0","method":"rob1.RobotConfig.getName","params":[],"id":1}
00047      *
00048      * @par JSON-RPC
00049      * {"id":1,"jsonrpc":"2.0","result":"rob1"}
00050      * \endchinese
00051      * \english
00052      * Get the robot's name.
00053      *
00054      * @return The robot's name.
00055      *
00056      * @throws arcs::common_interface::AuboException
00057      *
00058      * @par Python function prototype
00059      * getName(self: pyaubo_sdk.RobotConfig) -> str
00060      *
00061      * @par Lua function prototype

```

```

00062 * getName() -> string
00063 *
00064 * @par Lua example
00065 * robot_name = getName()
00066 *
00067 * @par JSON-RPC request example
00068 * {"jsonrpc":"2.0","method":"rob1.RobotConfig.getName","params":[],"id":1}
00069 *
00070 * @par JSON-RPC response example
00071 * {"id":1,"jsonrpc":"2.0","result":"rob1"}
00072 * \endenglish
00073 */
00074 std::string getName();
00075
00076 /**
00077 * @ingroup RobotConfig
00078 * \chinese
00079 * ( )
00080 *
00081 * @return
00082 *
00083 * @throws arcs::common_interface::AuboException
00084 *
00085 * @par Python
00086 * getDof(self: pyaubo_sdk.RobotConfig) -> int
00087 *
00088 * @par Lua
00089 * getDof() -> number
00090 *
00091 * @par Lua
00092 * robot_dof = getDof()
00093 *
00094 * @par JSON-RPC
00095 * {"jsonrpc":"2.0","method":"rob1.RobotConfig.getDof","params":[],"id":1}
00096 *
00097 * @par JSON-RPC
00098 * {"id":1,"jsonrpc":"2.0","result":6}
00099 * \endchinese
00100 * \english
00101 * Get the robot's degrees of freedom (from the hardware abstraction layer).
00102 *
00103 * @return The robot's degrees of freedom.
00104 *
00105 * @throws arcs::common_interface::AuboException
00106 *
00107 * @par Python function prototype
00108 * getDof(self: pyaubo_sdk.RobotConfig) -> int
00109 *
00110 * @par Lua function prototype
00111 * getDof() -> number
00112 *
00113 * @par Lua example
00114 * robot_dof = getDof()
00115 *
00116 * @par JSON-RPC request example
00117 * {"jsonrpc":"2.0","method":"rob1.RobotConfig.getDof","params":[],"id":1}
00118 *
00119 * @par JSON-RPC response example
00120 * {"id":1,"jsonrpc":"2.0","result":6}
00121 * \endenglish
00122 */
00123 int getDof();
00124
00125 /**
00126 * @ingroup RobotConfig
00127 * \chinese
00128 * ( )
00129 *
00130 * @return
00131 *
00132 * @throws arcs::common_interface::AuboException
00133 *
00134 * @par Python
00135 * getCycletime(self: pyaubo_sdk.RobotConfig) -> float
00136 *
00137 * @par Lua
00138 * getCycletime() -> number
00139 *
00140 * @par Lua
00141 * robot_Cycletime = getCycletime()
00142 *
00143 * @par JSON-RPC
00144 * {"jsonrpc":"2.0","method":"rob1.RobotConfig.getCycletime","params":[],"id":1}
00145 *
00146 * @par JSON-RPC
00147 * {"id":1,"jsonrpc":"2.0","result":0.005}
00148 * \endchinese

```

```

00149 * \english
00150 * Get the robot's servo control cycle time (from the hardware abstraction
00151 * layer).
00152 *
00153 * @return The robot's servo control cycle time.
00154 *
00155 * @throws arcs::common_interface::AuboException
00156 *
00157 * @par Python function prototype
00158 * getCycletime(self: pyaubo_sdk.RobotConfig) -> float
00159 *
00160 * @par Lua function prototype
00161 * getCycletime() -> number
00162 *
00163 * @par Lua example
00164 * robot_Cycletime = getCycletime()
00165 *
00166 * @par JSON-RPC request example
00167 * {"jsonrpc":"2.0","method":"rob1.RobotConfig.getCycletime","params":[],"id":1}
00168 *
00169 * @par JSON-RPC response example
00170 * {"id":1,"jsonrpc":"2.0","result":0.005}
00171 * \endenglish
00172 */
00173 double getCycletime();
00174
00175 /**
00176 * @ingroup RobotConfig
00177 * \chinese
00178 *
00179 *
00180 * @param level 1, 2
00181 * @param fraction
00182 *
00183 * @return 0
00184 * AUBO_BUSY
00185 * AUBO_BAD_STATE
00186 * AUBO_INVL_ARGUMENT
00187 * -AUBO_BAD_STATE
00188 *
00189 * @throws arcs::common_interface::AuboException
00190 *
00191 * @par JSON-RPC
00192 * {"jsonrpc":"2.0","method":"rob1.RobotConfig.setSlowDownFraction","params":[1,0.8],"id":1}
00193 *
00194 * @par JSON-RPC
00195 * {"id":1,"jsonrpc":"2.0","result":0}
00196 * \endchinese
00197 * \english
00198 * Set the speed reduction ratio for the preset slow-down mode.
00199 *
00200 * @param level Slow-down level 1, 2
00201 * @param fraction
00202 *
00203 * @return Returns 0 on success; error code on failure
00204 * AUBO_BUSY
00205 * AUBO_BAD_STATE
00206 * AUBO_INVL_ARGUMENT
00207 * -AUBO_BAD_STATE
00208 *
00209 * @throws arcs::common_interface::AuboException
00210 *
00211 * @par JSON-RPC request example
00212 * {"jsonrpc":"2.0","method":"rob1.RobotConfig.setSlowDownFraction","params":[1,0.8],"id":1}
00213 *
00214 * @par JSON-RPC response example
00215 * {"id":1,"jsonrpc":"2.0","result":0}
00216 * \endenglish
00217 */
00218 int setSlowDownFraction(int level, double fraction);
00219
00220 /**
00221 * @ingroup RobotConfig
00222 * \chinese
00223 *
00224 *
00225 * @param level 1, 2
00226 * @return
00227 *
00228 * @throws arcs::common_interface::AuboException
00229 *
00230 * @par JSON-RPC
00231 * {"jsonrpc":"2.0","method":"rob1.RobotConfig.getSlowDownFraction","params":[1],"id":1}
00232 *
00233 * @par JSON-RPC
00234 * {"id":1,"jsonrpc":"2.0","result":0.5}
00235 * \endchinese

```

```

00236 * \english
00237 * Get the speed reduction ratio for the preset slow-down mode.
00238 *
00239 * @param level Slow-down level 1, 2
00240 * @return The speed reduction ratio for the preset slow-down mode.
00241 *
00242 * @throws arcs::common_interface::AuboException
00243 *
00244 * @par JSON-RPC request example
00245 * {"jsonrpc":"2.0","method":"rob1.RobotConfig.getSlowDownFraction","params":[1],"id":1}
00246 *
00247 * @par JSON-RPC response example
00248 * {"id":1,"jsonrpc":"2.0","result":0.5}
00249 * \endenglish
00250 */
00251 double getSlowDownFraction(int level);
00252
00253 /**
00254 * @ingroup RobotConfig
00255 * \chinese
00256 *      m/s^2
00257 *
00258 * @return
00259 *
00260 * @throws arcs::common_interface::AuboException
00261 *
00262 * @par Python
00263 * getDefaultToolAcc(self: pyaubo_sdk.RobotConfig) -> float
00264 *
00265 * @par Lua
00266 * getDefaultToolAcc() -> number
00267 *
00268 * @par Lua
00269 * DefaultToolAcc = getDefaultToolAcc()
00270 *
00271 * @par JSON-RPC
00272 * {"jsonrpc":"2.0","method":"rob1.RobotConfig.getDefaultToolAcc","params":[],"id":1}
00273 *
00274 * @par JSON-RPC
00275 * {"id":1,"jsonrpc":"2.0","result":0.0}
00276 * \endchinese
00277 * \english
00278 * Get the default tool acceleration, in m/s^2.
00279 *
00280 * @return The default tool acceleration.
00281 *
00282 * @throws arcs::common_interface::AuboException
00283 *
00284 * @par Python function prototype
00285 * getDefaultToolAcc(self: pyaubo_sdk.RobotConfig) -> float
00286 *
00287 * @par Lua function prototype
00288 * getDefaultToolAcc() -> number
00289 *
00290 * @par Lua example
00291 * DefaultToolAcc = getDefaultToolAcc()
00292 *
00293 * @par JSON-RPC request example
00294 * {"jsonrpc":"2.0","method":"rob1.RobotConfig.getDefaultToolAcc","params":[],"id":1}
00295 *
00296 * @par JSON-RPC response example
00297 * {"id":1,"jsonrpc":"2.0","result":0.0}
00298 * \endenglish
00299 */
00300 double getDefaultToolAcc();
00301
00302 /**
00303 * @ingroup RobotConfig
00304 * \chinese
00305 *      m/s
00306 *
00307 * @return
00308 *
00309 * @throws arcs::common_interface::AuboException
00310 *
00311 * @par Python
00312 * getDefaultToolSpeed(self: pyaubo_sdk.RobotConfig) -> float
00313 *
00314 * @par Lua
00315 * getDefaultToolSpeed() -> number
00316 *
00317 * @par Lua
00318 * DefaultToolSpeed = getDefaultToolSpeed()
00319 *
00320 * @par JSON-RPC
00321 * {"jsonrpc":"2.0","method":"rob1.RobotConfig.getDefaultToolSpeed","params":[],"id":1}
00322 *

```

```

00323 * @par JSON-RPC
00324 * {"id":1,"jsonrpc":"2.0","result":0.0}
00325 * \endchinese
00326 * \english
00327 * Get the default tool speed, in m/s.
00328 *
00329 * @return The default tool speed.
00330 *
00331 * @throws arcs::common_interface::AuboException
00332 *
00333 * @par Python function prototype
00334 * getDefaultToolSpeed(self: pyaubo_sdk.RobotConfig) -> float
00335 *
00336 * @par Lua function prototype
00337 * getDefaultToolSpeed() -> number
00338 *
00339 * @par Lua example
00340 * DefaultToolSpeed = getDefaultToolSpeed()
00341 *
00342 * @par JSON-RPC request example
00343 * {"jsonrpc":"2.0","method":"rob1.RobotConfig.getDefaultToolSpeed","params":[],"id":1}
00344 *
00345 * @par JSON-RPC response example
00346 * {"id":1,"jsonrpc":"2.0","result":0.0}
00347 * \endenglish
00348 */
00349 double getDefaultToolSpeed();
00350
00351 /**
00352 * @ingroup RobotConfig
00353 * \chinese
00354 *      rad/s^2
00355 *
00356 * @return
00357 *
00358 * @throws arcs::common_interface::AuboException
00359 *
00360 * @par Python
00361 * getDefaultJointAcc(self: pyaubo_sdk.RobotConfig) -> float
00362 *
00363 * @par Lua
00364 * getDefaultJointAcc() -> number
00365 *
00366 * @par Lua
00367 * DefaultJointAcc = getDefaultJointAcc()
00368 *
00369 * @par JSON-RPC
00370 * {"jsonrpc":"2.0","method":"rob1.RobotConfig.getDefaultJointAcc","params":[],"id":1}
00371 *
00372 * @par JSON-RPC
00373 * {"id":1,"jsonrpc":"2.0","result":0.0}
00374 * \endchinese
00375 * \english
00376 * Get the default joint acceleration, in rad/s^2.
00377 *
00378 * @return The default joint acceleration.
00379 *
00380 * @throws arcs::common_interface::AuboException
00381 *
00382 * @par Python function prototype
00383 * getDefaultJointAcc(self: pyaubo_sdk.RobotConfig) -> float
00384 *
00385 * @par Lua function prototype
00386 * getDefaultJointAcc() -> number
00387 *
00388 * @par Lua example
00389 * DefaultJointAcc = getDefaultJointAcc()
00390 *
00391 * @par JSON-RPC request example
00392 * {"jsonrpc":"2.0","method":"rob1.RobotConfig.getDefaultJointAcc","params":[],"id":1}
00393 *
00394 * @par JSON-RPC response example
00395 * {"id":1,"jsonrpc":"2.0","result":0.0}
00396 * \endenglish
00397 */
00398 double getDefaultJointAcc();
00399
00400 /**
00401 * @ingroup RobotConfig
00402 * \chinese
00403 *      rad/s
00404 *
00405 * @return
00406 *
00407 * @throws arcs::common_interface::AuboException
00408 *
00409 * @par Python

```

```

00410 * getDefaultJointSpeed(self: pyaubo_sdk.RobotConfig) -> float
00411 *
00412 * @par Lua
00413 * getDefaultJointSpeed() -> number
00414 *
00415 * @par Lua
00416 * DefaultJointSpeed = getDefaultJointSpeed()
00417 *
00418 * @par JSON-RPC
00419 * {"jsonrpc":"2.0","method":"rob1.RobotConfig.getDefaultJointSpeed","params":[],"id":1}
00420 *
00421 * @par JSON-RPC
00422 * {"id":1,"jsonrpc":"2.0","result":0.0}
00423 * \endchinese
00424 * \english
00425 * Get the default joint speed, in rad/s.
00426 *
00427 * @return The default joint speed.
00428 *
00429 * @throws arcs::common_interface::AuboException
00430 *
00431 * @par Python function prototype
00432 * getDefaultJointSpeed(self: pyaubo_sdk.RobotConfig) -> float
00433 *
00434 * @par Lua function prototype
00435 * getDefaultJointSpeed() -> number
00436 *
00437 * @par Lua example
00438 * DefaultJointSpeed = getDefaultJointSpeed()
00439 *
00440 * @par JSON-RPC request example
00441 * {"jsonrpc":"2.0","method":"rob1.RobotConfig.getDefaultJointSpeed","params":[],"id":1}
00442 *
00443 * @par JSON-RPC response example
00444 * {"id":1,"jsonrpc":"2.0","result":0.0}
00445 * \endenglish
00446 */
00447 double getDefaultJointSpeed();
00448
00449 /**
00450 * @ingroup RobotConfig
00451 * \chinese
00452 *
00453 *
00454 * @return
00455 *
00456 * @throws arcs::common_interface::AuboException
00457 *
00458 * @par Python
00459 * getRobotType(self: pyaubo_sdk.RobotConfig) -> str
00460 *
00461 * @par Lua
00462 * getRobotType() -> string
00463 *
00464 * @par Lua
00465 * RobotType = getRobotType()
00466 *
00467 * @par JSON-RPC
00468 * {"jsonrpc":"2.0","method":"rob1.RobotConfig.getRobotType","params":[],"id":1}
00469 *
00470 * @par JSON-RPC
00471 * {"id":1,"jsonrpc":"2.0","result":"aubo_i5H"}
00472 * \endchinese
00473 * \english
00474 * Get the robot type code.
00475 *
00476 * @return The robot type code.
00477 *
00478 * @throws arcs::common_interface::AuboException
00479 *
00480 * @par Python function prototype
00481 * getRobotType(self: pyaubo_sdk.RobotConfig) -> str
00482 *
00483 * @par Lua function prototype
00484 * getRobotType() -> string
00485 *
00486 * @par Lua example
00487 * RobotType = getRobotType()
00488 *
00489 * @par JSON-RPC request example
00490 * {"jsonrpc":"2.0","method":"rob1.RobotConfig.getRobotType","params":[],"id":1}
00491 *
00492 * @par JSON-RPC response example
00493 * {"id":1,"jsonrpc":"2.0","result":"aubo_i5H"}
00494 * \endenglish
00495 */
00496 std::string getRobotType();

```



```

00497
00498 /**
00499  * @ingroup RobotConfig
00500  * \chinese
00501  *
00502  *
00503  * @return
00504  *
00505  * @throws arcs::common_interface::AuboException
00506  *
00507  * @par Python
00508  * getRobotSubType(self: pyaubo_sdk.RobotConfig) -> str
00509  *
00510  * @par Lua
00511  * getRobotSubType() -> string
00512  *
00513  * @par Lua
00514  * RobotSubType = getRobotSubType()
00515  *
00516  * @par JSON-RPC
00517  * {"jsonrpc":"2.0","method":"rob1.RobotConfig.getRobotSubType","params":[],"id":1}
00518  *
00519  * @par JSON-RPC
00520  * {"id":1,"jsonrpc":"2.0","result":"B0"}
00521  * \endchinese
00522  * \english
00523  * Get the robot sub-type code.
00524  *
00525  * @return The robot sub-type code.
00526  *
00527  * @throws arcs::common_interface::AuboException
00528  *
00529  * @par Python function prototype
00530  * getRobotSubType(self: pyaubo_sdk.RobotConfig) -> str
00531  *
00532  * @par Lua function prototype
00533  * getRobotSubType() -> string
00534  *
00535  * @par Lua example
00536  * RobotSubType = getRobotSubType()
00537  *
00538  * @par JSON-RPC request example
00539  * {"jsonrpc":"2.0","method":"rob1.RobotConfig.getRobotSubType","params":[],"id":1}
00540  *
00541  * @par JSON-RPC response example
00542  * {"id":1,"jsonrpc":"2.0","result":"B0"}
00543  * \endenglish
00544  */
00545 std::string getRobotSubType();
00546
00547 /**
00548  * @ingroup RobotConfig
00549  * \chinese
00550  *
00551  *
00552  * @return
00553  *
00554  * @throws arcs::common_interface::AuboException
00555  *
00556  * @par Python
00557  * getControlBoxType(self: pyaubo_sdk.RobotConfig) -> str
00558  *
00559  * @par Lua
00560  * getControlBoxType() -> string
00561  *
00562  * @par Lua
00563  * ControlBoxType = getControlBoxType()
00564  *
00565  * @par JSON-RPC
00566  * {"jsonrpc":"2.0","method":"rob1.RobotConfig.getControlBoxType","params":[],"id":1}
00567  *
00568  * @par JSON-RPC
00569  * {"id":1,"jsonrpc":"2.0","result":"cb_ISStation"}
00570  * \endchinese
00571  * \english
00572  * Get the control box type code.
00573  *
00574  * @return The control box type code.
00575  *
00576  * @throws arcs::common_interface::AuboException
00577  *
00578  * @par Python function prototype
00579  * getControlBoxType(self: pyaubo_sdk.RobotConfig) -> str
00580  *
00581  * @par Lua function prototype
00582  * getControlBoxType() -> string
00583  *

```

```

00584 * @par Lua example
00585 * ControlBoxType = getControlBoxType()
00586 *
00587 * @par JSON-RPC request example
00588 * {"jsonrpc":"2.0","method":"rob1.RobotConfig.getControlBoxType","params":[],"id":1}
00589 *
00590 * @par JSON-RPC response example
00591 * {"id":1,"jsonrpc":"2.0","result":"cb_ISSStation"}
00592 * \endenglish
00593 */
00594 std::string getControlBoxType();
00595
00596 /**
00597 * @ingroup RobotConfig
00598 * \chinese
00599 * ( ) world->base
00600 *
00601 * [0,0,0,0,0,0]
00602 *
00603 * @param pose
00604 * @return 0;
00605 * AUBO_BUSY
00606 * AUBO_BAD_STATE
00607 * -AUBO_INVL_ARGUMENT
00608 * -AUBO_BAD_STATE
00609 *
00610 * @throws arcs::common_interface::AuboException
00611 *
00612 * @par Python
00613 * setMountingPose(self: pyaubo_sdk.RobotConfig, arg0: List[float]) -> int
00614 *
00615 * @par Lua
00616 * setMountingPose(pose: table) -> nil
00617 *
00618 * @par Lua
00619 * setMountingPose({0.0,0.0,0.0,0.0,0.0,0.0})
00620 *
00621 * @par JSON-RPC
00622 * {"jsonrpc":"2.0","method":"rob1.RobotConfig.setMountingPose","params":[[0.0,0.0,0.0,0.0,0.0,0.0]],"id":1}
00623 *
00624 * @par JSON-RPC
00625 * {"id":1,"jsonrpc":"2.0","result":0}
00626 * \endchinese
00627 * \english
00628 * Set the mounting pose (robot base coordinate system relative to world
00629 * coordinate system) world->base
00630 *
00631 * Typically used in multi-robot systems, default is [0,0,0,0,0,0]
00632 *
00633 * @param pose Mounting pose
00634 * @return Returns 0 on success; error code on failure
00635 * AUBO_BUSY
00636 * AUBO_BAD_STATE
00637 * -AUBO_INVL_ARGUMENT
00638 * -AUBO_BAD_STATE
00639 *
00640 * @throws arcs::common_interface::AuboException
00641 *
00642 * @par Python function prototype
00643 * setMountingPose(self: pyaubo_sdk.RobotConfig, arg0: List[float]) -> int
00644 *
00645 * @par Lua function prototype
00646 * setMountingPose(pose: table) -> nil
00647 *
00648 * @par Lua example
00649 * setMountingPose({0.0,0.0,0.0,0.0,0.0,0.0})
00650 *
00651 * @par JSON-RPC request example
00652 * {"jsonrpc":"2.0","method":"rob1.RobotConfig.setMountingPose","params":[[0.0,0.0,0.0,0.0,0.0,0.0]],"id":1}
00653 *
00654 * @par JSON-RPC response example
00655 * {"id":1,"jsonrpc":"2.0","result":0}
00656 * \endenglish
00657 */
00658 int setMountingPose(const std::vector<double> &pose);
00659
00660 /**
00661 * @ingroup RobotConfig
00662 * \chinese
00663 * ( )
00664 *
00665 * @return
00666 *
00667 * @throws arcs::common_interface::AuboException
00668 *
00669 * @par Python
00670 * getMountingPose(self: pyaubo_sdk.RobotConfig) -> List[float]

```

```

00671 *
00672 * @par Lua
00673 * getMountingPose() -> table
00674 *
00675 * @par Lua
00676 * MountingPose = getMountingPose()
00677 *
00678 * @par JSON-RPC
00679 * {"jsonrpc":"2.0","method":"rob1.RobotConfig.getMountingPose","params":[],"id":1}
00680 *
00681 * @par JSON-RPC
00682 * {"id":1,"jsonrpc":"2.0","result":[0.0,0.0,0.0,0.0,0.0,0.0]}
00683 * \endchinese
00684 * \english
00685 * Get the mounting pose (robot base coordinate system relative to world
00686 * coordinate system)
00687 *
00688 * @return Mounting pose
00689 *
00690 * @throws arcs::common_interface::AuboException
00691 *
00692 * @par Python function prototype
00693 * getMountingPose(self: pyaubo_sdk.RobotConfig) -> List[float]
00694 *
00695 * @par Lua function prototype
00696 * getMountingPose() -> table
00697 *
00698 * @par Lua example
00699 * MountingPose = getMountingPose()
00700 *
00701 * @par JSON-RPC request example
00702 * {"jsonrpc":"2.0","method":"rob1.RobotConfig.getMountingPose","params":[],"id":1}
00703 *
00704 * @par JSON-RPC response example
00705 * {"id":1,"jsonrpc":"2.0","result":[0.0,0.0,0.0,0.0,0.0,0.0]}
00706 * \endenglish
00707 */
00708 std::vector<double> getMountingPose();
00709
00710 /**
00711 * @ingroup RobotConfig
00712 * \chinese
00713 *
00714 *
00715 * frame ROBOTBASE
00716 * \endchinese
00717 * \english
00718 * Attach the robot base to a coordinate frame. If this frame moves, the
00719 * robot will move accordingly. Used for applications like ground rails or
00720 * gantries. When this function is called, the relationship between the
00721 * frame and ROBOTBASE is fixed. \endenglish
00722 */
00723 int attachRobotBaseTo(const std::string &frame);
00724 std::string getRobotBaseParent();
00725
00726 /**
00727 * @ingroup RobotConfig
00728 * \chinese
00729 *
00730 * \endchinese
00731 * \english
00732 * Set work object data. All programmed points are based on the work object
00733 * coordinate system. \endenglish
00734 */
00735
00736 int setWorkObjectData(const WObjectData &wobj);
00737
00738 /**
00739 * @ingroup RobotConfig
00740 * \chinese
00741 *
00742 *
00743 *
00744 * @param level
00745 * 0:
00746 * 1-9:
00747 * @return 0;
00748 * AUBO_BUSY
00749 * AUBO_BAD_STATE
00750 * -AUBO_INVL_ARGUMENT
00751 * -AUBO_BAD_STATE
00752 *
00753 * @throws arcs::common_interface::AuboException
00754 *
00755 * @par Python
00756 * setCollisionLevel(self: pyaubo_sdk.RobotConfig, arg0: int) -> int
00757 *

```

```

00758 * @par Lua
00759 * setCollisionLevel(level: number) -> nil
00760 *
00761 * @par Lua
00762 * setCollisionLevel(6)
00763 *
00764 * @par JSON-RPC
00765 * {"jsonrpc": "2.0", "method": "rob1.RobotConfig.setCollisionLevel", "params": [6], "id": 1}
00766 *
00767 * @par JSON-RPC
00768 * {"id": 1, "jsonrpc": "2.0", "result": 0}
00769 * \endchinese
00770 * \english
00771 * Set the collision sensitivity level.
00772 * The higher the value, the more sensitive.
00773 *
00774 * @param level Collision sensitivity level
00775 * 0: Disable collision detection
00776 * 1-9: Collision sensitivity levels
00777 * @return Returns 0 on success; error code on failure
00778 * AUBO_BUSY
00779 * AUBO_BAD_STATE
00780 * -AUBO_INVL_ARGUMENT
00781 * -AUBO_BAD_STATE
00782 *
00783 * @throws arcs::common_interface::AuboException
00784 *
00785 * @par Python function prototype
00786 * setCollisionLevel(self: pyaubo_sdk.RobotConfig, arg0: int) -> int
00787 *
00788 * @par Lua function prototype
00789 * setCollisionLevel(level: number) -> nil
00790 *
00791 * @par Lua example
00792 * setCollisionLevel(6)
00793 *
00794 * @par JSON-RPC request example
00795 * {"jsonrpc": "2.0", "method": "rob1.RobotConfig.setCollisionLevel", "params": [6], "id": 1}
00796 *
00797 * @par JSON-RPC response example
00798 * {"id": 1, "jsonrpc": "2.0", "result": 0}
00799 * \endenglish
00800 */
00801 int setCollisionLevel(int level);
00802
00803 /**
00804 * @ingroup RobotConfig
00805 * \chinese
00806 *
00807 *
00808 * @return
00809 *
00810 * @throws arcs::common_interface::AuboException
00811 *
00812 * @par Python
00813 * getCollisionLevel(self: pyaubo_sdk.RobotConfig) -> int
00814 *
00815 * @par Lua
00816 * getCollisionLevel() -> number
00817 *
00818 * @par Lua
00819 * CollisionLevel = getCollisionLevel()
00820 *
00821 * @par JSON-RPC
00822 * {"jsonrpc": "2.0", "method": "rob1.RobotConfig.getCollisionLevel", "params": [], "id": 1}
00823 *
00824 * @par JSON-RPC
00825 * {"id": 1, "jsonrpc": "2.0", "result": 6}
00826 * \endchinese
00827 * \english
00828 * Get the collision sensitivity level.
00829 *
00830 * @return Collision sensitivity level.
00831 *
00832 * @throws arcs::common_interface::AuboException
00833 *
00834 * @par Python function prototype
00835 * getCollisionLevel(self: pyaubo_sdk.RobotConfig) -> int
00836 *
00837 * @par Lua function prototype
00838 * getCollisionLevel() -> number
00839 *
00840 * @par Lua example
00841 * CollisionLevel = getCollisionLevel()
00842 *
00843 * @par JSON-RPC request example
00844 * {"jsonrpc": "2.0", "method": "rob1.RobotConfig.getCollisionLevel", "params": [], "id": 1}

```

```

00845  *
00846  * @par JSON-RPC response example
00847  * {"id":1,"jsonrpc":"2.0","result":6}
00848  * \endenglish
00849  */
00850 int getCollisionLevel();
00851
00852 /**
00853  * @ingroup RobotConfig
00854  * \chinese
00855  *
00856  *
00857  * @param type \n
00858  * 0: \n
00859  * 1: \n
00860  * 2:
00861  *
00862  * @return 0;
00863  * AUBO_BUSY
00864  * AUBO_BAD_STATE
00865  * -AUBO_INVL_ARGUMENT
00866  * -AUBO_BAD_STATE
00867  *
00868  * @throws arcs::common_interface::AuboException
00869  *
00870  * @par Python
00871  * setCollisionStopType(self: pyaubo_sdk.RobotConfig, arg0: int) -> int
00872  *
00873  * @par Lua
00874  * setCollisionStopType(type: number) -> nil
00875  *
00876  * @par Lua
00877  * setCollisionStopType(1)
00878  *
00879  * @par JSON-RPC
00880  * {"jsonrpc":"2.0","method":"rob1.RobotConfig.setCollisionStopType","params":[1],"id":1}
00881  *
00882  * @par JSON-RPC
00883  * {"id":1,"jsonrpc":"2.0","result":0}
00884  * \endchinese
00885  * \english
00886  * Set the collision stop type.
00887  *
00888  * @param type Type \n
00889  * 0: Floating after collision, i.e., enters freedrive mode after collision
00890  * \n 1: Stop after collision \n 2: Brake after collision
00891  *
00892  * @return Returns 0 on success; error code on failure
00893  * AUBO_BUSY
00894  * AUBO_BAD_STATE
00895  * -AUBO_INVL_ARGUMENT
00896  * -AUBO_BAD_STATE
00897  *
00898  * @throws arcs::common_interface::AuboException
00899  *
00900  * @par Python function prototype
00901  * setCollisionStopType(self: pyaubo_sdk.RobotConfig, arg0: int) -> int
00902  *
00903  * @par Lua function prototype
00904  * setCollisionStopType(type: number) -> nil
00905  *
00906  * @par Lua example
00907  * setCollisionStopType(1)
00908  *
00909  * @par JSON-RPC request example
00910  * {"jsonrpc":"2.0","method":"rob1.RobotConfig.setCollisionStopType","params":[1],"id":1}
00911  *
00912  * @par JSON-RPC response example
00913  * {"id":1,"jsonrpc":"2.0","result":0}
00914  * \endenglish
00915  */
00916 int setCollisionStopType(int type);
00917
00918 /**
00919  * @ingroup RobotConfig
00920  * \chinese
00921  *
00922  *
00923  * @return \n
00924  * 0: \n
00925  * 1: \n
00926  * 2:
00927  *
00928  * @throws arcs::common_interface::AuboException
00929  *
00930  * @par Python
00931  * getCollisionStopType(self: pyaubo_sdk.RobotConfig) -> int

```

```

00932     *
00933     * @par Lua
00934     * getCollisionStopType() -> number
00935     *
00936     * @par Lua
00937     * CollisionStopType = getCollisionStopType()
00938     *
00939     * @par JSON-RPC
00940     * {"jsonrpc":"2.0","method":"rob1.RobotConfig.getCollisionStopType","params":[],"id":1}
00941     *
00942     * @par JSON-RPC
00943     * {"id":1,"jsonrpc":"2.0","result":1}
00944     *
00945     * \endchinese
00946     * \english
00947     * Get the collision stop type.
00948     *
00949     * @return The collision stop type. \n
00950     * 0: Floating after collision, i.e., enters freedrive mode after collision
00951     * \n 1: Stop after collision \n 2: Brake after collision
00952     *
00953     * @throws arcs::common_interface::AuboException
00954     *
00955     * @par Python function prototype
00956     * getCollisionStopType(self: pyaubo_sdk.RobotConfig) -> int
00957     *
00958     * @par Lua function prototype
00959     * getCollisionStopType() -> number
00960     *
00961     * @par Lua example
00962     * CollisionStopType = getCollisionStopType()
00963     *
00964     * @par JSON-RPC request example
00965     * {"jsonrpc":"2.0","method":"rob1.RobotConfig.getCollisionStopType","params":[],"id":1}
00966     *
00967     * @par JSON-RPC response example
00968     * {"id":1,"jsonrpc":"2.0","result":1}
00969     *
00970     * \endenglish
00971     */
00972 int getCollisionStopType();
00973
00974 /**
00975  * @ingroup RobotConfig
00976  * \chinese
00977  *
00978  *
00979  * @param positions
00980  * @return 0
00981  * AUBO_BUSY
00982  * AUBO_BAD_STATE
00983  * -AUBO_INVL_ARGUMENT
00984  * -AUBO_BAD_STATE
00985  *
00986  * @throws arcs::common_interface::AuboException
00987  *
00988  * @par JSON-RPC
00989  * {"jsonrpc":"2.0","method":"rob1.RobotConfig.setHomePosition","params":[[0.0,-
0.2617993877991494,1.74532925199433,0.4363323129985824,1.570796326794897,0.0]],"id":1}
00990  *
00991  * @par JSON-RPC
00992  * {"id":1,"jsonrpc":"2.0","result":0}
00993  * \endchinese
00994  * \english
00995  * Set the robot's home position.
00996  *
00997  * @param positions Joint angles
00998  * @return Returns 0 on success; error code on failure
00999  * AUBO_BUSY
01000  * AUBO_BAD_STATE
01001  * -AUBO_INVL_ARGUMENT
01002  * -AUBO_BAD_STATE
01003  *
01004  * @throws arcs::common_interface::AuboException
01005  *
01006  * @par JSON-RPC request example
01007  * {"jsonrpc":"2.0","method":"rob1.RobotConfig.setHomePosition","params":[[0.0,-
0.2617993877991494,1.74532925199433,0.4363323129985824,1.570796326794897,0.0]],"id":1}
01008  *
01009  * @par JSON-RPC response example
01010  * {"id":1,"jsonrpc":"2.0","result":0}
01011  * \endenglish
01012  */
01013 int setHomePosition(const std::vector<double> &positions);
01014
01015 /**
01016  * @ingroup RobotConfig

```

```

01017     * \chinese
01018     *
01019     *
01020     * @return
01021     *
01022     * @throws arcs::common_interface::AuboException
01023     *
01024     * @par JSON-RPC
01025     * {"jsonrpc":"2.0","method":"rob1.RobotConfig.getHomePosition","params":[],"id":1}
01026     *
01027     * @par JSON-RPC
01028     *
01029     * {"id":1,"jsonrpc":"2.0","result":[0.0,-0.2617993877991494,1.74532925199433,0.4363323129985824,1.570796326794897,0.0]}
01029     * \endchinese
01030     * \english
01031     * Get the robot's home position.
01032     *
01033     * @return
01034     *
01035     * @throws arcs::common_interface::AuboException
01036     *
01037     * @par JSON-RPC request example
01038     * {"jsonrpc":"2.0","method":"rob1.RobotConfig.getHomePosition","params":[],"id":1}
01039     *
01040     * @par JSON-RPC response example
01041     *
01042     * {"id":1,"jsonrpc":"2.0","result":[0.0,-0.2617993877991494,1.74532925199433,0.4363323129985824,1.570796326794897,0.0]}
01042     * \endenglish
01043     */
01044     std::vector<double> getHomePosition();
01045
01046     /**
01047     * @ingroup RobotConfig
01048     * \chinese
01049     *
01050     *
01051     * @param damp
01052     *
01053     * @return 0
01054     * AUBO_BUSY
01055     * AUBO_BAD_STATE
01056     * -AUBO_INVL_ARGUMENT
01057     * -AUBO_BAD_STATE
01058     *
01059     * @throws arcs::common_interface::AuboException
01060     *
01061     * @par Python
01062     * setFreedriveDamp(self: pyaubo_sdk.RobotConfig, arg0: List[float]) -> int
01063     *
01064     * @par Lua
01065     * setFreedriveDamp(damp: table) -> nil
01066     *
01067     * @par Lua
01068     * setFreedriveDamp({0.5,0.5,0.5,0.5,0.5,0.5})
01069     *
01070     * @par JSON-RPC
01071     * {"jsonrpc":"2.0","method":"rob1.RobotConfig.setFreedriveDamp","params":[[0.5,0.5,0.5,0.5,0.5,0.5]],"id":1}
01072     *
01073     * @par JSON-RPC
01074     * {"id":1,"jsonrpc":"2.0","result":0}
01075     *
01076     * \endchinese
01077     * \english
01078     * Set freedrive damping.
01079     *
01080     * @param damp Damping values.
01081     *
01082     * @return Returns 0 on success; error code on failure
01083     * AUBO_BUSY
01084     * AUBO_BAD_STATE
01085     * -AUBO_INVL_ARGUMENT
01086     * -AUBO_BAD_STATE
01087     *
01088     * @throws arcs::common_interface::AuboException
01089     *
01090     * @par Python function prototype
01091     * setFreedriveDamp(self: pyaubo_sdk.RobotConfig, arg0: List[float]) -> int
01092     *
01093     * @par Lua function prototype
01094     * setFreedriveDamp(damp: table) -> nil
01095     *
01096     * @par Lua example
01097     * setFreedriveDamp({0.5,0.5,0.5,0.5,0.5,0.5})
01098     *
01099     * @par JSON-RPC request example
01100     * {"jsonrpc":"2.0","method":"rob1.RobotConfig.setFreedriveDamp","params":[[0.5,0.5,0.5,0.5,0.5,0.5]],"id":1}
01101     *

```

```

01102     * @par JSON-RPC response example
01103     * {"id":1,"jsonrpc":"2.0","result":0}
01104     *
01105     * \endenglish
01106     */
01107 int setFreedriveDamp(const std::vector<double> &damp);
01108
01109 /**
01110  * @ingroup RobotConfig
01111  * \chinese
01112  *
01113  *
01114  * @return
01115  *
01116  * @throws arcs::common_interface::AuboException
01117  *
01118  * @par Python
01119  * getFreedriveDamp(self: pyaubo_sdk.RobotConfig) -> List[float]
01120  *
01121  * @par Lua
01122  * getFreedriveDamp() -> table
01123  *
01124  * @par Lua
01125  * FreedriveDamp = getFreedriveDamp()
01126  *
01127  * @par JSON-RPC
01128  * {"jsonrpc":"2.0","method":"rob1.RobotConfig.getFreedriveDamp","params":[],"id":1}
01129  *
01130  * @par JSON-RPC
01131  * {"id":1,"jsonrpc":"2.0","result":[0.5,0.5,0.5,0.5,0.5,0.5]}
01132  * \endchinese
01133  * \english
01134  * Get freedrive damping.
01135  *
01136  * @return Freedrive damping values.
01137  *
01138  * @throws arcs::common_interface::AuboException
01139  *
01140  * @par Python function prototype
01141  * getFreedriveDamp(self: pyaubo_sdk.RobotConfig) -> List[float]
01142  *
01143  * @par Lua function prototype
01144  * getFreedriveDamp() -> table
01145  *
01146  * @par Lua example
01147  * FreedriveDamp = getFreedriveDamp()
01148  *
01149  * @par JSON-RPC request example
01150  * {"jsonrpc":"2.0","method":"rob1.RobotConfig.getFreedriveDamp","params":[],"id":1}
01151  *
01152  * @par JSON-RPC response example
01153  * {"id":1,"jsonrpc":"2.0","result":[0.5,0.5,0.5,0.5,0.5,0.5]}
01154  * \endenglish
01155  */
01156 std::vector<double> getFreedriveDamp();
01157
01158 /**
01159  * @ingroup RobotConfig
01160  * \chinese
01161  *
01162  *
01163  * @param damp
01164  *
01165  * @return 0
01166  * AUBO_BUSY
01167  * AUBO_BAD_STATE
01168  * -AUBO_INVL_ARGUMENT
01169  * -AUBO_BAD_STATE
01170  *
01171  * @throws arcs::common_interface::AuboException
01172  *
01173  * @par Python
01174  * setHandguidDamp(self: pyaubo_sdk.RobotConfig, arg0: List[float]) -> int
01175  *
01176  * @par Lua
01177  * setHandguidDamp(damp: table) -> number
01178  *
01179  * @par Lua
01180  * setHandguidDamp({0.5,0.5,0.5,0.5,0.5,0.5})
01181  *
01182  * @par JSON-RPC
01183  * {"jsonrpc":"2.0","method":"rob1.RobotConfig.setHandguidDamp","params":[[0.5,0.5,0.5,0.5,0.5,0.5]],"id":1}
01184  *
01185  * @par JSON-RPC
01186  * {"id":1,"jsonrpc":"2.0","result":0}
01187  * \endchinese
01188  * \english

```



```

01189 * Set handguiding damping.
01190 *
01191 * @param damp damping coef.
01192 *
01193 * @return return 0 for success return rror code for failure
01194 * AUBO_BUSY
01195 * AUBO_BAD_STATE
01196 * -AUBO_INVL_ARGUMENT
01197 * -AUBO_BAD_STATE
01198 *
01199 * @throws arcs::common_interface::AuboException
01200 *
01201 * @par Python function prototype
01202 * setHandguidDamp(self: pyaubo_sdk.RobotConfig, arg0: List[float]) -> int
01203 *
01204 * @par Lua function prototype
01205 * setHandguidDamp(damp: table) -> number
01206 *
01207 * @par Lua example
01208 * setHandguidDamp({0.5,0.5,0.5,0.5,0.5,0.5})
01209 *
01210 * @par JSON-RPC request example
01211 * {"jsonrpc":"2.0","method":"rob1.RobotConfig.setHandguidDamp","params":[[0.5,0.5,0.5,0.5,0.5,0.5]],"id":1}
01212 *
01213 * @par JSON-RPC response example
01214 * {"id":1,"jsonrpc":"2.0","result":0}
01215 * \endenglish
01216 */
01217 int setHandguidDamp(const std::vector<double> &damp);
01218
01219 /**
01220 * @ingroup RobotConfig
01221 * \chinese
01222 *
01223 *
01224 * @return
01225 *
01226 * @throws arcs::common_interface::AuboException
01227 *
01228 * @par Python
01229 * getHandguidDamp(self: pyaubo_sdk.RobotConfig) -> List[float]
01230 *
01231 * @par Lua
01232 * getHandguidDamp() -> table
01233 *
01234 * @par Lua
01235 * HandguidDamp = getHandguidDamp()
01236 *
01237 * @par JSON-RPC
01238 * {"jsonrpc":"2.0","method":"rob1.RobotConfig.getHandguidDamp","params":[],"id":1}
01239 *
01240 * @par JSON-RPC
01241 * {"id":1,"jsonrpc":"2.0","result":[0.5,0.5,0.5,0.5,0.5,0.5]}
01242 * \endchinese
01243 * \english
01244 * Get handguiding damping.
01245 *
01246 * @return handguiding damping
01247 *
01248 * @throws arcs::common_interface::AuboException
01249 *
01250 * @par Python function prototype
01251 * getHandguidDamp(self: pyaubo_sdk.RobotConfig) -> List[float]
01252 *
01253 * @par Lua fuunction prototype
01254 * getHandguidDamp() -> table
01255 *
01256 * @par Lua example
01257 * HandguidDamp = getHandguidDamp()
01258 *
01259 * @par JSON-RPC request example
01260 * {"jsonrpc":"2.0","method":"rob1.RobotConfig.getHandguidDamp","params":[],"id":1}
01261 *
01262 * @par JSON-RPC response example
01263 * {"id":1,"jsonrpc":"2.0","result":[0.5,0.5,0.5,0.5,0.5,0.5]}
01264 * \endenglish
01265 */
01266 std::vector<double> getHandguidDamp();
01267
01268 /**
01269 * @ingroup RobotConfig
01270 * \chinese
01271 * DH
01272 * alpha a d theta beta
01273 *
01274 * @param real ( + )
01275 * @return DH

```

```

01276 *
01277 * @throws arcs::common_interface::AuboException
01278 *
01279 * @par Python
01280 * getKinematicsParam(self: pyaubo_sdk.RobotConfig, arg0: bool) -> Dict[str,
01281 * List[float]]
01282 *
01283 * @par Lua
01284 * getKinematicsParam(real: boolean) -> table
01285 *
01286 * @par Lua
01287 * KinematicsParam = getKinematicsParam(true)
01288 *
01289 * @par JSON-RPC
01290 * {"jsonrpc": "2.0", "method": "rob1.RobotConfig.getKinematicsParam", "params": [true], "id": 1}
01291 *
01292 * @par JSON-RPC
01293 * {"id": 1, "jsonrpc": "2.0", "result": {"a": [0.0, -0.0001255999959539622, 0.4086348000024445, 0.3757601999930339, -
0.00018950000230688602, 3.7000001611886546e-05],
01294 * "alpha": [0.0, -1.5701173967707458, 3.1440308735524347, 3.14650750358636, -
1.5703093767832055, 1.5751177669179182], "beta": [0.0, 0.0, 0.0, 0.0, 0.0, 0.0]},
01295 * "d": [0.122, 0.12154769999941345, -4.769999941345304e-05, 4.769999941345304e-05, 0.10287890000385232, 0.116],
01296 * "theta": [3.141592653589793, -1.5707963267948966, 0.0, -1.5707963267948966, 0.0, 0.0]}}
01297 * \endchinese
01298 * \english
01299 * Get the robot's DH parameters: alpha, a, d, theta, beta.
01300 *
01301 * @param real Read real parameters (theoretical + compensation) or
01302 * theoretical parameters.
01303 * @return Returns the robot's DH parameters.
01304 *
01305 * @throws arcs::common_interface::AuboException
01306 *
01307 * @par Python function prototype
01308 * getKinematicsParam(self: pyaubo_sdk.RobotConfig, arg0: bool) -> Dict[str,
01309 * List[float]]
01310 *
01311 * @par Lua function prototype
01312 * getKinematicsParam(real: boolean) -> table
01313 *
01314 * @par Lua example
01315 * KinematicsParam = getKinematicsParam(true)
01316 *
01317 * @par JSON-RPC request example
01318 * {"jsonrpc": "2.0", "method": "rob1.RobotConfig.getKinematicsParam", "params": [true], "id": 1}
01319 *
01320 * @par JSON-RPC response example
01321 * {"id": 1, "jsonrpc": "2.0", "result": {"a": [0.0, -0.0001255999959539622, 0.4086348000024445, 0.3757601999930339, -
0.00018950000230688602, 3.7000001611886546e-05],
01322 * "alpha": [0.0, -1.5701173967707458, 3.1440308735524347, 3.14650750358636, -
1.5703093767832055, 1.5751177669179182], "beta": [0.0, 0.0, 0.0, 0.0, 0.0, 0.0]},
01323 * "d": [0.122, 0.12154769999941345, -4.769999941345304e-05, 4.769999941345304e-05, 0.10287890000385232, 0.116],
01324 * "theta": [3.141592653589793, -1.5707963267948966, 0.0, -1.5707963267948966, 0.0, 0.0]}}
01325 * \endenglish
01326 */
01327 std::unordered_map<std::string, std::vector<double> getKinematicsParam(
01328     bool real);
01329
01330 /**
01331 * @ingroup RobotConfig
01332 * \chinese
01333 *     DH    :alpha a d theta beta
01334 *
01335 * @param ref_temperature    °C    20°C
01336 * @return DH
01337 *
01338 * @throws arcs::common_interface::AuboException
01339 *
01340 * @par Python
01341 * getKinematicsCompensate(self: pyaubo_sdk.RobotConfig, arg0: float) ->
01342 * Dict[str, List[float]]
01343 *
01344 * @par Lua
01345 * getKinematicsCompensate(ref_temperature: number) -> table
01346 *
01347 * @par Lua
01348 * KinematicsCompensate = getKinematicsCompensate(20)
01349 *
01350 * @par JSON-RPC
01351 * {"jsonrpc": "2.0", "method": "rob1.RobotConfig.getKinematicsCompensate", "params": [20], "id": 1}
01352 *
01353 * @par JSON-RPC
01354 * {"id": 1, "jsonrpc": "2.0", "result": {"a": [0.0, -0.0001255999959539622, 0.0006348000024445355, -0.0002398000069661066, -
0.00018950000230688602, 3.7000001611886546e-05],
01355 * "al-
pha": [0.0, 0.0, 0.000678930024150759, 0.002438219962641597, 0.0049148499965667725, 0.00048695001169107854, 0.004321440123021603],
01356 * "beta": [0.0, 0.0, 0.0, 0.0, 0.0, 0.0], "d": [0.0, 4.769999941345304e-05, -4.769999941345304e-05, 4.769999941345304e-

```

```

05,0.0003789000038523227,0.0],
01357 * "theta":[0.0,0.0,0.0,0.0,0.0,0.0]} }
01358 * \endchinese
01359 * \english
01360 * Get DH parameter compensation values (alpha, a, d, theta, beta) at the
01361 * specified temperature.
01362 *
01363 * @param ref_temperature Reference temperature in °C, default is 20°C
01364 * @return Returns DH parameter compensation values.
01365 *
01366 * @throws arcs::common_interface::AuboException
01367 *
01368 * @par Python function prototype
01369 * getKinematicsCompensate(self: pyaubo_sdk.RobotConfig, arg0: float) ->
01370 * Dict[str, List[float]]
01371 *
01372 * @par Lua function prototype
01373 * getKinematicsCompensate(ref_temperature: number) -> table
01374 *
01375 * @par Lua example
01376 * KinematicsCompensate = getKinematicsCompensate(20)
01377 *
01378 * @par JSON-RPC request example
01379 * {"jsonrpc":"2.0","method":"rob1.RobotConfig.getKinematicsCompensate","params":[20],"id":1}
01380 *
01381 * @par JSON-RPC response example
01382 * {"id":1,"jsonrpc":"2.0","result":{"a":[0.0,-0.0001255999959539622,0.0006348000024445355,-0.0002398000069661066,-
0.00018950000230688602,3.7000001611886546e-05],
01383 * "al-
pha":[0.0,0.000678930024150759,0.002438219962641597,0.0049148499965667725,0.00048695001169107854,0.004321440123021603],
01384 * "beta":[0.0,0.0,0.0,0.0,0.0,0.0],"d":[0.0,4.7699999941345304e-05,-4.7699999941345304e-05,4.7699999941345304e-
05,0.0003789000038523227,0.0],
01385 * "theta":[0.0,0.0,0.0,0.0,0.0,0.0]} }
01386 * \endenglish
01387 */
01388 std::unordered_map<std::string, std::vector<double>
01389 getKinematicsCompensate(double ref_temperature);
01390
01391 /**
01392 * @ingroup RobotConfig
01393 * \chinese
01394 * DH
01395 *
01396 * @param param
01397 *
01398 * @return 0
01399 * AUBO_BUSY
01400 * AUBO_BAD_STATE
01401 * -AUBO_INVL_ARGUMENT
01402 * -AUBO_BAD_STATE
01403 *
01404 * @throws arcs::common_interface::AuboException
01405 * \endchinese
01406 * \english
01407 * Set standard DH compensation to the robot.
01408 *
01409 * @param param
01410 *
01411 * @return Returns 0 on success; error code on failure
01412 * AUBO_BUSY
01413 * AUBO_BAD_STATE
01414 * -AUBO_INVL_ARGUMENT
01415 * -AUBO_BAD_STATE
01416 *
01417 * @throws arcs::common_interface::AuboException
01418 * \endenglish
01419 */
01420 int setKinematicsCompensate(
01421     const std::unordered_map<std::string, std::vector<double> &param);
01422
01423 /**
01424 * @ingroup RobotConfig
01425 * \chinese
01426 *
01427 *
01428 * @param param
01429 *
01430 * @return 0
01431 * AUBO_BUSY
01432 * AUBO_BAD_STATE
01433 * -AUBO_INVL_ARGUMENT
01434 * -AUBO_BAD_STATE
01435 *
01436 * @throws arcs::common_interface::AuboException
01437 *
01438 * @par Python
01439 * setPersistentParameters(self: pyaubo_sdk.RobotConfig, arg0: str) -> int

```

```

01440 *
01441 * @par Lua
01442 * setPersistentParameters(param: string) -> nil
01443 *
01444 * \endchinese
01445 * \english
01446 * Set parameters to be saved to the interface board base.
01447 *
01448 * @param param Compensation data.
01449 *
01450 * @return Returns 0 on success; error code on failure.
01451 * AUBO_BUSY
01452 * AUBO_BAD_STATE
01453 * -AUBO_INVL_ARGUMENT
01454 * -AUBO_BAD_STATE
01455 *
01456 * @throws arcs::common_interface::AuboException
01457 *
01458 * @par Python function prototype
01459 * setPersistentParameters(self: pyaubo_sdk.RobotConfig, arg0: str) -> int
01460 *
01461 * @par Lua function prototype
01462 * setPersistentParameters(param: string) -> nil
01463 *
01464 * \endenglish
01465 */
01466 int setPersistentParameters(const std::string &param);
01467
01468 /**
01469 * @ingroup RobotConfig
01470 * \chinese
01471 *
01472 *
01473 *
01474 * @param param
01475 *
01476 * @return 0
01477 * AUBO_BUSY
01478 * AUBO_BAD_STATE
01479 * -AUBO_INVL_ARGUMENT
01480 * -AUBO_BAD_STATE
01481 *
01482 * @throws arcs::common_interface::AuboException
01483 * \endchinese
01484 * \english
01485 * Set custom parameters for the hardware abstraction layer.
01486 * The purpose is to ensure compatibility between different hardware.
01487 *
01488 * @param param Custom parameters.
01489 *
01490 * @return Returns 0 on success; error code on failure.
01491 * AUBO_BUSY
01492 * AUBO_BAD_STATE
01493 * -AUBO_INVL_ARGUMENT
01494 * -AUBO_BAD_STATE
01495 *
01496 * @throws arcs::common_interface::AuboException
01497 * \endenglish
01498 */
01499 int setHardwareCustomParameters(const std::string &param);
01500
01501 /**
01502 * @ingroup RobotConfig
01503 * \chinese
01504 *
01505 *
01506 * @param
01507 * @return
01508 *
01509 * @throws arcs::common_interface::AuboException
01510 * \endchinese
01511 * \english
01512 * Get custom parameters from the hardware abstraction layer.
01513 *
01514 * @param
01515 * @return Returns custom parameters from the hardware abstraction layer.
01516 *
01517 * @throws arcs::common_interface::AuboException
01518 * \endenglish
01519 */
01520 std::string getHardwareCustomParameters(const std::string &param);
01521
01522 /**
01523 * @ingroup RobotConfig
01524 * \chinese
01525 *
01526 *

```

```

01527 * @return 0
01528 * AUBO_BUSY
01529 * AUBO_BAD_STATE
01530 * -AUBO_BAD_STATE
01531 *
01532 * @throws arcs::common_interface::AuboException
01533 *
01534 * @par Python
01535 * setRobotZero(self: pyaubo_sdk.RobotConfig) -> int
01536 *
01537 * @par Lua
01538 * setRobotZero() -> nil
01539 *
01540 * @par Lua
01541 * setRobotZero()
01542 *
01543 * @par JSON-RPC
01544 * {"jsonrpc":"2.0","method":"rob1.RobotConfig.setRobotZero","params":[],"id":1}
01545 *
01546 * @par JSON-RPC
01547 * {"id":1,"jsonrpc":"2.0","result":0}
01548 * \endchinese
01549 * \english
01550 * Set the robot joint zero position.
01551 *
01552 * @return Returns 0 on success; error code on failure
01553 * AUBO_BUSY
01554 * AUBO_BAD_STATE
01555 * -AUBO_BAD_STATE
01556 *
01557 * @throws arcs::common_interface::AuboException
01558 *
01559 * @par Python function prototype
01560 * setRobotZero(self: pyaubo_sdk.RobotConfig) -> int
01561 *
01562 * @par Lua function prototype
01563 * setRobotZero() -> nil
01564 *
01565 * @par Lua example
01566 * setRobotZero()
01567 *
01568 * @par JSON-RPC request example
01569 * {"jsonrpc":"2.0","method":"rob1.RobotConfig.setRobotZero","params":[],"id":1}
01570 *
01571 * @par JSON-RPC response example
01572 * {"id":1,"jsonrpc":"2.0","result":0}
01573 * \endenglish
01574 */
01575 int setRobotZero();
01576
01577 /**
01578 * @ingroup RobotConfig
01579 * \chinese
01580 *
01581 *
01582 * @return
01583 *
01584 * @throws arcs::common_interface::AuboException
01585 *
01586 * @par Python
01587 * getTcpForceSensorNames(self: pyaubo_sdk.RobotConfig) -> List[str]
01588 *
01589 * @par Lua
01590 * getTcpForceSensorNames() -> table
01591 *
01592 * @par Lua
01593 * TcpForceSensorNames = getTcpForceSensorNames()
01594 *
01595 * @par JSON-RPC
01596 * {"jsonrpc":"2.0","method":"rob1.RobotConfig.getTcpForceSensorNames","params":[],"id":1}
01597 *
01598 * @par JSON-RPC
01599 * {"id":1,"jsonrpc":"2.0","result":[]}
01600 *
01601 * \endchinese
01602 * \english
01603 * Get the names of available TCP force sensors.
01604 *
01605 * @return Returns the names of available TCP force sensors.
01606 *
01607 * @throws arcs::common_interface::AuboException
01608 *
01609 * @par Python function prototype
01610 * getTcpForceSensorNames(self: pyaubo_sdk.RobotConfig) -> List[str]
01611 *
01612 * @par Lua function prototype
01613 * getTcpForceSensorNames() -> table

```

```

01614 *
01615 * @par Lua example
01616 * TcpForceSensorNames = getTcpForceSensorNames()
01617 *
01618 * @par JSON-RPC request example
01619 * {"jsonrpc":"2.0","method":"rob1.RobotConfig.getTcpForceSensorNames","params":[],"id":1}
01620 *
01621 * @par JSON-RPC response example
01622 * {"id":1,"jsonrpc":"2.0","result":[]}
01623 *
01624 * \endenglish
01625 */
01626 std::vector<std::string> getTcpForceSensorNames();
01627
01628 /**
01629 * @ingroup RobotConfig
01630 * \chinese
01631 *
01632 *
01633 *
01634 * @param name
01635 *
01636 * @return 0
01637 * AUBO_BUSY
01638 * AUBO_BAD_STATE
01639 * -AUBO_INVL_ARGUMENT
01640 * -AUBO_BAD_STATE
01641 *
01642 * @throws arcs::common_interface::AuboException
01643 *
01644 * @par Python
01645 * selectTcpForceSensor(self: pyaubo_sdk.RobotConfig, arg0: str) -> int
01646 *
01647 * @par Lua
01648 * selectTcpForceSensor(name: string) -> nil
01649 * \endchinese
01650 * \english
01651 * Set the TCP force sensor.
01652 * If there is a built-in TCP force sensor, the built-in sensor will be used
01653 * by default.
01654 *
01655 * @param name Name of the TCP force sensor.
01656 *
01657 * @return Returns 0 on success; error code on failure.
01658 * AUBO_BUSY
01659 * AUBO_BAD_STATE
01660 * -AUBO_INVL_ARGUMENT
01661 * -AUBO_BAD_STATE
01662 *
01663 * @throws arcs::common_interface::AuboException
01664 *
01665 * @par Python function prototype
01666 * selectTcpForceSensor(self: pyaubo_sdk.RobotConfig, arg0: str) -> int
01667 *
01668 * @par Lua function prototype
01669 * selectTcpForceSensor(name: string) -> nil
01670 * \endenglish
01671 */
01672 int selectTcpForceSensor(const std::string &name);
01673
01674 /**
01675 * @ingroup RobotConfig
01676 * \chinese
01677 *
01678 *
01679 * @param sensor_pose
01680 *
01681 * @return 0
01682 * AUBO_BUSY
01683 * AUBO_BAD_STATE
01684 * -AUBO_INVL_ARGUMENT
01685 * -AUBO_BAD_STATE
01686 *
01687 * @throws arcs::common_interface::AuboException
01688 * \endchinese
01689 * \english
01690 * Set the sensor mounting pose.
01691 *
01692 * @param sensor_pose Sensor mounting pose
01693 *
01694 * @return Returns 0 on success; error code on failure
01695 * AUBO_BUSY
01696 * AUBO_BAD_STATE
01697 * -AUBO_INVL_ARGUMENT
01698 * -AUBO_BAD_STATE
01699 *
01700 * @throws arcs::common_interface::AuboException

```

```

01701     * \endenglish
01702     */
01703 int setTcpForceSensorPose(const std::vector<double> &sensor_pose);
01704
01705 /**
01706  * @ingroup RobotConfig
01707  * \chinese
01708  *
01709  *
01710  * @return
01711  *
01712  * @throws arcs::common_interface::AuboException
01713  *
01714  * @par JSON-RPC
01715  * {"jsonrpc":"2.0","method":"rob1.RobotConfig.getTcpForceSensorPose","params":[],"id":1}
01716  *
01717  * @par JSON-RPC
01718  * {"id":1,"jsonrpc":"2.0","result":[0.0,0.0,0.0,0.0,0.0,0.0]}
01719  * \endchinese
01720  * \english
01721  * Get the sensor mounting pose.
01722  *
01723  * @return Sensor mounting pose.
01724  *
01725  * @throws arcs::common_interface::AuboException
01726  *
01727  * @par JSON-RPC request example
01728  * {"jsonrpc":"2.0","method":"rob1.RobotConfig.getTcpForceSensorPose","params":[],"id":1}
01729  *
01730  * @par JSON-RPC response example
01731  * {"id":1,"jsonrpc":"2.0","result":[0.0,0.0,0.0,0.0,0.0,0.0]}
01732  * \endenglish
01733  */
01734 std::vector<double> getTcpForceSensorPose();
01735
01736 /**
01737  * @ingroup RobotConfig
01738  * \chinese
01739  *
01740  *
01741  * @return true; false
01742  *
01743  * @throws arcs::common_interface::AuboException
01744  *
01745  * @par Python
01746  * hasTcpForceSensor(self: pyaubo_sdk.RobotConfig) -> bool
01747  *
01748  * @par Lua
01749  * hasTcpForceSensor() -> boolean
01750  *
01751  * @par Lua
01752  * hasTcpForceSensor = hasTcpForceSensor()
01753  *
01754  * @par JSON-RPC
01755  * {"jsonrpc":"2.0","method":"rob1.RobotConfig.hasTcpForceSensor","params":[],"id":1}
01756  *
01757  * @par JSON-RPC
01758  * {"id":1,"jsonrpc":"2.0","result":true}
01759  *
01760  * \endchinese
01761  * \english
01762  * Whether a TCP force sensor is installed.
01763  *
01764  * @return Returns true if installed; false otherwise.
01765  *
01766  * @throws arcs::common_interface::AuboException
01767  *
01768  * @par Python function prototype
01769  * hasTcpForceSensor(self: pyaubo_sdk.RobotConfig) -> bool
01770  *
01771  * @par Lua function prototype
01772  * hasTcpForceSensor() -> boolean
01773  *
01774  * @par Lua example
01775  * hasTcpForceSensor = hasTcpForceSensor()
01776  *
01777  * @par JSON-RPC request example
01778  * {"jsonrpc":"2.0","method":"rob1.RobotConfig.hasTcpForceSensor","params":[],"id":1}
01779  *
01780  * @par JSON-RPC response example
01781  * {"id":1,"jsonrpc":"2.0","result":true}
01782  *
01783  * \endenglish
01784  */
01785 bool hasTcpForceSensor();
01786
01787 /**

```

```

01788 * @ingroup RobotConfig
01789 * \chinese
01790 *
01791 *
01792 * @param force_offset
01793 * @return 0;
01794 *
01795 * @throws arcs::common_interface::AuboException
01796 *
01797 * @par Python
01798 * setTcpForceOffset(self: pyaubo_sdk.RobotConfig, arg0: List[float]) -> int
01799 *
01800 * @par Lua
01801 * setTcpForceOffset(force_offset: table) -> nil
01802 *
01803 * @par Lua
01804 * setTcpForceOffset({0.0,0.0,0.0,0.0,0.0,0.0})
01805 *
01806 * \endchinese
01807 * \english
01808 * Set the TCP force/torque offset.
01809 *
01810 * @param force_offset TCP force/torque offset
01811 * @return Returns 0 on success; error code on failure
01812 *
01813 * @throws arcs::common_interface::AuboException
01814 *
01815 * @par Python function prototype
01816 * setTcpForceOffset(self: pyaubo_sdk.RobotConfig, arg0: List[float]) -> int
01817 *
01818 * @par Lua function prototype
01819 * setTcpForceOffset(force_offset: table) -> nil
01820 *
01821 * @par Lua example
01822 * setTcpForceOffset({0.0,0.0,0.0,0.0,0.0,0.0})
01823 *
01824 * \endenglish
01825 */
01826 int setTcpForceOffset(const std::vector<double> &force_offset);
01827
01828 /**
01829 * @ingroup RobotConfig
01830 * \chinese
01831 *
01832 *
01833 * @return
01834 *
01835 * @throws arcs::common_interface::AuboException
01836 *
01837 * @par Python
01838 * getTcpForceOffset(self: pyaubo_sdk.RobotConfig) -> List[float]
01839 *
01840 * @par Lua
01841 * getTcpForceOffset() -> table
01842 *
01843 * @par Lua
01844 * TcpForceOffset = getTcpForceOffset()
01845 *
01846 * @par JSON-RPC
01847 * {"jsonrpc":"2.0","method":"rob1.RobotConfig.getTcpForceOffset","params":[],"id":1}
01848 *
01849 * @par JSON-RPC
01850 * {"id":1,"jsonrpc":"2.0","result":[0.0,0.0,0.0,0.0,0.0,0.0]}
01851 * \endchinese
01852 * \english
01853 * Get the TCP force/torque offset.
01854 *
01855 * @return Returns the TCP force/torque offset.
01856 *
01857 * @throws arcs::common_interface::AuboException
01858 *
01859 * @par Python function prototype
01860 * getTcpForceOffset(self: pyaubo_sdk.RobotConfig) -> List[float]
01861 *
01862 * @par Lua function prototype
01863 * getTcpForceOffset() -> table
01864 *
01865 * @par Lua example
01866 * TcpForceOffset = getTcpForceOffset()
01867 *
01868 * @par JSON-RPC request example
01869 * {"jsonrpc":"2.0","method":"rob1.RobotConfig.getTcpForceOffset","params":[],"id":1}
01870 *
01871 * @par JSON-RPC response example
01872 * {"id":1,"jsonrpc":"2.0","result":[0.0,0.0,0.0,0.0,0.0,0.0]}
01873 * \endenglish
01874 */

```



```

01875     std::vector<double> getTcpForceOffset();
01876
01877     /**
01878     * @ingroup RobotConfig
01879     * \chinese
01880     *
01881     *
01882     * @return
01883     *
01884     * @throws arcs::common_interface::AuboException
01885     *
01886     * @par Python
01887     * getBaseForceSensorNames(self: pyaubo_sdk.RobotConfig) -> List[str]
01888     *
01889     * @par Lua
01890     * getBaseForceSensorNames() -> table
01891     *
01892     * @par Lua
01893     * BaseForceSensorNames = getBaseForceSensorNames()
01894     *
01895     * @par JSON-RPC
01896     * {"jsonrpc":"2.0","method":"rob1.RobotConfig.getBaseForceSensorNames","params":[],"id":1}
01897     *
01898     * @par JSON-RPC
01899     * {"id":1,"jsonrpc":"2.0","result":[]}
01900     *
01901     * \endchinese
01902     * \english
01903     * Get the names of available base force sensors.
01904     *
01905     * @return Returns the names of available base force sensors.
01906     *
01907     * @throws arcs::common_interface::AuboException
01908     *
01909     * @par Python function prototype
01910     * getBaseForceSensorNames(self: pyaubo_sdk.RobotConfig) -> List[str]
01911     *
01912     * @par Lua function prototype
01913     * getBaseForceSensorNames() -> table
01914     *
01915     * @par Lua example
01916     * BaseForceSensorNames = getBaseForceSensorNames()
01917     *
01918     * @par JSON-RPC request example
01919     * {"jsonrpc":"2.0","method":"rob1.RobotConfig.getBaseForceSensorNames","params":[],"id":1}
01920     *
01921     * @par JSON-RPC response example
01922     * {"id":1,"jsonrpc":"2.0","result":[]}
01923     *
01924     * \endenglish
01925     */
01926     std::vector<std::string> getBaseForceSensorNames();
01927
01928     /**
01929     * @ingroup RobotConfig
01930     * \chinese
01931     *
01932     *
01933     *
01934     * @param name
01935     *
01936     * @return 0
01937     * AUBO_BUSY
01938     * AUBO_BAD_STATE
01939     * -AUBO_INVL_ARGUMENT
01940     * -AUBO_BAD_STATE
01941     *
01942     * @throws arcs::common_interface::AuboException
01943     *
01944     * @par Python
01945     * selectBaseForceSensor(self: pyaubo_sdk.RobotConfig, arg0: str) -> int
01946     *
01947     * @par Lua
01948     * selectBaseForceSensor(name: string) -> nil
01949     * \endchinese
01950     * \english
01951     * Set the base force sensor.
01952     * If there is a built-in base force sensor, the built-in sensor will be
01953     * used by default.
01954     *
01955     * @param name Name of the base force sensor.
01956     *
01957     * @return Returns 0 on success; error code on failure.
01958     * AUBO_BUSY
01959     * AUBO_BAD_STATE
01960     * -AUBO_INVL_ARGUMENT
01961     * -AUBO_BAD_STATE

```

```

01962     *
01963     * @throws arcs::common_interface::AuboException
01964     *
01965     * @par Python function prototype
01966     * selectBaseForceSensor(self: pyaubo_sdk.RobotConfig, arg0: str) -> int
01967     *
01968     * @par Lua function prototype
01969     * selectBaseForceSensor(name: string) -> nil
01970     * \endenglish
01971     */
01972     int selectBaseForceSensor(const std::string &name);
01973
01974     /**
01975     * @ingroup RobotConfig
01976     * \chinese
01977     *
01978     *
01979     * @return true; false
01980     *
01981     * @throws arcs::common_interface::AuboException
01982     *
01983     * @par Python
01984     * hasBaseForceSensor(self: pyaubo_sdk.RobotConfig) -> bool
01985     *
01986     * @par Lua
01987     * hasBaseForceSensor() -> boolean
01988     *
01989     * @par Lua
01990     * hasBaseForceSensor = hasBaseForceSensor()
01991     *
01992     * @par JSON-RPC
01993     * {"jsonrpc":"2.0","method":"rob1.RobotConfig.hasBaseForceSensor","params":[],"id":1}
01994     *
01995     * @par JSON-RPC
01996     * {"id":1,"jsonrpc":"2.0","result":false}
01997     *
01998     * \endchinese
01999     * \english
02000     * Whether a base force sensor is installed.
02001     *
02002     * @return Returns true if installed; false otherwise.
02003     *
02004     * @throws arcs::common_interface::AuboException
02005     *
02006     * @par Python function prototype
02007     * hasBaseForceSensor(self: pyaubo_sdk.RobotConfig) -> bool
02008     *
02009     * @par Lua function prototype
02010     * hasBaseForceSensor() -> boolean
02011     *
02012     * @par Lua example
02013     * hasBaseForceSensor = hasBaseForceSensor()
02014     *
02015     * @par JSON-RPC request example
02016     * {"jsonrpc":"2.0","method":"rob1.RobotConfig.hasBaseForceSensor","params":[],"id":1}
02017     *
02018     * @par JSON-RPC response example
02019     * {"id":1,"jsonrpc":"2.0","result":false}
02020     *
02021     * \endenglish
02022     */
02023     bool hasBaseForceSensor();
02024
02025     /**
02026     * @ingroup RobotConfig
02027     * \chinese
02028     *
02029     *
02030     * @param force_offset
02031     *
02032     * @return 0
02033     * AUBO_BUSY
02034     * AUBO_BAD_STATE
02035     * -AUBO_INVL_ARGUMENT
02036     * -AUBO_BAD_STATE
02037     *
02038     * @throws arcs::common_interface::AuboException
02039     *
02040     * @par Python
02041     * setBaseForceOffset(self: pyaubo_sdk.RobotConfig, arg0: List[float]) ->
02042     * int
02043     *
02044     * @par Lua
02045     * setBaseForceOffset(force_offset: table) -> nil
02046     *
02047     * @par Lua
02048     * setBaseForceOffset({0.0,0.0,0.0,0.0,0.0,0.0})

```

```

02049  *
02050  * \endchinese
02051  * \english
02052  * Set the base force/torque offset.
02053  *
02054  * @param force_offset Base force/torque offset
02055  *
02056  * @return Returns 0 on success; error code on failure
02057  * AUBO_BUSY
02058  * AUBO_BAD_STATE
02059  * -AUBO_INVL_ARGUMENT
02060  * -AUBO_BAD_STATE
02061  *
02062  * @throws arcs::common_interface::AuboException
02063  *
02064  * @par Python function prototype
02065  * setBaseForceOffset(self: pyaubo_sdk.RobotConfig, arg0: List[float]) ->
02066  * int
02067  *
02068  * @par Lua function prototype
02069  * setBaseForceOffset(force_offset: table) -> nil
02070  *
02071  * @par Lua
02072  * setBaseForceOffset({0.0,0.0,0.0,0.0,0.0,0.0,0.0})
02073  *
02074  * \endenglish
02075  */
02076  int setBaseForceOffset(const std::vector<double> &force_offset);
02077
02078  /**
02079  * @ingroup RobotConfig
02080  * \chinese
02081  *
02082  *
02083  * @return
02084  *
02085  * @throws arcs::common_interface::AuboException
02086  *
02087  * @par Python
02088  * getBaseForceOffset(self: pyaubo_sdk.RobotConfig) -> List[float]
02089  *
02090  * @par Lua
02091  * getBaseForceOffset() -> table
02092  *
02093  * @par Lua
02094  * BaseForceOffset = getBaseForceOffset()
02095  *
02096  * @par JSON-RPC
02097  * {"jsonrpc":"2.0","method":"rob1.RobotConfig.getBaseForceOffset","params":[],"id":1}
02098  *
02099  * @par JSON-RPC
02100  * {"id":1,"jsonrpc":"2.0","result":[]}
02101  *
02102  * \endchinese
02103  * \english
02104  * Get the base force/torque offset.
02105  *
02106  * @return Returns the base force/torque offset.
02107  *
02108  * @throws arcs::common_interface::AuboException
02109  *
02110  * @par Python function prototype
02111  * getBaseForceOffset(self: pyaubo_sdk.RobotConfig) -> List[float]
02112  *
02113  * @par Lua function prototype
02114  * getBaseForceOffset() -> table
02115  *
02116  * @par Lua example
02117  * BaseForceOffset = getBaseForceOffset()
02118  *
02119  * @par JSON-RPC request example
02120  * {"jsonrpc":"2.0","method":"rob1.RobotConfig.getBaseForceOffset","params":[],"id":1}
02121  *
02122  * @par JSON-RPC response example
02123  * {"id":1,"jsonrpc":"2.0","result":[]}
02124  *
02125  * \endenglish
02126  */
02127  std::vector<double> getBaseForceOffset();
02128
02129  /**
02130  * @ingroup RobotConfig
02131  * \chinese
02132  * CRC32
02133  *
02134  * @return
02135  *

```

```

02136 * @throws arcs::common_interface::AuboException
02137 *
02138 * @par Python
02139 * getSafetyParametersChecksum(self: pyaubo_sdk.RobotConfig) -> int
02140 *
02141 * @par Lua
02142 * getSafetyParametersChecksum() -> number
02143 *
02144 * @par Lua
02145 * SafetyParametersChecksum = getSafetyParametersChecksum()
02146 *
02147 * @par JSON-RPC
02148 * {"jsonrpc":"2.0","method":"rob1.RobotConfig.getSafetyParametersChecksum","params":[],"id":1}
02149 *
02150 * @par JSON-RPC
02151 * {"id":1,"jsonrpc":"2.0","result":2033397241}
02152 *
02153 * \endchinese
02154 * \english
02155 * Get the safety parameter checksum CRC32.
02156 *
02157 * @return Returns the safety parameter checksum.
02158 *
02159 * @throws arcs::common_interface::AuboException
02160 *
02161 * @par Python function prototype
02162 * getSafetyParametersChecksum(self: pyaubo_sdk.RobotConfig) -> int
02163 *
02164 * @par Lua function prototype
02165 * getSafetyParametersChecksum() -> number
02166 *
02167 * @par Lua
02168 * SafetyParametersChecksum = getSafetyParametersChecksum()
02169 *
02170 * @par JSON-RPC request example
02171 * {"jsonrpc":"2.0","method":"rob1.RobotConfig.getSafetyParametersChecksum","params":[],"id":1}
02172 *
02173 * @par JSON-RPC response example
02174 * {"id":1,"jsonrpc":"2.0","result":2033397241}
02175 *
02176 * \endenglish
02177 */
02178 uint32_t getSafetyParametersChecksum();
02179
02180 /**
02181 * @ingroup RobotConfig
02182 * \chinese
02183 * :
02184 *     flash
02185 *
02186 * @param parameters
02187 *
02188 * @return 0
02189 * AUBO_BUSY
02190 * AUBO_BAD_STATE
02191 * -AUBO_INVL_ARGUMENT
02192 * -AUBO_BAD_STATE
02193 *
02194 * @throws arcs::common_interface::AuboException
02195 *
02196 * @par Python
02197 * confirmSafetyParameters(self: pyaubo_sdk.RobotConfig, arg0:
02198 * arcs::common_interface::RobotSafetyParameterRange) -> int
02199 *
02200 * @par Lua
02201 *
02202 * \endchinese
02203 * \english
02204 * Initiate a request to confirm safety configuration parameters:
02205 * Write safety configuration parameters to the safety interface board flash
02206 * or file.
02207 *
02208 * @param parameters Safety configuration parameters.
02209 *
02210 * @return Returns 0 on success; error code on failure.
02211 * AUBO_BUSY
02212 * AUBO_BAD_STATE
02213 * -AUBO_INVL_ARGUMENT
02214 * -AUBO_BAD_STATE
02215 *
02216 * @throws arcs::common_interface::AuboException
02217 *
02218 * @par Python function prototype
02219 * confirmSafetyParameters(self: pyaubo_sdk.RobotConfig, arg0:
02220 * arcs::common_interface::RobotSafetyParameterRange) -> int
02221 *
02222 * @par Lua function prototype

```

Generated by Doxygen

```

02308 *
02309 * @par Lua
02310 * getJointMinPositions() -> table
02311 *
02312 * @par Lua
02313 * JointMinPositions = getJointMinPositions()
02314 *
02315 * @par JSON-RPC
02316 * {"jsonrpc":"2.0","method":"rob1.RobotConfig.getJointMinPositions","params":[],"id":1}
02317 *
02318 * @par JSON-RPC
02319 * {"id":1,"jsonrpc":"2.0","result":[-6.283185307179586,-6.283185307179586,-6.283185307179586,-6.283185307179586,-
6.283185307179586,-6.283185307179586]}
02320 * \endchinese
02321 * \english
02322 * Get the joint minimum positions (physical limits).
02323 *
02324 * @return Returns the joint minimum positions.
02325 *
02326 * @throws arcs::common_interface::AuboException
02327 *
02328 * @par Python function prototype
02329 * getJointMinPositions(self: pyaubo_sdk.RobotConfig) -> List[float]
02330 *
02331 * @par Lua function prototype
02332 * getJointMinPositions() -> table
02333 *
02334 * @par Lua example
02335 * JointMinPositions = getJointMinPositions()
02336 *
02337 * @par JSON-RPC request example
02338 * {"jsonrpc":"2.0","method":"rob1.RobotConfig.getJointMinPositions","params":[],"id":1}
02339 *
02340 * @par JSON-RPC response example
02341 * {"id":1,"jsonrpc":"2.0","result":[-6.283185307179586,-6.283185307179586,-6.283185307179586,-6.283185307179586,-
6.283185307179586,-6.283185307179586]}
02342 * \endenglish
02343 */
02344 std::vector<double> getJointMinPositions();
02345
02346 /**
02347 * @ingroup RobotConfig
02348 * \chinese
02349 *
02350 *
02351 * @return
02352 *
02353 * @throws arcs::common_interface::AuboException
02354 *
02355 * @par Python
02356 * getJointMaxSpeeds(self: pyaubo_sdk.RobotConfig) -> List[float]
02357 *
02358 * @par Lua
02359 * getJointMaxSpeeds() -> table
02360 *
02361 * @par Lua
02362 * JointMaxSpeeds = getJointMaxSpeeds()
02363 *
02364 * @par JSON-RPC
02365 * {"jsonrpc":"2.0","method":"rob1.RobotConfig.getJointMaxSpeeds","params":[],"id":1}
02366 *
02367 * @par JSON-RPC
02368 * {"id":1,"jsonrpc":"2.0","result":[3.892084231947355,3.892084231947355,3.892084231947355,3.1066860685499065,3.1066860685499065,3.1066860685499065]}
02369 * \endchinese
02370 * \english
02371 * Get the joint maximum speeds (physical limits).
02372 *
02373 * @return Returns the joint maximum speeds.
02374 *
02375 * @throws arcs::common_interface::AuboException
02376 *
02377 * @par Python function prototype
02378 * getJointMaxSpeeds(self: pyaubo_sdk.RobotConfig) -> List[float]
02379 *
02380 * @par Lua function prototype
02381 * getJointMaxSpeeds() -> table
02382 *
02383 * @par Lua example
02384 * JointMaxSpeeds = getJointMaxSpeeds()
02385 *
02386 * @par JSON-RPC request example
02387 * {"jsonrpc":"2.0","method":"rob1.RobotConfig.getJointMaxSpeeds","params":[],"id":1}
02388 *
02389 * @par JSON-RPC response example
02390 * {"id":1,"jsonrpc":"2.0","result":[3.892084231947355,3.892084231947355,3.892084231947355,3.1066860685499065,3.1066860685499065,3.1066860685499065]}

```

```

02391     * \endenglish
02392     */
02393     std::vector<double> getJointMaxSpeeds();
02394
02395     /**
02396     * @ingroup RobotConfig
02397     * \chinese
02398     *
02399     *
02400     * @return
02401     *
02402     * @throws arcs::common_interface::AuboException
02403     *
02404     * @par Python
02405     * getJointMaxAccelerations(self: pyaubo_sdk.RobotConfig) -> List[float]
02406     *
02407     * @par Lua
02408     * getJointMaxAccelerations() -> table
02409     *
02410     * @par Lua
02411     * JointMaxAccelerations = getJointMaxAccelerations()
02412     *
02413     * @par JSON-RPC
02414     * {"jsonrpc":"2.0","method":"rob1.RobotConfig.getJointMaxAccelerations","params":[],"id":1}
02415     *
02416     * @par JSON-RPC
02417     * {"id":1,"jsonrpc":"2.0","result":[31.104877758314785,31.104877758314785,31.104877758314785,20.73625684294463,20.73625684294463,20.73625684294463]}
02418     * \endchinese
02419     * \english
02420     * Get the joint maximum accelerations (physical limits).
02421     *
02422     * @return Returns the joint maximum accelerations.
02423     *
02424     * @throws arcs::common_interface::AuboException
02425     *
02426     * @par Python function prototype
02427     * getJointMaxAccelerations(self: pyaubo_sdk.RobotConfig) -> List[float]
02428     *
02429     * @par Lua function prototype
02430     * getJointMaxAccelerations() -> table
02431     *
02432     * @par Lua example
02433     * JointMaxAccelerations = getJointMaxAccelerations()
02434     *
02435     * @par JSON-RPC request example
02436     * {"jsonrpc":"2.0","method":"rob1.RobotConfig.getJointMaxAccelerations","params":[],"id":1}
02437     *
02438     * @par JSON-RPC response example
02439     * {"id":1,"jsonrpc":"2.0","result":[31.104877758314785,31.104877758314785,31.104877758314785,20.73625684294463,20.73625684294463,20.73625684294463]}
02440     * \endenglish
02441     */
02442     std::vector<double> getJointMaxAccelerations();
02443
02444     /**
02445     * @ingroup RobotConfig
02446     * \chinese
02447     * TCP
02448     *
02449     * @return TCP
02450     *
02451     * @throws arcs::common_interface::AuboException
02452     *
02453     * @par Python
02454     * getTcpMaxSpeeds(self: pyaubo_sdk.RobotConfig) -> List[float]
02455     *
02456     * @par Lua
02457     * getTcpMaxSpeeds() -> table
02458     *
02459     * @par Lua
02460     * TcpMaxSpeeds = getTcpMaxSpeeds()
02461     *
02462     * @par JSON-RPC
02463     * {"jsonrpc":"2.0","method":"rob1.RobotConfig.getTcpMaxSpeeds","params":[],"id":1}
02464     *
02465     * @par JSON-RPC
02466     * {"id":1,"jsonrpc":"2.0","result":[2.0,5.0]}
02467     * \endchinese
02468     * \english
02469     * Get the TCP maximum speed (physical limits).
02470     *
02471     * @return Returns the TCP maximum speed.
02472     *
02473     * @throws arcs::common_interface::AuboException
02474     *
02475     * @par Python function prototype

```

```

02476 * getTcpMaxSpeeds(self: pyaubo_sdk.RobotConfig) -> List[float]
02477 *
02478 * @par Lua function prototype
02479 * getTcpMaxSpeeds() -> table
02480 *
02481 * @par Lua example
02482 * TcpMaxSpeeds = getTcpMaxSpeeds()
02483 *
02484 * @par JSON-RPC request example
02485 * {"jsonrpc":"2.0","method":"rob1.RobotConfig.getTcpMaxSpeeds","params":[],"id":1}
02486 *
02487 * @par JSON-RPC response example
02488 * {"id":1,"jsonrpc":"2.0","result":[2.0,5.0]}
02489 * \endenglish
02490 */
02491 std::vector<double> getTcpMaxSpeeds();
02492
02493 /**
02494 * @ingroup RobotConfig
02495 * \chinese
02496 * TCP
02497 *
02498 * @return TCP
02499 *
02500 * @throws arcs::common_interface::AuboException
02501 *
02502 * @par Python
02503 * getTcpMaxAccelerations(self: pyaubo_sdk.RobotConfig) -> List[float]
02504 *
02505 * @par Lua
02506 * getTcpMaxAccelerations() -> table
02507 *
02508 * @par Lua
02509 * TcpMaxAccelerations = getTcpMaxAccelerations()
02510 *
02511 * @par JSON-RPC
02512 * {"jsonrpc":"2.0","method":"rob1.RobotConfig.getTcpMaxAccelerations","params":[],"id":1}
02513 *
02514 * @par JSON-RPC
02515 * {"id":1,"jsonrpc":"2.0","result":[10.0,10.0]}
02516 *
02517 * \endchinese
02518 * \english
02519 * Get the TCP maximum acceleration (physical limits).
02520 *
02521 * @return Returns the TCP maximum acceleration.
02522 *
02523 * @throws arcs::common_interface::AuboException
02524 *
02525 * @par Python function prototype
02526 * getTcpMaxAccelerations(self: pyaubo_sdk.RobotConfig) -> List[float]
02527 *
02528 * @par Lua function prototype
02529 * getTcpMaxAccelerations() -> table
02530 *
02531 * @par Lua example
02532 * TcpMaxAccelerations = getTcpMaxAccelerations()
02533 *
02534 * @par JSON-RPC request example
02535 * {"jsonrpc":"2.0","method":"rob1.RobotConfig.getTcpMaxAccelerations","params":[],"id":1}
02536 *
02537 * @par JSON-RPC response example
02538 * {"id":1,"jsonrpc":"2.0","result":[10.0,10.0]}
02539 * \endenglish
02540 */
02541 std::vector<double> getTcpMaxAccelerations();
02542
02543 /**
02544 * @ingroup RobotConfig
02545 * \chinese
02546 *
02547 *
02548 * @param gravity
02549 *
02550 * @return 0
02551 * AUBO_BUSY
02552 * AUBO_BAD_STATE
02553 * -AUBO_INVL_ARGUMENT
02554 * -AUBO_BAD_STATE
02555 *
02556 * @throws arcs::common_interface::AuboException
02557 *
02558 * @par Python
02559 * setGravity(self: pyaubo_sdk.RobotConfig, arg0: List[float]) -> int
02560 *
02561 * @par Lua
02562 * setGravity(gravity: table) -> nil

```



```

02563 *
02564 * @par Lua
02565 * setGravity({0.0,0.0,-9.87654321})
02566 *
02567 * @par JSON-RPC
02568 * {"jsonrpc":"2.0","method":"rob1.RobotConfig.setGravity","params":[[0.0,0.0,-9.87654321]],"id":1}
02569 *
02570 * @par JSON-RPC
02571 * {"id":1,"jsonrpc":"2.0","result":0}
02572 * \endchinese
02573 * \english
02574 * Set the robot installation attitude (gravity vector).
02575 *
02576 * @param gravity Installation attitude (gravity vector)
02577 *
02578 * @return Returns 0 on success; error code on failure
02579 * AUBO_BUSY
02580 * AUBO_BAD_STATE
02581 * -AUBO_INVL_ARGUMENT
02582 * -AUBO_BAD_STATE
02583 *
02584 * @throws arcs::common_interface::AuboException
02585 *
02586 * @par Python function prototype
02587 * setGravity(self: pyaubo_sdk.RobotConfig, arg0: List[float]) -> int
02588 *
02589 * @par Lua function prototype
02590 * setGravity(gravity: table) -> nil
02591 *
02592 * @par Lua example
02593 * setGravity({0.0,0.0,-9.87654321})
02594 *
02595 * @par JSON-RPC request example
02596 * {"jsonrpc":"2.0","method":"rob1.RobotConfig.setGravity","params":[[0.0,0.0,-9.87654321]],"id":1}
02597 *
02598 * @par JSON-RPC response example
02599 * {"id":1,"jsonrpc":"2.0","result":0}
02600 * \endenglish
02601 */
02602 int setGravity(const std::vector<double> &gravity);
02603
02604 /**
02605 * @ingroup RobotConfig
02606 * \chinese
02607 *
02608 *
02609 *
02610 *
02611 * @return
02612 *
02613 * @throws arcs::common_interface::AuboException
02614 *
02615 * @par Python
02616 * getGravity(self: pyaubo_sdk.RobotConfig) -> List[float]
02617 *
02618 * @par Lua
02619 * getGravity() -> table
02620 *
02621 * @par Lua
02622 * Gravity = getGravity()
02623 *
02624 * @par JSON-RPC
02625 * {"jsonrpc":"2.0","method":"rob1.RobotConfig.getGravity","params":[],"id":1}
02626 *
02627 * @par JSON-RPC
02628 * {"id":1,"jsonrpc":"2.0","result":[0.0,0.0,-9.87654321]}
02629 *
02630 * \endchinese
02631 * \english
02632 * Get the robot's installation attitude (gravity vector).
02633 *
02634 * If the robot base is equipped with an attitude sensor, data is read from
02635 * the sensor; otherwise, the user setting is used.
02636 *
02637 * @return Returns the installation attitude.
02638 *
02639 * @throws arcs::common_interface::AuboException
02640 *
02641 * @par Python function prototype
02642 * getGravity(self: pyaubo_sdk.RobotConfig) -> List[float]
02643 *
02644 * @par Lua function prototype
02645 * getGravity() -> table
02646 *
02647 * @par Lua example
02648 * Gravity = getGravity()
02649 *

```

```

02650 * @par JSON-RPC request example
02651 * {"jsonrpc":"2.0","method":"rob1.RobotConfig.getGravity","params":[],"id":1}
02652 *
02653 * @par JSON-RPC response example
02654 * {"id":1,"jsonrpc":"2.0","result":[0.0,0.0,-9.87654321]}
02655 *
02656 * \endenglish
02657 */
02658 std::vector<double> getGravity();
02659 /**
02660 * @ingroup RobotConfig
02661 * \chinese
02662 * TCP
02663 *
02664 *
02665 * TCP (x,y,z,rx,ry,rz)
02666 * x y z TCP m
02667 * rx ry rz TCP ZYX rad
02668 *
02669 * @param offset TCP , (x,y,z,rx,ry,rz)
02670 *
02671 * @return 0
02672 * AUBO_BUSY
02673 * AUBO_BAD_STATE
02674 * -AUBO_INVL_ARGUMENT
02675 * -AUBO_BAD_STATE
02676 *
02677 * @throws arcs::common_interface::AuboException
02678 *
02679 * @par Python
02680 * setTcpOffset(self: pyaubo_sdk.RobotConfig, arg0: List[float]) -> int
02681 *
02682 * @par Lua
02683 * setTcpOffset(offset: table) -> nil
02684 *
02685 * @par Lua
02686 * setTcpOffset({0.0,0.0,0.0,0.0,0.0,0.0})
02687 *
02688 * @par JSON-RPC
02689 * {"jsonrpc":"2.0","method":"rob1.RobotConfig.setTcpOffset","params":[[0.0,0.0,0.0,0.0,0.0,0.0]],"id":1}
02690 *
02691 * @par JSON-RPC
02692 * {"id":1,"jsonrpc":"2.0","result":0}
02693 * \endchinese
02694 * \english
02695 * Set the current TCP offset.
02696 *
02697 * The TCP offset is represented as (x, y, z, rx, ry, rz).
02698 * x, y, z are the position offsets of the Tool Center Point (TCP) relative
02699 * to the flange center in the base coordinate system, in meters. rx, ry, rz
02700 * are the orientation offsets of the TCP relative to the flange center in
02701 * the base coordinate system, as ZYX Euler angles in radians.
02702 *
02703 * @param offset The current TCP offset, in the form (x, y, z, rx, ry, rz)
02704 *
02705 * @return Returns 0 on success; error code on failure
02706 * AUBO_BUSY
02707 * AUBO_BAD_STATE
02708 * -AUBO_INVL_ARGUMENT
02709 * -AUBO_BAD_STATE
02710 *
02711 * @throws arcs::common_interface::AuboException
02712 *
02713 * @par Python function prototype
02714 * setTcpOffset(self: pyaubo_sdk.RobotConfig, arg0: List[float]) -> int
02715 *
02716 * @par Lua function prototype
02717 * setTcpOffset(offset: table) -> nil
02718 *
02719 * @par Lua example
02720 * setTcpOffset({0.0,0.0,0.0,0.0,0.0,0.0})
02721 *
02722 * @par JSON-RPC request example
02723 * {"jsonrpc":"2.0","method":"rob1.RobotConfig.setTcpOffset","params":[[0.0,0.0,0.0,0.0,0.0,0.0]],"id":1}
02724 *
02725 * @par JSON-RPC response example
02726 * {"id":1,"jsonrpc":"2.0","result":0}
02727 * \endenglish
02728 */
02729 int setTcpOffset(const std::vector<double> &offset);
02730 /**
02731 * @ingroup RobotConfig
02732 * \chinese
02733 * TCP
02734 *
02735 *
02736 * TCP (x,y,z,rx,ry,rz)

```

```

02737 * x y z TCP m
02738 * rx ry rz TCP ZYX rad
02739 *
02740 * @return TCP , (x,y,z,rx,ry,rz)
02741 *
02742 * @throws arcs::common_interface::AuboException
02743 *
02744 * @par Python
02745 * getTcpOffset(self: pyaubo_sdk.RobotConfig) -> List[float]
02746 *
02747 * @par Lua
02748 * getTcpOffset() -> table
02749 *
02750 * @par Lua
02751 * TcpOffset = getTcpOffset()
02752 *
02753 * @par JSON-RPC
02754 * {"jsonrpc":"2.0","method":"rob1.RobotConfig.getTcpOffset","params":[],"id":1}
02755 *
02756 * @par JSON-RPC
02757 * {"id":1,"jsonrpc":"2.0","result":[0.0,0.0,0.0,0.0,0.0,0.0]}
02758 * \endchinese
02759 * \english
02760 * Get the current TCP offset.
02761 *
02762 * The TCP offset is represented as (x, y, z, rx, ry, rz).
02763 * x, y, z are the position offsets of the Tool Center Point (TCP) relative
02764 * to the flange center in the base coordinate system, in meters. rx, ry, rz
02765 * are the orientation offsets of the TCP relative to the flange center in
02766 * the base coordinate system, as ZYX Euler angles in radians.
02767 *
02768 * @return The current TCP offset, in the form (x, y, z, rx, ry, rz)
02769 *
02770 * @throws arcs::common_interface::AuboException
02771 *
02772 * @par Python function prototype
02773 * getTcpOffset(self: pyaubo_sdk.RobotConfig) -> List[float]
02774 *
02775 * @par Lua function prototype
02776 * getTcpOffset() -> table
02777 *
02778 * @par Lua example
02779 * TcpOffset = getTcpOffset()
02780 *
02781 * @par JSON-RPC request example
02782 * {"jsonrpc":"2.0","method":"rob1.RobotConfig.getTcpOffset","params":[],"id":1}
02783 *
02784 * @par JSON-RPC response example
02785 * {"id":1,"jsonrpc":"2.0","result":[0.0,0.0,0.0,0.0,0.0,0.0]}
02786 * \endenglish
02787 */
02788 std::vector<double> getTcpOffset();
02789
02790 /**
02791 * @ingroup RobotConfig
02792 * \chinese
02793 *
02794 *
02795 * @param m
02796 * @param com
02797 * @param inertial
02798 *
02799 * @return 0
02800 * AUBO_BUSY
02801 * AUBO_BAD_STATE
02802 * -AUBO_INVL_ARGUMENT
02803 * -AUBO_BAD_STATE
02804 *
02805 * @throws arcs::common_interface::AuboException
02806 *
02807 * @par Python
02808 * setToolInertial(self: pyaubo_sdk.RobotConfig, arg0: float, arg1:
02809 * List[float], arg2: List[float]) -> int
02810 *
02811 * @par Lua
02812 * setToolInertial(m: number, com: table, inertial: table) -> nil
02813 *
02814 * @par Lua
02815 * setToolInertial(3, {0,0,0}, {0,0,0}, {0,0,0,0,0,0,0,0})
02816 *
02817 * \endchinese
02818 * \english
02819 * Set tool mass, center of mass, and inertia.
02820 *
02821 * @param m Tool mass
02822 * @param com Center of mass
02823 * @param inertial Inertia

```

```

02824 *
02825 * @return Returns 0 on success; error code on failure
02826 * AUBO_BUSY
02827 * AUBO_BAD_STATE
02828 * -AUBO_INVL_ARGUMENT
02829 * -AUBO_BAD_STATE
02830 *
02831 * @throws arcs::common_interface::AuboException
02832 *
02833 * @par Python function prototype
02834 * setToolInertial(self: pyaubo_sdk.RobotConfig, arg0: float, arg1:
02835 * List[float], arg2: List[float]) -> int
02836 *
02837 * @par Lua function prototype
02838 * setToolInertial(m: number, com: table, inertial: table) -> nil
02839 *
02840 * @par Lua example
02841 * setToolInertial(3, {0,0,0}, {0,0,0}, {0,0,0,0,0,0,0,0})
02842 *
02843 * \endenglish
02844 */
02845 int setToolInertial(double m, const std::vector<double> &com,
02846                    const std::vector<double> &inertial);
02847
02848 /**
02849 * @ingroup RobotConfig
02850 * \chinese
02851 *
02852 *
02853 * @param m , : kg
02854 * @param cog , : m, (CoGx, CoGy, CoGz)
02855 * @param aom , : rad, (rx, ry, rz)
02856 * @param inertia , : kg*m^2, (Ixx, Iyy, Izz, Ixy, Ixz,
02857 * Iyz) (Ixx, Ixy, Ixz, Iyx, Iyy, Iyz, Izx, Izy, Izz)
02858 *
02859 * @return 0
02860 * AUBO_BUSY
02861 * AUBO_BAD_STATE
02862 * -AUBO_INVL_ARGUMENT
02863 * -AUBO_BAD_STATE
02864 *
02865 * @throws arcs::common_interface::AuboException
02866 *
02867 * @par Python
02868 * setPayload(self: pyaubo_sdk.RobotConfig, arg0: float, arg1: List[float],
02869 * arg2: List[float], arg3: List[float]) -> int
02870 *
02871 * @par Lua
02872 * setPayload(m: number, cog: table, aom: table, inertia: table) -> nil
02873 *
02874 * @par Lua
02875 * setPayload(3, {0,0,0}, {0,0,0}, {0,0,0,0,0,0,0,0})
02876 *
02877 * @par JSON-RPC
02878 * {"jsonrpc":"2.0","method":"rob1.RobotConfig.setPayload","params":[3,[0,0,0],[0,0,0],[0,0,0,0,0,0,0,0]], "id":1}
02879 *
02880 * @par JSON-RPC
02881 * {"id":1,"jsonrpc":"2.0","result":0}
02882 * \endchinese
02883 * \english
02884 * Set the payload.
02885 *
02886 * @param m Mass, unit: kg
02887 * @param cog Center of gravity, unit: m, format (CoGx, CoGy, CoGz)
02888 * @param aom Axis of moment, unit: rad, format (rx, ry, rz)
02889 * @param inertia Inertia, unit: kg*m^2, format (Ixx, Iyy, Izz, Ixy, Ixz,
02890 * Iyz) or (Ixx, Ixy, Ixz, Iyx, Iyy, Iyz, Izx, Izy, Izz)
02891 *
02892 * @return Returns 0 on success; error code on failure
02893 * AUBO_BUSY
02894 * AUBO_BAD_STATE
02895 * -AUBO_INVL_ARGUMENT
02896 * -AUBO_BAD_STATE
02897 *
02898 * @throws arcs::common_interface::AuboException
02899 *
02900 * @par Python function prototype
02901 * setPayload(self: pyaubo_sdk.RobotConfig, arg0: float, arg1: List[float],
02902 * arg2: List[float], arg3: List[float]) -> int
02903 *
02904 * @par Lua function prototype
02905 * setPayload(m: number, cog: table, aom: table, inertia: table) -> nil
02906 *
02907 * @par Lua example
02908 * setPayload(3, {0,0,0}, {0,0,0}, {0,0,0,0,0,0,0,0})
02909 *
02910 * @par JSON-RPC request example

```

```

02911     * {"jsonrpc":"2.0","method":"rob1.RobotConfig.setPayload","params":[3,[0,0,0],[0,0,0],[0,0,0,0,0,0,0,0,0]],"id":1}
02912     *
02913     * @par JSON-RPC response example
02914     * {"id":1,"jsonrpc":"2.0","result":0}
02915     * \endenglish
02916     */
02917 int setPayload(double m, const std::vector<double> &cog,
02918               const std::vector<double> &aom,
02919               const std::vector<double> &inertia);
02920
02921 /**
02922  * @ingroup RobotConfig
02923  * \chinese
02924  *
02925  *
02926  * @return .
02927  *      , : kg;
02928  *      , : m, (CoGx, CoGy, CoGz);
02929  *      , : rad, (rx, ry, rz);
02930  *      , : kg*m^2, (Ixx, Iyy, Izz, Ixy, Ixz, Iyz)
02931  *
02932  * @throws arcs::common_interface::AuboException
02933  *
02934  * @par Python
02935  * getPayload(self: pyaubo_sdk.RobotConfig) -> Tuple[float, List[float],
02936  * List[float], List[float]]
02937  *
02938  * @par Lua
02939  * getPayload() -> number, table, table, table
02940  *
02941  * @par Lua
02942  * m, cog, aom, inertia = getPayload()
02943  *
02944  * @par JSON-RPC
02945  * {"jsonrpc":"2.0","method":"rob1.RobotConfig.getPayload","params":[],"id":1}
02946  *
02947  * @par JSON-RPC
02948  * {"id":1,"jsonrpc":"2.0","result":[3.0,[0.0,0.0,0.0],[0.0,0.0,0.0],[0.0,0.0,0.0,0.0,0.0,0.0]]}
02949  * \endchinese
02950  * \english
02951  * Get the payload.
02952  *
02953  * @return The payload.
02954  * The first element is the mass in kg;
02955  * The second element is the center of gravity in meters, format (CoGx,
02956  * CoGy, CoGz); The third element is the axis of moment in radians, format
02957  * (rx, ry, rz); The fourth element is the inertia in kg*m^2, format (Ixx,
02958  * Iyy, Izz, Ixy, Ixz, Iyz)
02959  *
02960  * @throws arcs::common_interface::AuboException
02961  *
02962  * @par Python function prototype
02963  * getPayload(self: pyaubo_sdk.RobotConfig) -> Tuple[float, List[float],
02964  * List[float], List[float]]
02965  *
02966  * @par Lua function prototype
02967  * getPayload() -> number, table, table, table
02968  *
02969  * @par Lua example
02970  * m, cog, aom, inertia = getPayload()
02971  *
02972  * @par JSON-RPC request example
02973  * {"jsonrpc":"2.0","method":"rob1.RobotConfig.getPayload","params":[],"id":1}
02974  *
02975  * @par JSON-RPC response example
02976  * {"id":1,"jsonrpc":"2.0","result":[3.0,[0.0,0.0,0.0],[0.0,0.0,0.0],[0.0,0.0,0.0,0.0,0.0,0.0]]}
02977  * \endenglish
02978  */
02979 Payload getPayload();
02980
02981 /**
02982  * @ingroup RobotConfig
02983  * \chinese
02984  *
02985  *
02986  * @param pose
02987  * @return true; false
02988  *
02989  * @throws arcs::common_interface::AuboException
02990  *
02991  * @par Python
02992  * toolSpaceInRange(self: pyaubo_sdk.RobotConfig, arg0: List[float]) -> bool
02993  *
02994  * @par Lua
02995  * toolSpaceInRange(pose: table) -> boolean
02996  *
02997  * @par Lua

```

```

02998 * SpaceInRange = toolSpaceInRange({0.58712,-0.15775, 0.48703, 2.76,
02999 * 0.344, 1.432})
03000 *
03001 * @par JSON-RPC
03002 * {"jsonrpc":"2.0","method":"rob1.RobotConfig.toolSpaceInRange","params":[[0.58712,
03003 * -0.15775, 0.48703, 2.76, 0.344, 1.432]],"id":1}
03004 *
03005 * @par JSON-RPC
03006 * {"id":1,"jsonrpc":"2.0","result":true}
03007 * \endchinese
03008 * \english
03009 * Whether the end-effector pose is within the safety range.
03010 *
03011 * @param pose End-effector pose
03012 * @return Returns true if within the safety range; otherwise returns false
03013 *
03014 * @throws arcs::common_interface::AuboException
03015 *
03016 * @par Python function prototype
03017 * toolSpaceInRange(self: pyaubo_sdk.RobotConfig, arg0: List[float]) -> bool
03018 *
03019 * @par Lua function prototype
03020 * toolSpaceInRange(pose: table) -> boolean
03021 *
03022 * @par Lua example
03023 * SpaceInRange = toolSpaceInRange({0.58712,-0.15775, 0.48703, 2.76,
03024 * 0.344, 1.432})
03025 *
03026 * @par JSON-RPC request example
03027 * {"jsonrpc":"2.0","method":"rob1.RobotConfig.toolSpaceInRange","params":[[0.58712,
03028 * -0.15775, 0.48703, 2.76, 0.344, 1.432]],"id":1}
03029 *
03030 * @par JSON-RPC response example
03031 * {"id":1,"jsonrpc":"2.0","result":true}
03032 * \endenglish
03033 */
03034 bool toolSpaceInRange(const std::vector<double> &pose);
03035
03036 /**
03037 * @ingroup RobotConfig
03038 * \chinese
03039 *
03040 *
03041 * @param fw          : \#
03042 *          (#)      (.)
03043 *          ! ( )
03044 *
03045 *          \n
03046 *          \n
03047 *
03048 * /tmp/firmware_update-1.0.42-rc.5+2347b0d.firm\#master_mcu!,slace_mcu!,
03049 * base!,tool!,joint1!,joint2!,joint3!,joint4!,joint5!,joint6!
03050 *          6 joint1 joint6 \n
03051 * all
03052 * /tmp/firm_XXX.firm\#all
03053 * /tmp/firm_XXX.firm\#all
03054 *
03055 * @return 0; \n
03056 * -AUBO_BAD_STATE: (RuntimeMachine) Stopped,
03057 *          AUBO_BAD_STATE 1 \n
03058 * -AUBO_TIMEOUT: AUBO_TIMEOUT 4 \n
03059 *
03060 * @throws arcs::common_interface::AuboException
03061 *
03062 * @par Python
03063 * firmwareUpdate(self: pyaubo_sdk.RobotConfig, arg0: str) -> int
03064 *
03065 * @par Lua
03066 * firmwareUpdate(fw: string) -> nil
03067 *
03068 * @par Lua
03069 * firmwareUpdate("/tmp/firmware_update-1.0.42-rc.12+3e33eac.firm#master_mcu")
03070 *
03071 * @par JSON-RPC
03072 * {"jsonrpc":"2.0","method":"rob1.RobotConfig.firmwareUpdate","params":["/tmp/firmware_update-1.0.42-
rc.12+3e33eac.firm#master_mcu"],"id":1}
03073 *
03074 * @par JSON-RPC
03075 * {"id":1,"jsonrpc":"2.0","result":0}
03076 * \endchinese
03077 * \english
03078 * Initiate a firmware upgrade request. The controller software will enter
03079 * firmware upgrade mode.
03080 *
03081 * @param fw Firmware upgrade path. The format is: firmware package
03082 * path\#upgrade node list. The firmware package path and node list are
03083 * separated by a hash (#), and nodes are separated by commas (.). If a node

```

```

03084 * name ends with !, it means to force upgrade (without version check) that
03085 * node; otherwise, the node will be upgraded only if the current version
03086 * differs from the target version. You can flexibly specify which nodes to
03087 * upgrade as needed. For example,
03088 * /tmp/firmware_update-1.0.42-
rc.5+2347b0d.firm#master_mcu!,slace_mcu!,base!,tool!,joint1!,joint2!,joint3!,joint4!,joint5!,joint6!
03089 * means to force upgrade the interface board main, interface board slave,
03090 * base, tool, and joints 1-6. "all" means all nodes, e.g.
03091 * /tmp/firm_XXX.firm#all means upgrade all nodes with version check,
03092 * /tmp/firm_XXX.firm#all! means force upgrade all nodes.
03093 *
03094 * @return Returns 0 if the command is sent successfully; otherwise, returns
03095 * an error code. -AUBO_BAD_STATE: The current state of the RuntimeMachine
03096 * is not Stopped, so the firmware upgrade request is rejected.
03097 * AUBO_BAD_STATE = 1. -AUBO_TIMEOUT: Timeout. AUBO_TIMEOUT = 4.
03098 *
03099 * @throws arcs::common_interface::AuboException
03100 *
03101 * @par Python function prototype
03102 * firmwareUpdate(self: pyaubo_sdk.RobotConfig, arg0: str) -> int
03103 *
03104 * @par Lua function prototype
03105 * firmwareUpdate(fw: string) -> nil
03106 *
03107 * @par Lua example
03108 * firmwareUpdate("/tmp/firmware_update-1.0.42-rc.12+3e33eac.firm#master_mcu")
03109 *
03110 * @par JSON-RPC request example
03111 * {"jsonrpc": "2.0", "method": "rob1.RobotConfig.firmwareUpdate", "params": ["/tmp/firmware_update-1.0.42-
rc.12+3e33eac.firm#master_mcu"], "id": 1}
03112 *
03113 * @par JSON-RPC response example
03114 * {"id": 1, "jsonrpc": "2.0", "result": 0}
03115 * \endenglish
03116 */
03117 int firmwareUpdate(const std::string &fw);
03118
03119 /**
03120 * @ingroup RobotConfig
03121 * \chinese
03122 *
03123 * @return \n
03124 *         failed \n
03125 *         (0~1)  ("", 1)
03126 *
03127 * @throws arcs::common_interface::AuboException
03128 *
03129 *
03130 * @par Python
03131 * getFirmwareUpdateProcess(self: pyaubo_sdk.RobotConfig) -> Tuple[str,
03132 * float]
03133 *
03134 * @par Lua
03135 * getFirmwareUpdateProcess() -> table
03136 *
03137 * @par Lua
03138 * step,progress = getFirmwareUpdateProcess()
03139 *
03140 * @par JSON-RPC
03141 * {"jsonrpc": "2.0", "method": "rob1.RobotConfig.getFirmwareUpdateProcess", "params": [], "id": 1}
03142 *
03143 * @par JSON-RPC
03144 * {"id": 1, "jsonrpc": "2.0", "result": ["", 0.0]}
03145 *
03146 * \endchinese
03147 * \english
03148 * Get the current firmware upgrade process.
03149 *
03150 * @return The current firmware upgrade process. \n
03151 * The first element is the step name. If it is "failed", the firmware
03152 * upgrade failed. \n The second element is the upgrade progress (0~1). When
03153 * finished, returns ("", 1)
03154 *
03155 * @throws arcs::common_interface::AuboException
03156 *
03157 * @par Python function prototype
03158 * getFirmwareUpdateProcess(self: pyaubo_sdk.RobotConfig) -> Tuple[str,
03159 * float]
03160 *
03161 * @par Lua function prototype
03162 * getFirmwareUpdateProcess() -> table
03163 *
03164 * @par Lua example
03165 * step,progress = getFirmwareUpdateProcess()
03166 *
03167 * @par JSON-RPC request example
03168 * {"jsonrpc": "2.0", "method": "rob1.RobotConfig.getFirmwareUpdateProcess", "params": [], "id": 1}

```

```

03169     *
03170     * @par JSON-RPC response example
03171     * {"id":1,"jsonrpc":"2.0","result":[" ",0.0]}
03172     *
03173     * \endenglish
03174     */
03175     std::tuple<std::string, double> getFirmwareUpdateProcess();
03176
03177     /**
03178     * @ingroup RobotConfig
03179     * \chinese
03180     *
03181     *
03182     * @return
03183     *
03184     * @throws arcs::common_interface::AuboException
03185     *
03186     * @par Python
03187     * getLimitJointMaxPositions(self: pyaubo_sdk.RobotConfig) -> List[float]
03188     *
03189     * @par Lua
03190     * getLimitJointMaxPositions() -> table
03191     *
03192     * @par Lua
03193     * LimitJointMaxPositions = getLimitJointMaxPositions()
03194     *
03195     * @par JSON-RPC
03196     * {"jsonrpc":"2.0","method":"rob1.RobotConfig.getLimitJointMaxPositions","params":[],"id":1}
03197     *
03198     * @par JSON-RPC
03199     *
03200     * {"id":1,"jsonrpc":"2.0","result":[6.2831854820251465,6.2831854820251465,6.2831854820251465,6.2831854820251465,6.2831854820251465,6.2831854820251465,6.2831854820251465,6.2831854820251465]}
03201     * \endchinese
03202     * \english
03203     * Get the joint maximum positions (currently used limit values).
03204     *
03205     * @return Returns the joint maximum positions (currently used limit
03206     * values).
03207     *
03208     * @throws arcs::common_interface::AuboException
03209     *
03210     * @par Python function prototype
03211     * getLimitJointMaxPositions(self: pyaubo_sdk.RobotConfig) -> List[float]
03212     *
03213     * @par Lua function prototype
03214     * getLimitJointMaxPositions() -> table
03215     *
03216     * @par Lua example
03217     * LimitJointMaxPositions = getLimitJointMaxPositions()
03218     *
03219     * @par JSON-RPC request example
03220     * {"jsonrpc":"2.0","method":"rob1.RobotConfig.getLimitJointMaxPositions","params":[],"id":1}
03221     *
03222     * @par JSON-RPC response example
03223     * {"id":1,"jsonrpc":"2.0","result":[6.2831854820251465,6.2831854820251465,6.2831854820251465,6.2831854820251465,6.2831854820251465,6.2831854820251465,6.2831854820251465,6.2831854820251465]}
03224     * \endenglish
03225     */
03226     std::vector<double> getLimitJointMaxPositions();
03227
03228     /**
03229     * @ingroup RobotConfig
03230     * \chinese
03231     *
03232     *
03233     * @return
03234     *
03235     * @throws arcs::common_interface::AuboException
03236     *
03237     * @par Python
03238     * getLimitJointMinPositions(self: pyaubo_sdk.RobotConfig) -> List[float]
03239     *
03240     * @par Lua
03241     * getLimitJointMinPositions() -> table
03242     *
03243     * @par Lua
03244     * LimitJointMinPositions = getLimitJointMinPositions()
03245     *
03246     * @par JSON-RPC
03247     * {"jsonrpc":"2.0","method":"rob1.RobotConfig.getLimitJointMinPositions","params":[],"id":1}
03248     *
03249     * @par JSON-RPC
03250     * {"id":1,"jsonrpc":"2.0","result":[-6.2831854820251465,-6.2831854820251465,-6.2831854820251465,-6.2831854820251465,-6.2831854820251465,-6.2831854820251465,-6.2831854820251465,-6.2831854820251465]}
03251     * \endchinese
03252     * \english
03253     * Get the joint minimum positions (currently used limit values).

```



```

03253     *
03254     * @return Returns the joint minimum positions (currently used limit
03255     * values).
03256     *
03257     * @throws arcs::common_interface::AuboException
03258     *
03259     * @par Python function prototype
03260     * getLimitJointMinPositions(self: pyaubo_sdk.RobotConfig) -> List[float]
03261     *
03262     * @par Lua function prototype
03263     * getLimitJointMinPositions() -> table
03264     *
03265     * @par Lua example
03266     * LimitJointMinPositions = getLimitJointMinPositions()
03267     *
03268     * @par JSON-RPC request example
03269     * {"jsonrpc": "2.0", "method": "rob1.RobotConfig.getLimitJointMinPositions", "params": [], "id": 1}
03270     *
03271     * @par JSON-RPC response example
03272     * {"id": 1, "jsonrpc": "2.0", "result": [-6.2831854820251465, -6.2831854820251465, -6.2831854820251465, -
6.2831854820251465, -6.2831854820251465, -6.2831854820251465]}
03273     * \endenglish
03274     */
03275     std::vector<double> getLimitJointMinPositions();
03276
03277     /**
03278     * @ingroup RobotConfig
03279     * \chinese
03280     *
03281     *
03282     * @return
03283     *
03284     * @throws arcs::common_interface::AuboException
03285     *
03286     * @par Python
03287     * getLimitJointMaxSpeeds(self: pyaubo_sdk.RobotConfig) -> List[float]
03288     *
03289     * @par Lua
03290     * getLimitJointMaxSpeeds() -> table
03291     *
03292     * @par Lua
03293     * LimitJointMaxSpeeds = getLimitJointMaxSpeeds()
03294     *
03295     * @par JSON-RPC
03296     * {"jsonrpc": "2.0", "method": "rob1.RobotConfig.getLimitJointMaxSpeeds", "params": [], "id": 1}
03297     *
03298     * @par JSON-RPC
03299     * {"id": 1, "jsonrpc": "2.0", "result": [3.8920841217041016, 3.8920841217041016, 3.8920841217041016, 3.1066861152648926, 3.1066861152648926, 3.1066861152648926]}
03300     * \endchinese
03301     * \english
03302     * Get the joint maximum speeds (currently used limit values).
03303     *
03304     * @return Returns the joint maximum speeds (currently used limit values).
03305     *
03306     * @throws arcs::common_interface::AuboException
03307     *
03308     * @par Python function prototype
03309     * getLimitJointMaxSpeeds(self: pyaubo_sdk.RobotConfig) -> List[float]
03310     *
03311     * @par Lua function prototype
03312     * getLimitJointMaxSpeeds() -> table
03313     *
03314     * @par Lua example
03315     * LimitJointMaxSpeeds = getLimitJointMaxSpeeds()
03316     *
03317     * @par JSON-RPC request example
03318     * {"jsonrpc": "2.0", "method": "rob1.RobotConfig.getLimitJointMaxSpeeds", "params": [], "id": 1}
03319     *
03320     * @par JSON-RPC response example
03321     * {"id": 1, "jsonrpc": "2.0", "result": [3.8920841217041016, 3.8920841217041016, 3.8920841217041016, 3.1066861152648926, 3.1066861152648926, 3.1066861152648926]}
03322     * \endenglish
03323     */
03324     std::vector<double> getLimitJointMaxSpeeds();
03325
03326     /**
03327     * @ingroup RobotConfig
03328     * \chinese
03329     *
03330     *
03331     * @return
03332     *
03333     * @throws arcs::common_interface::AuboException
03334     *
03335     * @par Python
03336     * getLimitJointMaxAccelerations(self: pyaubo_sdk.RobotConfig) ->

```

```

03337 * List[float]
03338 *
03339 * @par Lua
03340 * getLimitJointMaxAccelerations() -> table
03341 *
03342 * @par Lua
03343 * LimitJointMaxAccelerations = getLimitJointMaxAccelerations()
03344 *
03345 * @par JSON-RPC
03346 * {"jsonrpc":"2.0","method":"rob1.RobotConfig.getLimitJointMaxAccelerations","params":[],"id":1}
03347 *
03348 * @par JSON-RPC
03349 *
03350 {"id":1,"jsonrpc":"2.0","result":[31.104877758314785,31.104877758314785,31.104877758314785,20.73625684294463,20.73625684294463,20.73625684294463]}
03350 * \endchinese
03351 * \english
03352 * Get the joint maximum accelerations (currently used limit values).
03353 *
03354 * @return Returns the joint maximum accelerations (currently used limit
03355 * values).
03356 *
03357 * @throws arcs::common_interface::AuboException
03358 *
03359 * @par Python function prototype
03360 * getLimitJointMaxAccelerations(self: pyaubo_sdk.RobotConfig) ->
03361 * List[float]
03362 *
03363 * @par Lua function prototype
03364 * getLimitJointMaxAccelerations() -> table
03365 *
03366 * @par Lua example
03367 * LimitJointMaxAccelerations = getLimitJointMaxAccelerations()
03368 *
03369 * @par JSON-RPC request example
03370 * {"jsonrpc":"2.0","method":"rob1.RobotConfig.getLimitJointMaxAccelerations","params":[],"id":1}
03371 *
03372 * @par JSON-RPC response example
03373 *
03374 {"id":1,"jsonrpc":"2.0","result":[31.104877758314785,31.104877758314785,31.104877758314785,20.73625684294463,20.73625684294463,20.73625684294463]}
03374 * \endenglish
03375 */
03376 std::vector<double> getLimitJointMaxAccelerations();
03377
03378 /**
03379 * @ingroup RobotConfig
03380 * \chinese
03381 * TCP
03382 *
03383 * @return TCP
03384 *
03385 * @throws arcs::common_interface::AuboException
03386 *
03387 * @par Python
03388 * getLimitTcpMaxSpeed(self: pyaubo_sdk.RobotConfig) -> List[float]
03389 *
03390 * @par Lua
03391 * getLimitTcpMaxSpeed() -> number
03392 *
03393 * @par Lua
03394 * LimitTcpMaxSpeed = getLimitTcpMaxSpeed()
03395 *
03396 * @par JSON-RPC
03397 * {"jsonrpc":"2.0","method":"rob1.RobotConfig.getLimitTcpMaxSpeed","params":[],"id":1}
03398 *
03399 * @par JSON-RPC
03400 * {"id":1,"jsonrpc":"2.0","result":2.0}
03401 * \endchinese
03402 * \english
03403 * Get the TCP maximum speed (currently used limit value).
03404 *
03405 * @return Returns the TCP maximum speed (currently used limit value).
03406 *
03407 * @throws arcs::common_interface::AuboException
03408 *
03409 * @par Python function prototype
03410 * getLimitTcpMaxSpeed(self: pyaubo_sdk.RobotConfig) -> List[float]
03411 *
03412 * @par Lua function prototype
03413 * getLimitTcpMaxSpeed() -> number
03414 *
03415 * @par Lua example
03416 * LimitTcpMaxSpeed = getLimitTcpMaxSpeed()
03417 *
03418 * @par JSON-RPC request example
03419 * {"jsonrpc":"2.0","method":"rob1.RobotConfig.getLimitTcpMaxSpeed","params":[],"id":1}
03420 *
03421 * @par JSON-RPC response example

```

```

03422     * {"id":1,"jsonrpc":"2.0","result":2.0}
03423     * \endenglish
03424     */
03425 double getLimitTcpMaxSpeed();
03426
03427 /**
03428  * @ingroup RobotConfig
03429  * \chinese
03430  *
03431  *
03432  * @return
03433  *
03434  * @par JSON-RPC
03435  * {"jsonrpc":"2.0","method":"rob1.RobotConfig.getSafeguardStopType","params":[],"id":1}
03436  *
03437  * @par JSON-RPC
03438  * {"id":1,"jsonrpc":"2.0","result":"None"}
03439  * \endchinese
03440  * \english
03441  * Get the current safeguard stop type.
03442  *
03443  * @return Returns the current safeguard stop type.
03444  *
03445  * @par JSON-RPC request example
03446  * {"jsonrpc":"2.0","method":"rob1.RobotConfig.getSafeguardStopType","params":[],"id":1}
03447  *
03448  * @par JSON-RPC response example
03449  * {"id":1,"jsonrpc":"2.0","result":"None"}
03450  * \endenglish
03451  */
03452 SafeguardedStopType getSafeguardStopType();
03453
03454 /**
03455  * @ingroup RobotConfig
03456  * \chinese
03457  *
03458  *
03459  * @return
03460  *
03461  *      :
03462  *      IO      - 1«0
03463  *      IO      - 1«1
03464  *      SI      - 1«2
03465  *              - 1«3
03466  *              - 1«4
03467  *
03468  * @par JSON-RPC
03469  * {"jsonrpc":"2.0","method":"rob1.RobotConfig.getSafeguardStopSource","params":[],"id":1}
03470  *
03471  * @par JSON-RPC
03472  * {"id":1,"jsonrpc":"2.0","result":0}
03473  * \endchinese
03474  * \english
03475  * Get the complete safeguard stop source as a bitmask.
03476  *
03477  * @return Returns all safeguard stop sources.
03478  *
03479  * Reasons for safeguard stop:
03480  * Safeguard stop triggered by configurable safety IO in manual mode - 1«0
03481  * Safeguard stop triggered by configurable safety IO in automatic mode -
03482  * 1«1 Safeguard stop triggered by control box SI input - 1«2 Safeguard
03483  * stop triggered by teach pendant three-position switch - 1«3 Safeguard
03484  * stop triggered by switching from auto to manual - 1«4
03485  *
03486  * @par JSON-RPC request example
03487  * {"jsonrpc":"2.0","method":"rob1.RobotConfig.getSafeguardStopSource","params":[],"id":1}
03488  *
03489  * @par JSON-RPC response example
03490  * {"id":1,"jsonrpc":"2.0","result":0}
03491  * \endenglish
03492  */
03493 int getSafeguardStopSource();
03494
03495 /**
03496  * @ingroup RobotConfig
03497  * \chinese
03498  *
03499  *
03500  * @return
03501  *
03502  *      :
03503  *      - 1«0
03504  *      - 1«1
03505  *      - 1«2
03506  *      IO      - 1«3
03507  *
03508  * @par JSON-RPC

```

```

03509 * {"jsonrpc":"2.0","method":"rob1.RobotConfig.getRobotEmergencyStopSource","params":[],"id":1}
03510 *
03511 * @par JSON-RPC
03512 * {"id":1,"jsonrpc":"2.0","result":0}
03513 * \endchinese
03514 * \english
03515 * Get the complete robot emergency stop source as a bitmask.
03516 *
03517 * @return Returns all robot emergency stop sources.
03518 *
03519 * Reasons for emergency stop:
03520 * Emergency stop triggered by control box button - 1«0
03521 * Emergency stop triggered by teach pendant button - 1«1
03522 * Emergency stop triggered by handle button - 1«2
03523 * Emergency stop triggered by fixed IO - 1«3
03524 *
03525 * @par JSON-RPC request example
03526 * {"jsonrpc":"2.0","method":"rob1.RobotConfig.getRobotEmergencyStopSource","params":[],"id":1}
03527 *
03528 * @par JSON-RPC response example
03529 * {"id":1,"jsonrpc":"2.0","result":0}
03530 * \endenglish
03531 */
03532 int getRobotEmergencyStopSource();
03533
03534 /**
03535 * @ingroup RobotConfig
03536 * \chinese
03537 *
03538 *
03539 * @return
03540 *
03541 * @throws arcs::common_interface::AuboException
03542 *
03543 * @par Python
03544 * getSelectedTcpForceSensorName(self: pyaubo_sdk.RobotConfig) -> str
03545 *
03546 * @par Lua
03547 * getSelectedTcpForceSensorName() -> str
03548 *
03549 * @par Lua
03550 * TcpForceSensorName = getSelectedTcpForceSensorName()
03551 *
03552 * @par JSON-RPC
03553 * {"jsonrpc":"2.0","method":"rob1.RobotConfig.getSelectedTcpForceSensorName","params":[],"id":1}
03554 * \endchinese
03555 * \english
03556 * Get the name of the selected TCP force sensor.
03557 *
03558 * @return The name of the currently selected TCP force sensor.
03559 *
03560 * @throws arcs::common_interface::AuboException
03561 *
03562 * @par Python function prototype
03563 * getSelectedTcpForceSensorName(self: pyaubo_sdk.RobotConfig) -> str
03564 *
03565 * @par Lua function prototype
03566 * getSelectedTcpForceSensorName() -> str
03567 *
03568 * @par Lua example
03569 * TcpForceSensorName = getSelectedTcpForceSensorName()
03570 *
03571 * @par JSON-RPC request example
03572 * {"jsonrpc":"2.0","method":"rob1.RobotConfig.getSelectedTcpForceSensorName","params":[],"id":1}
03573 * \endenglish
03574 */
03575 std::string getSelectedTcpForceSensorName();
03576
03577 /**
03578 * @ingroup RobotConfig
03579 * \chinese
03580 *
03581 *
03582 * @param m : kg
03583 * @param cog : m, (CoGx, CoGy, CoGz)
03584 * @param inertia : kg*m^2, (Ixx, Iyy, Izz, Ixy,
03585 * Ixz, Iyz) (Ixx, Ixy, Ixz, Iyx, Iyy, Iyz, Izx, Izy, Izz)
03586 * @return 0;
03587 *
03588 * @throws arcs::common_interface::AuboException
03589 *
03590 * @par Python
03591 * attachWeldingGun(self: pyaubo_sdk.RobotConfig, arg0: float, arg1:
03592 * List[float], arg2: List[float]) -> int
03593 *
03594 * @par Lua
03595 * attachWeldingGun(m: number, cog: table, inertia: table) -> nil

```

```

03596 *
03597 * @par Lua
03598 * attachWeldingGun(3, {0,0,0}, {0,0,0,0,0,0,0,0})
03599 *
03600 * @par JSON-RPC
03601 * {"jsonrpc":"2.0","method":"rob1.RobotConfig.attachWeldingGun","params":[3,[0,0,0],[0,0,0,0,0,0,0,0]],"id":1}
03602 *
03603 * @par JSON-RPC
03604 * {"id":1,"jsonrpc":"2.0","result":0}
03605 * \endchinese
03606 */
03607 int attachWeldingGun(double m, const std::vector<double> &cog,
03608                     const std::vector<double> &inertia);
03609 /**
03610 * @ingroup RobotConfig
03611 * \chinese
03612 *
03613 *
03614 * @param threshold
03615 * @return 0;
03616 *
03617 * @throws arcs::common_interface::AuboException
03618 *
03619 * @par Python
03620 * setCollisionThreshold(self: pyaubo_sdk.RobotConfig, arg0: List[float])
03621 * -> int
03622 *
03623 * @par Lua
03624 * setCollisionThreshold(threshold: table) -> nil
03625 *
03626 * @par Lua
03627 * setCollisionThreshold({99.0,
03628 * 114.0,
03629 * 103.0, 56.0, 51.0, 60.0,54.6111111, 66.22222222, 59.66666667, 28.94444444,
03630 * 27.05555556, 30.11111111,19.1, 28.0, 25.0, 7.3, 7.9, 6.2})
03631 *
03632 * @par JSON-RPC
03633 * {"jsonrpc":"2.0","method":"rob1.RobotConfig.setCollisionThreshold","params":[[99.0,
03634 * 114.0,
03635 * 103.0, 56.0, 51.0, 60.0,54.6111111, 66.22222222, 59.66666667, 28.94444444,
03636 * 27.05555556, 30.11111111,19.1, 28.0, 25.0, 7.3, 7.9, 6.2]],"id":1}
03637 *
03638 * @par JSON-RPC
03639 * {"id":1,"jsonrpc":"2.0","result":0}
03640 * \endchinese
03641 * \english
03642 * Enable optimized collision force.
03643 *
03644 * @param threshold
03645 * @return Returns 0 on success; error code on failure.
03646 *
03647 * @par JSON-RPC request example
03648 * {"jsonrpc":"2.0","method":"rob1.RobotConfig.setCollisionThreshold","params":[[99.0,
03649 * 114.0,
03650 * 103.0, 56.0, 51.0, 60.0,54.6111111, 66.22222222, 59.66666667, 28.94444444,
03651 * 27.05555556, 30.11111111,19.1, 28.0, 25.0, 7.3, 7.9, 6.2]],"id":1}
03652 *
03653 * @par JSON-RPC response example
03654 * {"id":1,"jsonrpc":"2.0","result":"None"}
03655 * \endenglish
03656 */
03657 int setCollisionThreshold(const std::vector<double> &threshold);
03658
03659 /**
03660 * @ingroup RobotConfig
03661 * \chinese
03662 *
03663 *
03664 * @return
03665 *
03666 * @throws arcs::common_interface::AuboException
03667 *
03668 * @par Python
03669 * getCollisionThreshold(self: pyaubo_sdk.RobotConfig) -> List[float]
03670 *
03671 * @par Lua
03672 * getCollisionThreshold() -> table
03673 *
03674 * @par Lua
03675 * CollisionThreshold = getCollisionThreshold()
03676 *
03677 * @par JSON-RPC
03678 * {"jsonrpc":"2.0","method":"rob1.MotionControl.getCollisionThreshold","params":[99.0,
03679 * 114.0, 103.0, 56.0, 51.0, 60.0],"id":1}
03680 *
03681 * @par JSON-RPC
03682 * {"id":1,"jsonrpc":"2.0","result":1}

```

```

03683 * \endchinese
03684 * \english
03685 * Get collision threshold.
03686 *
03687 * @return collision threshold.
03688 *
03689 * @throws arcs::common_interface::AuboException
03690 *
03691 * @par Python function prototype
03692 * getCollisionThreshold(self: pyaubo_sdk.RobotConfig) -> List[float]
03693 *
03694 * @par Lua function prototype
03695 * getCollisionThreshold() -> table
03696 *
03697 * @par Lua example
03698 * CollisionThreshold = getCollisionThreshold()
03699 *
03700 * @par JSON-RPC request example
03701 * {"jsonrpc":"2.0","method":"rob1.RobotConfig.getCollisionThreshold","params":[[99.0,
03702 * 114.0,
03703 * 103.0, 56.0, 51.0, 60.0,54.61111111, 66.22222222, 59.66666667, 28.94444444,
03704 * 27.05555556, 30.11111111,19.1, 28.0, 25.0, 7.3, 7.9, 6.2]],"id":1}
03705 *
03706 * @par JSON-RPC response example
03707 * {"id":1,"jsonrpc":"2.0","result":None}
03708 * \endenglish
03709 */
03710 std::vector<double> getCollisionThreshold();
03711
03712 int enableAxisGroup(const std::string &group_name);
03713 int disableAxisGroup();
03714
03715 /**
03716 * @ingroup RobotConfig
03717 * \chinese
03718 *
03719 *
03720 * @param enable
03721 * @return 0;
03722 *
03723 * @throws arcs::common_interface::AuboException
03724 *
03725 * @par Python
03726 * enableEndCollisionCheck(self: pyaubo_sdk.RobotConfig, arg0: bool)
03727 * -> int
03728 *
03729 * @par Lua
03730 * enableEndCollisionCheck(threshold: table) -> nil
03731 *
03732 * @par Lua
03733 * enableEndCollisionCheck(true)
03734 *
03735 * @par JSON-RPC
03736 * {"jsonrpc":"2.0","method":"rob1.RobotConfig.enableEndCollisionCheck","params":[true],"id":1}
03737 *
03738 * @par JSON-RPC
03739 * {"id":1,"jsonrpc":"2.0","result":0}
03740 * \endchinese
03741 * \english
03742 * Set end collision threshold.
03743 *
03744 * @param threshold
03745 * @return Returns 0 on success; error code on failure.
03746 *
03747 * @par JSON-RPC request example
03748 * {"jsonrpc":"2.0","method":"rob1.RobotConfig.enableEndCollisionCheck","params":[true],"id":1}
03749 *
03750 * @par JSON-RPC response example
03751 * {"id":1,"jsonrpc":"2.0","result":None}
03752 * \endenglish
03753 */
03754 int enableEndCollisionCheck(bool enable);
03755
03756 /**
03757 * @ingroup RobotConfig
03758 * \chinese
03759 *
03760 *
03761 * @return
03762 *
03763 * @throws arcs::common_interface::AuboException
03764 *
03765 * @par Python
03766 * isEndCollisionCheckEnabled(self: pyaubo_sdk.RobotConfig) -> List[float]
03767 *
03768 * @par Lua
03769 * isEndCollisionCheckEnabled() -> table

```

```

03770  *
03771  * @par Lua
03772  * IsEndCollisionCheckEnabled = isEndCollisionCheckEnabled()
03773  *
03774  * @par JSON-RPC
03775  * {"jsonrpc":"2.0","method":"rob1.MotionControl.isEndCollisionCheckEnabled","params":[],"id":1}
03776  *
03777  * @par JSON-RPC
03778  * {"id":1,"jsonrpc":"2.0","result":true}
03779  * \endchinese
03780  * \english
03781  * Get collision check is enabled.
03782  *
03783  * @return end collision threshold.
03784  *
03785  * @throws arcs::common_interface::AuboException
03786  *
03787  * @par Python function prototype
03788  * isEndCollisionCheckEnabled(self: pyaubo_sdk.RobotConfig) -> bool
03789  *
03790  * @par Lua function prototype
03791  * isEndCollisionCheckEnabled() -> bool
03792  *
03793  * @par Lua example
03794  * IsEndCollisionCheckEnabled = isEndCollisionCheckEnabled()
03795  *
03796  * @par JSON-RPC request example
03797  * {"jsonrpc":"2.0","method":"rob1.RobotConfig.isEndCollisionCheckEnabled","params":[],"id":1}
03798  *
03799  * @par JSON-RPC response example
03800  * {"id":1,"jsonrpc":"2.0","result":true}
03801  * \endenglish
03802  */
03803 bool isEndCollisionCheckEnabled();
03804
03805 protected:
03806 void *d_;
03807 };
03808 using RobotConfigPtr = std::shared_ptr<RobotConfig>;
03809
03810 } // namespace common_interface
03811 } // namespace arcs
03812 #endif // AUBO_SDK_ROBOT_CONFIG_H

```

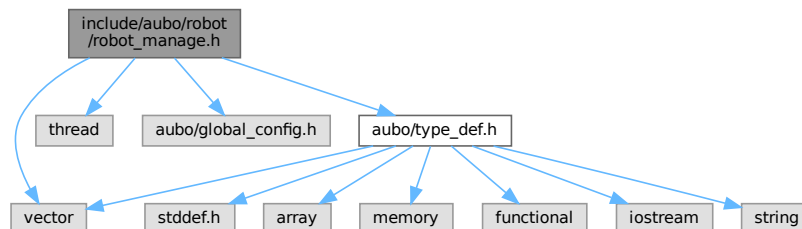
12.30 include/aubo/robot/robot_manage.h File Reference

```

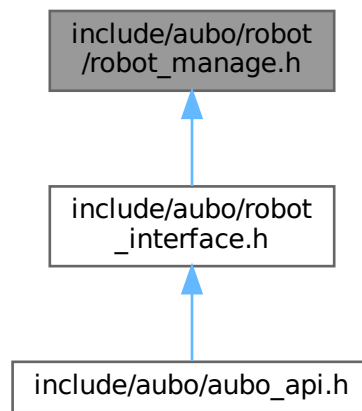
#include <vector>
#include <thread>
#include <aubo/global_config.h>
#include <aubo/type_def.h>

```

Include dependency graph for robot_manage.h:



This graph shows which files directly or indirectly include this file:



Classes

- class [arcs::common_interface::RobotManage](#)

Namespaces

- namespace [arcs](#)
- namespace [arcs::common_interface](#)

Typedefs

- using [arcs::common_interface::RobotManagePtr](#) = std::shared_ptr<[RobotManage](#)>

12.30.1 Detailed Description

Definition in file [robot_manage.h](#).

12.31 robot_manage.h

[Go to the documentation of this file.](#)

```

00001 /** @file robot_manage.h
00002  * @brief
00003  */
00004 #ifndef AUBO_SDK_ROBOT_CONTROL_INTERFACE_H
00005 #define AUBO_SDK_ROBOT_CONTROL_INTERFACE_H
00006
00007 #include <vector>
00008 #include <thread>
00009
00010 #include <aubo/global_config.h>
00011 #include <aubo/type_def.h>
00012
00013 namespace arcs {
00014 namespace common_interface {
00015
00016 /**
00017  * @defgroup RobotManage RobotManage ( )
00018  * @ingroup RobotInterface
00019  * RobotManage
00020  */
00021 class ARCS_ABI_EXPORT RobotManage
00022 {
00023 public:
00024     RobotManage();
00025     virtual ~RobotManage();
00026
00027     /**
00028      * @ingroup RobotManage
00029      * \chinese
00030      *
00031      *
00032      * @return 0;
00033      * AUBO_BAD_STATE
00034      *
00035      * @throws arcs::common_interface::AuboException
00036      *
00037      * @par Python
00038      * poweron(self: pyaubo_sdk.RobotManage) -> int
00039      *
00040      * @par Lua
00041      * poweron() -> number
00042      *
00043      * @par Lua
00044      * num = poweron()
00045      *
00046      * @par JSON-RPC
00047      * {"jsonrpc":"2.0","method":"rob1.RobotManage.poweron","params":[],"id":1}
00048      *
00049      * @par JSON-RPC
00050      * {"id":1,"jsonrpc":"2.0","result":0}
00051      *
00052      * @par C++
00053      * @code
00054      * auto rpc_cli = std::make_shared<RpcClient>();
00055      * auto robot_name = rpc_cli->getRobotNames().front();
00056      * rpc_cli->getRobotInterface(robot_name)->getRobotManage()->poweron();
00057      * @endcode
00058      *
00059      * \endchinese
00060      *
00061      * \english
00062      * Initiate robot power-on request
00063      *
00064      * @return Returns 0 on success; error code on failure
00065      * AUBO_BAD_STATE
00066      *
00067      * @throws arcs::common_interface::AuboException
00068      *
00069      * @par Python function prototype
00070      * poweron(self: pyaubo_sdk.RobotManage) -> int
00071      *
00072      * @par Lua function prototype
00073      * poweron() -> number
00074      *
00075      * @par Lua example
00076      * num = poweron()
00077      *
00078      * @par JSON-RPC request example
00079      * {"jsonrpc":"2.0","method":"rob1.RobotManage.poweron","params":[],"id":1}
00080      *
00081      * @par JSON-RPC response example
00082      * {"id":1,"jsonrpc":"2.0","result":0}

```

```

00083  *
00084  * @par C++ example
00085  * @code
00086  * auto rpc_cli = std::make_shared<RpcClient>();
00087  * auto robot_name = rpc_cli->getRobotNames().front();
00088  * rpc_cli->getRobotInterface(robot_name)->getRobotManage()->poweron();
00089  * @endcode
00090  * \endenglish
00091  */
00092  int poweron();
00093
00094  /**
00095  * @ingroup RobotManage
00096  * \chinese
00097  *
00098  *
00099  * @return 0;
00100  * AUBO_BAD_STATE
00101  *
00102  * @throws arcs::common_interface::AuboException
00103  *
00104  * @par Python
00105  * startup(self: pyaubo_sdk.RobotManage) -> int
00106  *
00107  * @par Lua
00108  * startup() -> number
00109  *
00110  * @par Lua
00111  * num = startup()
00112  *
00113  * @par JSON-RPC
00114  * {"jsonrpc":"2.0","method":"rob1.RobotManage.startup","params":[],"id":1}
00115  *
00116  * @par JSON-RPC
00117  * {"id":1,"jsonrpc":"2.0","result":0}
00118  *
00119  * @par C++
00120  * @code
00121  * auto rpc_cli = std::make_shared<RpcClient>();
00122  * auto robot_name = rpc_cli->getRobotNames().front();
00123  * rpc_cli->getRobotInterface(robot_name)->getRobotManage()->startup();
00124  * @endcode
00125  * \endchinese
00126  *
00127  * \english
00128  * Initiate robot startup request
00129  *
00130  * @return Returns 0 on success; error code on failure
00131  * AUBO_BAD_STATE
00132  *
00133  * @throws arcs::common_interface::AuboException
00134  *
00135  * @par Python function prototype
00136  * startup(self: pyaubo_sdk.RobotManage) -> int
00137  *
00138  * @par Lua function prototype
00139  * startup() -> number
00140  *
00141  * @par Lua example
00142  * num = startup()
00143  *
00144  * @par JSON-RPC request example
00145  * {"jsonrpc":"2.0","method":"rob1.RobotManage.startup","params":[],"id":1}
00146  *
00147  * @par JSON-RPC response example
00148  * {"id":1,"jsonrpc":"2.0","result":0}
00149  *
00150  * @par C++ example
00151  * @code
00152  * auto rpc_cli = std::make_shared<RpcClient>();
00153  * auto robot_name = rpc_cli->getRobotNames().front();
00154  * rpc_cli->getRobotInterface(robot_name)->getRobotManage()->startup();
00155  * @endcode
00156  * \endenglish
00157  */
00158  int startup();
00159
00160  /**
00161  * @ingroup RobotManage
00162  * \chinese
00163  *
00164  *
00165  * @return 0;
00166  * AUBO_BAD_STATE
00167  *
00168  * @throws arcs::common_interface::AuboException
00169  *

```

```

00170 * @par Python
00171 * releaseRobotBrake(self: pyaubo_sdk.RobotManage) -> int
00172 *
00173 * @par Lua
00174 * releaseRobotBrake() -> number
00175 *
00176 * @par Lua
00177 * num = releaseRobotBrake()
00178 *
00179 * @par JSON-RPC
00180 * {"jsonrpc": "2.0", "method": "rob1.RobotManage.releaseRobotBrake", "params": [], "id": 1}
00181 *
00182 * @par JSON-RPC
00183 * {"id": 1, "jsonrpc": "2.0", "result": 0}
00184 *
00185 * @par C++
00186 * @code
00187 * auto rpc_cli = std::make_shared<RpcClient>();
00188 * auto robot_name = rpc_cli->getRobotNames().front();
00189 * rpc_cli->getRobotInterface(robot_name)->getRobotManage()->releaseRobotBrake();
00190 * @endcode
00191 * \endchinese
00192 *
00193 * \english
00194 * Initiate robot brake release request
00195 *
00196 * @return Returns 0 on success; error code on failure
00197 * AUBO_BAD_STATE
00198 *
00199 * @throws arcs::common_interface::AuboException
00200 *
00201 * @par Python function prototype
00202 * releaseRobotBrake(self: pyaubo_sdk.RobotManage) -> int
00203 *
00204 * @par Lua function prototype
00205 * releaseRobotBrake() -> number
00206 *
00207 * @par Lua example
00208 * num = releaseRobotBrake()
00209 *
00210 * @par JSON-RPC request example
00211 * {"jsonrpc": "2.0", "method": "rob1.RobotManage.releaseRobotBrake", "params": [], "id": 1}
00212 *
00213 * @par JSON-RPC response example
00214 * {"id": 1, "jsonrpc": "2.0", "result": 0}
00215 *
00216 * @par C++ example
00217 * @code
00218 * auto rpc_cli = std::make_shared<RpcClient>();
00219 * auto robot_name = rpc_cli->getRobotNames().front();
00220 * rpc_cli->getRobotInterface(robot_name)->getRobotManage()->releaseRobotBrake();
00221 * @endcode
00222 * \endenglish
00223 */
00224 int releaseRobotBrake();
00225
00226 /**
00227 * @ingroup RobotManage
00228 * \chinese
00229 *
00230 *
00231 * @return 0;
00232 * AUBO_BAD_STATE
00233 *
00234 * @throws arcs::common_interface::AuboException
00235 *
00236 * @par Python
00237 * lockRobotBrake(self: pyaubo_sdk.RobotManage) -> int
00238 *
00239 * @par Lua
00240 * lockRobotBrake() -> number
00241 *
00242 * @par Lua
00243 * num = lockRobotBrake()
00244 *
00245 * @par JSON-RPC
00246 * {"jsonrpc": "2.0", "method": "rob1.RobotManage.lockRobotBrake", "params": [], "id": 1}
00247 *
00248 * @par JSON-RPC
00249 * {"id": 1, "jsonrpc": "2.0", "result": 0}
00250 *
00251 * @par C++
00252 * @code
00253 * auto rpc_cli = std::make_shared<RpcClient>();
00254 * auto robot_name = rpc_cli->getRobotNames().front();
00255 * rpc_cli->getRobotInterface(robot_name)->getRobotManage()->lockRobotBrake();
00256 * @endcode

```

```

00257 * \endchinese
00258 *
00259 * \english
00260 * Initiate robot brake lock request
00261 *
00262 * @return Returns 0 on success; error code on failure
00263 * AUBO_BAD_STATE
00264 *
00265 * @throws arcs::common_interface::AuboException
00266 *
00267 * @par Python function prototype
00268 * lockRobotBrake(self: pyaubo_sdk.RobotManage) -> int
00269 *
00270 * @par Lua function prototype
00271 * lockRobotBrake() -> number
00272 *
00273 * @par Lua example
00274 * num = lockRobotBrake()
00275 *
00276 * @par JSON-RPC request example
00277 * {"jsonrpc": "2.0", "method": "rob1.RobotManage.lockRobotBrake", "params": [], "id": 1}
00278 *
00279 * @par JSON-RPC response example
00280 * {"id": 1, "jsonrpc": "2.0", "result": 0}
00281 *
00282 * @par C++ example
00283 * @code
00284 * auto rpc_cli = std::make_shared<RpcClient>();
00285 * auto robot_name = rpc_cli->getRobotNames().front();
00286 * rpc_cli->getRobotInterface(robot_name)->getRobotManage()->lockRobotBrake();
00287 * @endcode
00288 * \endenglish
00289 */
00290 int lockRobotBrake();
00291
00292 /**
00293 * @ingroup RobotManage
00294 * \chinese
00295 *
00296 *
00297 * @return 0;
00298 * AUBO_BAD_STATE
00299 *
00300 * @throws arcs::common_interface::AuboException
00301 *
00302 * @par Python
00303 * poweroff(self: pyaubo_sdk.RobotManage) -> int
00304 *
00305 * @par Lua
00306 * poweroff() -> number
00307 *
00308 * @par Lua
00309 * num = poweroff()
00310 *
00311 * @par JSON-RPC
00312 * {"jsonrpc": "2.0", "method": "rob1.RobotManage.poweroff", "params": [], "id": 1}
00313 *
00314 * @par JSON-RPC
00315 * {"id": 1, "jsonrpc": "2.0", "result": 0}
00316 *
00317 * @par C++
00318 * @code
00319 * auto rpc_cli = std::make_shared<RpcClient>();
00320 * auto robot_name = rpc_cli->getRobotNames().front();
00321 * rpc_cli->getRobotInterface(robot_name)->getRobotManage()->poweroff();
00322 * @endcode
00323 * \endchinese
00324 *
00325 * \english
00326 * Initiate robot power-off request
00327 *
00328 * @return Returns 0 on success; error code on failure
00329 * AUBO_BAD_STATE
00330 *
00331 * @throws arcs::common_interface::AuboException
00332 *
00333 * @par Python function prototype
00334 * poweroff(self: pyaubo_sdk.RobotManage) -> int
00335 *
00336 * @par Lua function prototype
00337 * poweroff() -> number
00338 *
00339 * @par Lua example
00340 * num = poweroff()
00341 *
00342 * @par JSON-RPC request example
00343 * {"jsonrpc": "2.0", "method": "rob1.RobotManage.poweroff", "params": [], "id": 1}

```

```

00344 *
00345 * @par JSON-RPC response example
00346 * {"id":1,"jsonrpc":"2.0","result":0}
00347 *
00348 * @par C++ example
00349 * @code
00350 * auto rpc_cli = std::make_shared<RpcClient>();
00351 * auto robot_name = rpc_cli->getRobotNames().front();
00352 * rpc_cli->getRobotInterface(robot_name)->getRobotManage()->poweroff();
00353 * @endcode
00354 * \endenglish
00355 */
00356 int poweroff();
00357
00358 /**
00359 * @ingroup RobotManage
00360 * \chinese
00361 *
00362 *
00363 * @param enable
00364 *
00365 * @return 0;
00366 * AUBO_BUSY
00367 * AUBO_BAD_STATE
00368 * -AUBO_INVL_ARGUMENT
00369 * -AUBO_BAD_STATE
00370 *
00371 * @throws arcs::common_interface::AuboException
00372 *
00373 * @par Python
00374 * backdrive(self: pyaubo_sdk.RobotManage, arg0: bool) -> int
00375 *
00376 * @par Lua
00377 * backdrive(enable: boolean) -> number
00378 *
00379 * @par Lua
00380 * num = backdrive(false)
00381 *
00382 * @par JSON-RPC
00383 * {"jsonrpc":"2.0","method":"rob1.RobotManage.backdrive","params":[false],"id":1}
00384 *
00385 * @par JSON-RPC
00386 * {"id":1,"jsonrpc":"2.0","result":0}
00387 *
00388 * @par C++
00389 * @code
00390 * auto rpc_cli = std::make_shared<RpcClient>();
00391 * auto robot_name = rpc_cli->getRobotNames().front();
00392 * rpc_cli->getRobotInterface(robot_name)->getRobotManage()->backdrive(true);
00393 * @endcode
00394 * \endchinese
00395 *
00396 * \english
00397 * Initiate robot backdrive request
00398 *
00399 * @param enable
00400 *
00401 * @return Returns 0 on success; error code on failure
00402 * AUBO_BUSY
00403 * AUBO_BAD_STATE
00404 * -AUBO_INVL_ARGUMENT
00405 * -AUBO_BAD_STATE
00406 *
00407 * @throws arcs::common_interface::AuboException
00408 *
00409 * @par Python function prototype
00410 * backdrive(self: pyaubo_sdk.RobotManage, arg0: bool) -> int
00411 *
00412 * @par Lua function prototype
00413 * backdrive(enable: boolean) -> number
00414 *
00415 * @par Lua example
00416 * num = backdrive(false)
00417 *
00418 * @par JSON-RPC request example
00419 * {"jsonrpc":"2.0","method":"rob1.RobotManage.backdrive","params":[false],"id":1}
00420 *
00421 * @par JSON-RPC response example
00422 * {"id":1,"jsonrpc":"2.0","result":0}
00423 *
00424 * @par C++ example
00425 * @code
00426 * auto rpc_cli = std::make_shared<RpcClient>();
00427 * auto robot_name = rpc_cli->getRobotNames().front();
00428 * rpc_cli->getRobotInterface(robot_name)->getRobotManage()->backdrive(true);
00429 * @endcode
00430 * \endenglish

```

```

00431     */
00432     int backdrive(bool enable);
00433
00434     /**
00435      * @ingroup RobotManage
00436      * \chinese
00437      *
00438      * 0.31.x handguideMode
00439      * handguideMode({1,1,1,1,1}, {0,0,0,0,0})
00440      *
00441      * @param enable
00442      *
00443      * @return 0;
00444      * AUBO_BUSY
00445      * AUBO_BAD_STATE
00446      * -AUBO_INVL_ARGUMENT
00447      * -AUBO_BAD_STATE
00448      *
00449      * @throws arcs::common_interface::AuboException
00450      *
00451      * @par Python
00452      * freedrive(self: pyaubo_sdk.RobotManage, arg0: bool) -> int
00453      *
00454      * @par Lua
00455      * freedrive(enable: boolean) -> number
00456      *
00457      * @par Lua
00458      * num = freedrive(false)
00459      *
00460      * @par JSON-RPC
00461      * {"jsonrpc":"2.0","method":"rob1.RobotManage.freedrive","params":[true],"id":1}
00462      *
00463      * @par JSON-RPC
00464      * {"id":1,"jsonrpc":"2.0","result":0}
00465      *
00466      * @par C++
00467      * @code
00468      * auto rpc_cli = std::make_shared<RpcClient>();
00469      * auto robot_name = rpc_cli->getRobotNames().front();
00470      * rpc_cli->getRobotInterface(robot_name)->getRobotManage()->freedrive(true);
00471      * @endcode
00472      * \endchinese
00473      *
00474      * \english
00475      * Initiate robot freedrive request
00476      * This interface is deprecated after software version 0.31.x, use
00477      * handguideMode instead: handguideMode({1,1,1,1,1}, {0,0,0,0,0})
00478      *
00479      * @param enable
00480      *
00481      * @return Returns 0 on success; error code on failure
00482      * AUBO_BUSY
00483      * AUBO_BAD_STATE
00484      * -AUBO_INVL_ARGUMENT
00485      * -AUBO_BAD_STATE
00486      *
00487      * @throws arcs::common_interface::AuboException
00488      *
00489      * @par Python function prototype
00490      * freedrive(self: pyaubo_sdk.RobotManage, arg0: bool) -> int
00491      *
00492      * @par Lua function prototype
00493      * freedrive(enable: boolean) -> number
00494      *
00495      * @par Lua example
00496      * num = freedrive(false)
00497      *
00498      * @par JSON-RPC request example
00499      * {"jsonrpc":"2.0","method":"rob1.RobotManage.freedrive","params":[true],"id":1}
00500      *
00501      * @par JSON-RPC response example
00502      * {"id":1,"jsonrpc":"2.0","result":0}
00503      *
00504      * @par C++ example
00505      * @code
00506      * auto rpc_cli = std::make_shared<RpcClient>();
00507      * auto robot_name = rpc_cli->getRobotNames().front();
00508      * rpc_cli->getRobotInterface(robot_name)->getRobotManage()->freedrive(true);
00509      * @endcode
00510      * \endenglish
00511     */
00512     int freedrive(bool enable);
00513
00514     /**
00515      * @ingroup RobotManage
00516      * \chinese
00517      *

```

```

00518 *
00519 * @param freeAxes    0- 1-
00520 * @param feature    0   TCP
00521 *
00522 * @return    0;
00523 * AUBO_BUSY
00524 * AUBO_BAD_STATE
00525 * -AUBO_INVL_ARGUMENT
00526 * -AUBO_BAD_STATE
00527 *
00528 * @throws arcs::common_interface::AuboException
00529 * \endchinese
00530 *
00531 * \english
00532 * Advanced hand-guiding mode
00533 *
00534 * @param freeAxes Axes that can be moved: 0-cannot move, 1-can move
00535 * @param feature If the dimension is 0, it means hand-guiding based on
00536 * the TCP coordinate system
00537 *
00538 * @return Returns 0 on success; error code on failure
00539 * AUBO_BUSY
00540 * AUBO_BAD_STATE
00541 * -AUBO_INVL_ARGUMENT
00542 * -AUBO_BAD_STATE
00543 *
00544 * @throws arcs::common_interface::AuboException
00545 * \endenglish
00546 */
00547 int setHandguideParams(const std::vector<int> &freeAxes,
00548                       const std::vector<double> &feature);
00549
00550 /**
00551 * @ingroup RobotManage
00552 * \chinese
00553 *
00554 * \endchinese
00555 *
00556 * \english
00557 * Get Axes that can be moved
00558 * \endenglish
00559 */
00560 std::vector<int> getHandguideFreeAxes();
00561
00562 /**
00563 * @ingroup RobotManage
00564 * \chinese
00565 *
00566 * \endchinese
00567 *
00568 * \english
00569 * Get the drag reference coordinate system
00570 * \endenglish
00571 */
00572 std::vector<double> getHandguideFeature();
00573
00574 /**
00575 * @ingroup RobotManage
00576 * \chinese
00577 *
00578 *
00579 * @param freeAxes    0- 1-
00580 * @param feature    0   TCP
00581 *
00582 * @return    0;
00583 * AUBO_BUSY
00584 * AUBO_BAD_STATE
00585 * -AUBO_INVL_ARGUMENT
00586 * -AUBO_BAD_STATE
00587 *
00588 * @throws arcs::common_interface::AuboException
00589 * \endchinese
00590 *
00591 * \english
00592 * Advanced hand-guiding mode
00593 *
00594 * @param freeAxes Axes that can be moved: 0-cannot move, 1-can move
00595 * @param feature If the dimension is 0, it means hand-guiding based on the
00596 * TCP coordinate system
00597 *
00598 * @return Returns 0 on success; error code on failure
00599 * AUBO_BUSY
00600 * AUBO_BAD_STATE
00601 * -AUBO_INVL_ARGUMENT
00602 * -AUBO_BAD_STATE
00603 *
00604 * @throws arcs::common_interface::AuboException

```

```

00605     * \endenglish
00606     */
00607 int handguideMode(const std::vector<int> &freeAxes,
00608                  const std::vector<double> &feature);
00609
00610 /**
00611  * @ingroup RobotManage
00612  * \chinese
00613  *
00614  *
00615  * @note 0.31.x
00616  *
00617  * @return 0;
00618  * AUBO_BUSY
00619  * AUBO_BAD_STATE
00620  * -AUBO_BAD_STATE
00621  *
00622  * @throws arcs::common_interface::AuboException
00623  * \endchinese
00624  *
00625  * \english
00626  * Exit hand-guiding mode
00627  *
00628  * @note Requires software version 0.31.x or later
00629  *
00630  * @return Returns 0 on success; error code on failure
00631  * AUBO_BUSY
00632  * AUBO_BAD_STATE
00633  * -AUBO_BAD_STATE
00634  *
00635  * @throws arcs::common_interface::AuboException
00636  * \endenglish
00637  */
00638 int exitHandguideMode();
00639
00640 /**
00641  * @ingroup RobotManage
00642  * \chinese
00643  *
00644  *
00645  * @note
00646  *
00647  * @return
00648  * • 0 - .
00649  * • 1 - .
00650  * • 2 - .
00651  *
00652  * @throws arcs::common_interface::AuboException
00653  * \endchinese
00654  *
00655  * \english
00656  * Get the status of the hand-guiding device (whether it is in a singular
00657  * space)
00658  *
00659  * @note Not implemented yet
00660  *
00661  * @return
00662  * • 0 - Normal operation.
00663  * • 1 - Approaching singular space.
00664  * • 2 - Extremely close to a singularity, large hand-guiding damping will
00665  * occur.
00666  *
00667  * @throws arcs::common_interface::AuboException
00668  * \endenglish
00669  */
00670 int getHandguideStatus();
00671
00672 /**
00673  * @ingroup RobotManage
00674  * \chinese
00675  *
00676  *
00677  * @note
00678  *
00679  * @return
00680  *
00681  * @throws arcs::common_interface::AuboException
00682  * \endchinese
00683  *
00684  * \english
00685  * Get the trigger source of the hand-guiding device
00686  *
00687  * @note Not implemented yet
00688  *
00689  * @return
00690  *
00691  * @throws arcs::common_interface::AuboException

```



```

00692     * \endenglish
00693     */
00694 int getHandguideTrigger();
00695
00696 /**
00697  * @ingroup RobotManage
00698  * \chinese
00699  *
00700  *
00701  * @return true; false
00702  *
00703  * @throws arcs::common_interface::AuboException
00704  *
00705  * @par Lua
00706  * isHandguideEnabled() -> boolean
00707  *
00708  * @par Lua
00709  * Handguide = isHandguideEnabled()
00710  *
00711  * @par JSON-RPC
00712  * {"jsonrpc":"2.0","method":"rob1.RobotManage.isHandguideEnabled","params":[],"id":1}
00713  *
00714  * @par JSON-RPC
00715  * {"id":1,"jsonrpc":"2.0","result":false}
00716  *
00717  * \endchinese
00718  * \english
00719  * Get the hand-guiding enable status
00720  *
00721  * @return Returns true if enabled; false if disabled
00722  *
00723  * @throws arcs::common_interface::AuboException
00724  *
00725  * @par Lua function prototype
00726  * isHandguideEnabled() -> boolean
00727  *
00728  * @par Lua example
00729  * Handguide = isHandguideEnabled()
00730  *
00731  * @par JSON-RPC request example
00732  * {"jsonrpc":"2.0","method":"rob1.RobotManage.isHandguideEnabled","params":[],"id":1}
00733  *
00734  * @par JSON-RPC response example
00735  * {"id":1,"jsonrpc":"2.0","result":false}
00736  *
00737  * \endenglish
00738  */
00739 bool isHandguideEnabled();
00740
00741 /**
00742  * @ingroup RobotManage
00743  * \chinese
00744  * /
00745  *
00746  * @param enable
00747  *
00748  * @return 0
00749  * AUBO_BUSY
00750  * AUBO_BAD_STATE
00751  * -AUBO_INVL_ARGUMENT
00752  * -AUBO_BAD_STATE
00753  *
00754  * @throws arcs::common_interface::AuboException
00755  *
00756  * @par Python
00757  * setSim(self: pyaubo_sdk.RobotManage, arg0: bool) -> int
00758  *
00759  * @par Lua
00760  * setSim(enable: boolean) -> number
00761  *
00762  * @par Lua
00763  * num = setSim(true)
00764  *
00765  * @par JSON-RPC
00766  * {"jsonrpc":"2.0","method":"rob1.RobotManage.setSim","params":[true],"id":1}
00767  *
00768  * @par JSON-RPC
00769  * {"id":1,"jsonrpc":"2.0","result":0}
00770  *
00771  * @par C++
00772  * @code
00773  * auto rpc_cli = std::make_shared<RpcClient>();
00774  * auto robot_name = rpc_cli->getRobotNames().front();
00775  * rpc_cli->getRobotInterface(robot_name)->getRobotManage()->setSim(true);
00776  * @endcode
00777  * \endchinese
00778  *

```

```

00779 * \english
00780 * Initiate robot enter/exit simulation mode request
00781 *
00782 * @param enable
00783 *
00784 * @return Returns 0 on success; error code on failure
00785 * AUBO_BUSY
00786 * AUBO_BAD_STATE
00787 * -AUBO_INVL_ARGUMENT
00788 * -AUBO_BAD_STATE
00789 *
00790 * @throws arcs::common_interface::AuboException
00791 *
00792 * @par Python function prototype
00793 * setSim(self: pyaubo_sdk.RobotManage, arg0: bool) -> int
00794 *
00795 * @par Lua function prototype
00796 * setSim(enable: boolean) -> number
00797 *
00798 * @par Lua example
00799 * num = setSim(true)
00800 *
00801 * @par JSON-RPC request example
00802 * {"jsonrpc": "2.0", "method": "rob1.RobotManage.setSim", "params": [true], "id": 1}
00803 *
00804 * @par JSON-RPC response example
00805 * {"id": 1, "jsonrpc": "2.0", "result": 0}
00806 *
00807 * @par C++ example
00808 * @code
00809 * auto rpc_cli = std::make_shared<RpcClient>();
00810 * auto robot_name = rpc_cli->getRobotNames().front();
00811 * rpc_cli->getRobotInterface(robot_name)->getRobotManage()->setSim(true);
00812 * @endcode
00813 * \endenglish
00814 */
00815 int setSim(bool enable);
00816
00817 /**
00818 * @ingroup RobotManage
00819 * \chinese
00820 *
00821 *
00822 * @param mode
00823 *
00824 * @return 0;
00825 * -AUBO_BAD_STATE
00826 *
00827 * @throws arcs::common_interface::AuboException
00828 *
00829 * @par Python
00830 * setOperationalMode(self: pyaubo_sdk.RobotManage, arg0:
00831 * arcs::common_interface::OperationalModeType) -> int
00832 *
00833 * @par Lua
00834 * setOperationalMode(mode: number) -> number
00835 *
00836 * @par Lua
00837 * num = setOperationalMode("Manual")
00838 *
00839 * @par JSON-RPC
00840 * {"jsonrpc": "2.0", "method": "rob1.RobotManage.setOperationalMode", "params": ["Manual"], "id": 1}
00841 *
00842 * @par JSON-RPC
00843 * {"id": 1, "jsonrpc": "2.0", "result": 0}
00844 *
00845 * @par C++
00846 * @code
00847 * auto rpc_cli = std::make_shared<RpcClient>();
00848 * auto robot_name = rpc_cli->getRobotNames().front();
00849 * rpc_cli->getRobotInterface(robot_name)->getRobotManage()->setOperationalMode(OperationalModeType::Automatic);
00850 * @endcode
00851 * \endchinese
00852 * \english
00853 * Set the robot operational mode
00854 *
00855 * @param mode Operational mode
00856 *
00857 * @return Returns 0 on success; error code on failure
00858 * -AUBO_BAD_STATE
00859 *
00860 * @throws arcs::common_interface::AuboException
00861 *
00862 * @par Python function prototype
00863 * setOperationalMode(self: pyaubo_sdk.RobotManage, arg0:
00864 * arcs::common_interface::OperationalModeType) -> int

```

```

00865  *
00866  * @par Lua function prototype
00867  * setOperationalMode(mode: number) -> number
00868  *
00869  * @par Lua example
00870  * num = setOperationalMode("Manual")
00871  *
00872  * @par JSON-RPC request example
00873  * {"jsonrpc":"2.0","method":"rob1.RobotManage.setOperationalMode","params":["Manual"],"id":1}
00874  *
00875  * @par JSON-RPC response example
00876  * {"id":1,"jsonrpc":"2.0","result":0}
00877  *
00878  * @par C++ example
00879  * @code
00880  * auto rpc_cli = std::make_shared<RpcClient>();
00881  * auto robot_name = rpc_cli->getRobotNames().front();
00882  *
00883  * rpc_cli->getRobotInterface(robot_name)->getRobotManage()->setOperationalMode(OperationalModeType::Automatic);
00884  * @endcode
00885  * \endenglish
00886  */
00887  int setOperationalMode(OperationalModeType mode);
00888  /**
00889  * @ingroup RobotManage
00890  * \chinese
00891  *
00892  * @return
00893  *
00894  * @throws arcs::common_interface::AuboException
00895  *
00896  * @par Python
00897  * getOperationalMode(self: pyaubo_sdk.RobotManage) ->
00898  * arcs::common_interface::OperationalModeType
00899  *
00900  * @par Lua
00901  * getOperationalMode() -> number
00902  *
00903  * @par Lua
00904  * num = getOperationalMode()
00905  *
00906  * @par JSON-RPC
00907  * {"jsonrpc":"2.0","method":"rob1.RobotManage.getOperationalMode","params":[],"id":1}
00908  *
00909  * @par JSON-RPC
00910  * {"id":1,"jsonrpc":"2.0","result":"Manual"}
00911  *
00912  * @par C++
00913  * @code
00914  * auto rpc_cli = std::make_shared<RpcClient>();
00915  * auto robot_name = rpc_cli->getRobotNames().front();
00916  * OperationalModeType mode =
00917  * rpc_cli->getRobotInterface(robot_name)->getRobotManage()->getOperationalMode();
00918  * @endcode
00919  * \endchinese
00920  * \english
00921  * Get the robot operational mode
00922  *
00923  * @return Robot operational mode
00924  *
00925  * @throws arcs::common_interface::AuboException
00926  *
00927  * @par Python function prototype
00928  * getOperationalMode(self: pyaubo_sdk.RobotManage) ->
00929  * arcs::common_interface::OperationalModeType
00930  *
00931  * @par Lua function prototype
00932  * getOperationalMode() -> number
00933  *
00934  * @par Lua example
00935  * num = getOperationalMode()
00936  *
00937  * @par JSON-RPC request example
00938  * {"jsonrpc":"2.0","method":"rob1.RobotManage.getOperationalMode","params":[],"id":1}
00939  *
00940  * @par JSON-RPC response example
00941  * {"id":1,"jsonrpc":"2.0","result":"Manual"}
00942  *
00943  * @par C++ example
00944  * @code
00945  * auto rpc_cli = std::make_shared<RpcClient>();
00946  * auto robot_name = rpc_cli->getRobotNames().front();
00947  * OperationalModeType mode =
00948  * rpc_cli->getRobotInterface(robot_name)->getRobotManage()->getOperationalMode();
00949  * @endcode
00950

```

```

00951     * \endenglish
00952     */
00953     OperationalModeType getOperationalMode();
00954
00955     /**
00956     * @ingroup RobotManage
00957     * \chinese
00958     *
00959     *
00960     * @return
00961     *
00962     * @throws arcs::common_interface::AuboException
00963     *
00964     * @par Python
00965     * getRobotControlMode(self: pyaubo_sdk.RobotManage) ->
00966     * arcs::common_interface::RobotControlModeType
00967     *
00968     * @par Lua
00969     * getRobotControlMode() -> number
00970     *
00971     * @par Lua
00972     * num = getRobotControlMode()
00973     *
00974     * @par JSON-RPC
00975     * {"jsonrpc":"2.0","method":"rob1.RobotManage.getRobotControlMode","params":[],"id":1}
00976     *
00977     * @par JSON-RPC
00978     * {"id":1,"jsonrpc":"2.0","result":"Position"}
00979     *
00980     * @par C++
00981     * @code
00982     * auto rpc_cli = std::make_shared<RpcClient>();
00983     * auto robot_name = rpc_cli->getRobotNames().front();
00984     * RobotControlModeType mode =
00985     * rpc_cli->getRobotInterface(robot_name)->getRobotManage()->getRobotControlMode();
00986     * @endcode
00987     * \endchinese
00988     * \english
00989     * Get the control mode
00990     *
00991     * @return Control mode
00992     *
00993     * @throws arcs::common_interface::AuboException
00994     *
00995     * @par Python function prototype
00996     * getRobotControlMode(self: pyaubo_sdk.RobotManage) ->
00997     * arcs::common_interface::RobotControlModeType
00998     *
00999     * @par Lua function prototype
01000     * getRobotControlMode() -> number
01001     *
01002     * @par Lua example
01003     * num = getRobotControlMode()
01004     *
01005     * @par JSON-RPC request example
01006     * {"jsonrpc":"2.0","method":"rob1.RobotManage.getRobotControlMode","params":[],"id":1}
01007     *
01008     * @par JSON-RPC response example
01009     * {"id":1,"jsonrpc":"2.0","result":"Position"}
01010     *
01011     * @par C++ example
01012     * @code
01013     * auto rpc_cli = std::make_shared<RpcClient>();
01014     * auto robot_name = rpc_cli->getRobotNames().front();
01015     * RobotControlModeType mode =
01016     * rpc_cli->getRobotInterface(robot_name)->getRobotManage()->getRobotControlMode();
01017     * @endcode
01018     * \endenglish
01019     */
01020     RobotControlModeType getRobotControlMode();
01021
01022     /**
01023     * @ingroup RobotManage
01024     * \chinese
01025     *
01026     *
01027     * @return true; false
01028     *
01029     * @throws arcs::common_interface::AuboException
01030     *
01031     * @par Python
01032     * isFreedriveEnabled(self: pyaubo_sdk.RobotManage) -> bool
01033     *
01034     * @par Lua
01035     * isFreedriveEnabled() -> boolean
01036     *
01037     * @par Lua

```

```

01038 * FreedriveEnabled = isFreedriveEnabled()
01039 *
01040 * @par JSON-RPC
01041 * {"jsonrpc":"2.0","method":"rob1.RobotManage.isFreedriveEnabled","params":[],"id":1}
01042 *
01043 * @par JSON-RPC
01044 * {"id":1,"jsonrpc":"2.0","result":false}
01045 *
01046 * @par C++
01047 * @code
01048 * auto rpc_cli = std::make_shared<RpcClient>();
01049 * auto robot_name = rpc_cli->getRobotNames().front();
01050 * bool isEnabled =
01051 * rpc_cli->getRobotInterface(robot_name)->getRobotManage()->isFreedriveEnabled();
01052 * @endcode
01053 * \endchinese
01054 * \english
01055 * Whether the freedrive mode is enabled
01056 *
01057 * @return Returns true if enabled; otherwise returns false
01058 *
01059 * @throws arcs::common_interface::AuboException
01060 *
01061 * @par Python function prototype
01062 * isFreedriveEnabled(self: pyaubo_sdk.RobotManage) -> bool
01063 *
01064 * @par Lua function prototype
01065 * isFreedriveEnabled() -> boolean
01066 *
01067 * @par Lua example
01068 * FreedriveEnabled = isFreedriveEnabled()
01069 *
01070 * @par JSON-RPC request example
01071 * {"jsonrpc":"2.0","method":"rob1.RobotManage.isFreedriveEnabled","params":[],"id":1}
01072 *
01073 * @par JSON-RPC response example
01074 * {"id":1,"jsonrpc":"2.0","result":false}
01075 *
01076 * @par C++ example
01077 * @code
01078 * auto rpc_cli = std::make_shared<RpcClient>();
01079 * auto robot_name = rpc_cli->getRobotNames().front();
01080 * bool isEnabled =
01081 * rpc_cli->getRobotInterface(robot_name)->getRobotManage()->isFreedriveEnabled();
01082 * @endcode
01083 * \endenglish
01084 */
01085 bool isFreedriveEnabled();
01086
01087 /**
01088 * @ingroup RobotManage
01089 * \chinese
01090 *
01091 *
01092 * @return true; false
01093 *
01094 * @throws arcs::common_interface::AuboException
01095 *
01096 * @par Python
01097 * isBackdriveEnabled(self: pyaubo_sdk.RobotManage) -> bool
01098 *
01099 * @par Lua
01100 * isBackdriveEnabled() -> boolean
01101 *
01102 * @par Lua
01103 * BackdriveEnabled = isBackdriveEnabled()
01104 *
01105 * @par JSON-RPC
01106 * {"jsonrpc":"2.0","method":"rob1.RobotManage.isBackdriveEnabled","params":[],"id":1}
01107 *
01108 * @par JSON-RPC
01109 * {"id":1,"jsonrpc":"2.0","result":false}
01110 *
01111 * @par C++
01112 * @code
01113 * auto rpc_cli = std::make_shared<RpcClient>();
01114 * auto robot_name = rpc_cli->getRobotNames().front();
01115 * bool isEnabled =
01116 * rpc_cli->getRobotInterface(robot_name)->getRobotManage()->isBackdriveEnabled();
01117 * @endcode
01118 * \endchinese
01119 * \english
01120 * Whether the backdrive mode is enabled
01121 *
01122 * @return Returns true if enabled; otherwise returns false
01123 *
01124 * @throws arcs::common_interface::AuboException

```

```

01125     *
01126     * @par Python function prototype
01127     * isBackdriveEnabled(self: pyaubo_sdk.RobotManage) -> bool
01128     *
01129     * @par Lua function prototype
01130     * isBackdriveEnabled() -> boolean
01131     *
01132     * @par Lua example
01133     * BackdriveEnabled = isBackdriveEnabled()
01134     *
01135     * @par JSON-RPC request example
01136     * {"jsonrpc":"2.0","method":"rob1.RobotManage.isBackdriveEnabled","params":[],"id":1}
01137     *
01138     * @par JSON-RPC response example
01139     * {"id":1,"jsonrpc":"2.0","result":false}
01140     *
01141     * @par C++ example
01142     * @code
01143     * auto rpc_cli = std::make_shared<RpcClient>();
01144     * auto robot_name = rpc_cli->getRobotNames().front();
01145     * bool isEnabled =
01146     * rpc_cli->getRobotInterface(robot_name)->getRobotManage()->isBackdriveEnabled();
01147     * @endcode
01148     * \endenglish
01149     */
01150     bool isBackdriveEnabled();
01151
01152     /**
01153     * @ingroup RobotManage
01154     * \chinese
01155     *
01156     *
01157     * @return true; false
01158     *
01159     * @throws arcs::common_interface::AuboException
01160     *
01161     * @par Python
01162     * isSimulationEnabled(self: pyaubo_sdk.RobotManage) -> bool
01163     *
01164     * @par Lua
01165     * isSimulationEnabled() -> boolean
01166     *
01167     * @par Lua
01168     * SimulationEnabled = isSimulationEnabled()
01169     *
01170     * @par JSON-RPC
01171     * {"jsonrpc":"2.0","method":"rob1.RobotManage.isSimulationEnabled","params":[],"id":1}
01172     *
01173     * @par JSON-RPC
01174     * {"id":1,"jsonrpc":"2.0","result":true}
01175     *
01176     * @par C++
01177     * @code
01178     * auto rpc_cli = std::make_shared<RpcClient>();
01179     * auto robot_name = rpc_cli->getRobotNames().front();
01180     * bool isEnabled =
01181     * rpc_cli->getRobotInterface(robot_name)->getRobotManage()->isSimulationEnabled();
01182     * @endcode
01183     * \endchinese
01184     * \english
01185     * Whether the simulation mode is enabled
01186     *
01187     * @return Returns true if enabled; otherwise returns false
01188     *
01189     * @throws arcs::common_interface::AuboException
01190     *
01191     * @par Python function prototype
01192     * isSimulationEnabled(self: pyaubo_sdk.RobotManage) -> bool
01193     *
01194     * @par Lua function prototype
01195     * isSimulationEnabled() -> boolean
01196     *
01197     * @par Lua example
01198     * SimulationEnabled = isSimulationEnabled()
01199     *
01200     * @par JSON-RPC request example
01201     * {"jsonrpc":"2.0","method":"rob1.RobotManage.isSimulationEnabled","params":[],"id":1}
01202     *
01203     * @par JSON-RPC response example
01204     * {"id":1,"jsonrpc":"2.0","result":true}
01205     *
01206     * @par C++ example
01207     * @code
01208     * auto rpc_cli = std::make_shared<RpcClient>();
01209     * auto robot_name = rpc_cli->getRobotNames().front();
01210     * bool isEnabled =
01211     * rpc_cli->getRobotInterface(robot_name)->getRobotManage()->isSimulationEnabled();

```

```

01212     * @endcode
01213     * \endenglish
01214     */
01215     bool isSimulationEnabled();
01216
01217     /**
01218     * @ingroup RobotManage
01219     * \chinese
01220     *
01221     *
01222     * @return 0
01223     * AUBO_BUSY
01224     * AUBO_BAD_STATE
01225     * -AUBO_BAD_STATE
01226     *
01227     * @throws arcs::common_interface::AuboException
01228     *
01229     * @par Python
01230     * setUnlockProtectiveStop(self: pyaubo_sdk.RobotManage) -> int
01231     *
01232     * @par Lua
01233     * setUnlockProtectiveStop() -> number
01234     *
01235     * @par Lua
01236     * num = setUnlockProtectiveStop()
01237     *
01238     * @par JSON-RPC
01239     * {"jsonrpc":"2.0","method":"rob1.RobotManage.setUnlockProtectiveStop","params":[],"id":1}
01240     *
01241     * @par JSON-RPC
01242     * {"id":1,"jsonrpc":"2.0","result":0}
01243     *
01244     * @par C++
01245     * @code
01246     * auto rpc_cli = std::make_shared<RpcClient>();
01247     * auto robot_name = rpc_cli->getRobotNames().front();
01248     * rpc_cli->getRobotInterface(robot_name)->getRobotManage()->setUnlockProtectiveStop();
01249     * @endcode
01250     * \endchinese
01251     * \english
01252     * Clear protective stop, including collision stop
01253     *
01254     * @return Returns 0 on success; error code on failure
01255     * AUBO_BUSY
01256     * AUBO_BAD_STATE
01257     * -AUBO_BAD_STATE
01258     *
01259     * @throws arcs::common_interface::AuboException
01260     *
01261     * @par Python function prototype
01262     * setUnlockProtectiveStop(self: pyaubo_sdk.RobotManage) -> int
01263     *
01264     * @par Lua function prototype
01265     * setUnlockProtectiveStop() -> number
01266     *
01267     * @par Lua example
01268     * num = setUnlockProtectiveStop()
01269     *
01270     * @par JSON-RPC request example
01271     * {"jsonrpc":"2.0","method":"rob1.RobotManage.setUnlockProtectiveStop","params":[],"id":1}
01272     *
01273     * @par JSON-RPC response example
01274     * {"id":1,"jsonrpc":"2.0","result":0}
01275     *
01276     * @par C++ example
01277     * @code
01278     * auto rpc_cli = std::make_shared<RpcClient>();
01279     * auto robot_name = rpc_cli->getRobotNames().front();
01280     * rpc_cli->getRobotInterface(robot_name)->getRobotManage()->setUnlockProtectiveStop();
01281     * @endcode
01282     * \endenglish
01283     */
01284     int setUnlockProtectiveStop();
01285
01286     /**
01287     * @ingroup RobotManage
01288     * \chinese
01289     *
01290     *
01291     * @return 0;
01292     * AUBO_BUSY
01293     * AUBO_BAD_STATE
01294     * -AUBO_BAD_STATE
01295     *
01296     * @throws arcs::common_interface::AuboException
01297     *
01298     * @par Python

```

```

01299 * restartInterfaceBoard(self: pyaubo_sdk.RobotManage) -> int
01300 *
01301 * @par Lua
01302 * restartInterfaceBoard() -> number
01303 *
01304 * @par Lua
01305 * num = restartInterfaceBoard()
01306 *
01307 * @par JSON-RPC
01308 * {"jsonrpc":"2.0","method":"rob1.RobotManage.restartInterfaceBoard","params":[],"id":1}
01309 *
01310 * @par JSON-RPC
01311 * {"id":1,"jsonrpc":"2.0","result":0}
01312 *
01313 * @par C++
01314 * @code
01315 * auto rpc_cli = std::make_shared<RpcClient>();
01316 * auto robot_name = rpc_cli->getRobotNames().front();
01317 * rpc_cli->getRobotInterface(robot_name)->getRobotManage()->restartInterfaceBoard();
01318 * @endcode
01319 * \endchinese
01320 * \english
01321 * Reset the safety interface board, usually called after the robot is
01322 * powered off and needs to be reset, such as after emergency stop or fault.
01323 *
01324 * @return Returns 0 on success; error code on failure
01325 * AUBO_BUSY
01326 * AUBO_BAD_STATE
01327 * -AUBO_BAD_STATE
01328 *
01329 * @throws arcs::common_interface::AuboException
01330 *
01331 * @par Python function prototype
01332 * restartInterfaceBoard(self: pyaubo_sdk.RobotManage) -> int
01333 *
01334 * @par Lua function prototype
01335 * restartInterfaceBoard() -> number
01336 *
01337 * @par Lua example
01338 * num = restartInterfaceBoard()
01339 *
01340 * @par JSON-RPC request example
01341 * {"jsonrpc":"2.0","method":"rob1.RobotManage.restartInterfaceBoard","params":[],"id":1}
01342 *
01343 * @par JSON-RPC response example
01344 * {"id":1,"jsonrpc":"2.0","result":0}
01345 *
01346 * @par C++ example
01347 * @code
01348 * auto rpc_cli = std::make_shared<RpcClient>();
01349 * auto robot_name = rpc_cli->getRobotNames().front();
01350 * rpc_cli->getRobotInterface(robot_name)->getRobotManage()->restartInterfaceBoard();
01351 * @endcode
01352 * \endenglish
01353 */
01354 int restartInterfaceBoard();
01355
01356 /**
01357 * @ingroup RobotManage
01358 * \chinese
01359 *
01360 *
01361 * @param name
01362 *
01363 *
01364 * @return 0 <0
01365 *
01366 * @throws arcs::common_interface::AuboException
01367 *
01368 * @par Lua
01369 * recordCacheFree(name: string) -> nil
01370 *
01371 * @par Lua
01372 * recordCacheFree("rec")
01373 *
01374 * \endchinese
01375 * \english
01376 * Free and clear recorded data of the specified memory cache
01377 *
01378 * @param name
01379 * Cache name
01380 *
01381 * @return Returns 0 on success; negative error code on failure
01382 *
01383 * @throws arcs::common_interface::AuboException
01384 *
01385 * @par Lua function prototype

```



```

01386     * recordCacheFree(name: string) -> nil
01387     *
01388     * @par Lua example
01389     * recordCacheFree("rec")
01390     *
01391     * \endenglish
01392     */
01393     int recordCacheFree(const std::string &name);
01394
01395     /**
01396     * @ingroup RobotManage
01397     * \chinese
01398     *
01399     *
01400     * @param name
01401     *
01402     *
01403     * @return 0 <0
01404     *
01405     * @throws arcs::common_interface::AuboException
01406     *
01407     * @par Lua
01408     * startRecordCache(name: string) -> nil
01409     *
01410     * @par Lua
01411     * startRecordCache("rec")
01412     *
01413     * \endchinese
01414     * \english
01415     * Start real-time trajectory recording to memory cache (no file output)
01416     *
01417     * @param name
01418     * Cache name used to index recorded data in memory. Empty string returns
01419     * invalid argument.
01420     *
01421     * @return Returns 0 on success; negative error code on failure
01422     *
01423     * @throws arcs::common_interface::AuboException
01424     *
01425     * @par Lua function prototype
01426     * startRecordCache(name: string) -> nil
01427     *
01428     * @par Lua example
01429     * startRecordCache("rec")
01430     *
01431     * \endenglish
01432     */
01433     int startRecordCache(const std::string &name);
01434
01435     /**
01436     * @ingroup RobotManage
01437     * \chinese
01438     *
01439     *
01440     * @return 0 <0
01441     *
01442     * @throws arcs::common_interface::AuboException
01443     *
01444     * @par Lua
01445     * stopRecordCache() -> nil
01446     *
01447     * @par Lua
01448     * stopRecordCache()
01449     *
01450     * \endchinese
01451     * \english
01452     * Stop current real-time trajectory recording to memory cache
01453     *
01454     * @return Returns 0 on success; negative error code on failure
01455     *
01456     * @throws arcs::common_interface::AuboException
01457     *
01458     * @par Lua function prototype
01459     * stopRecordCache() -> nil
01460     *
01461     * @par Lua example
01462     * stopRecordCache()
01463     *
01464     * \endenglish
01465     */
01466     int stopRecordCache();
01467
01468     /**
01469     * @ingroup RobotManage
01470     * \chinese
01471     * /
01472     *

```

```

01473 * @param pause
01474 * true    false
01475 *
01476 * @return  0    <0
01477 *
01478 * @throws arcs::common_interface::AuboException
01479 *
01480 * @par Lua
01481 * pauseRecordCache(pause: boolean) -> nil
01482 *
01483 * @par Lua
01484 * pauseRecordCache(true)  --
01485 * pauseRecordCache(false) --
01486 *
01487 * \endchinese
01488 * \english
01489 * Pause or resume current real-time trajectory recording to memory cache
01490 *
01491 * @param pause
01492 * true to pause cache recording; false to resume cache recording
01493 *
01494 * @return Returns 0 on success; negative error code on failure
01495 *
01496 * @throws arcs::common_interface::AuboException
01497 *
01498 * @par Lua function prototype
01499 * pauseRecordCache(pause: boolean) -> nil
01500 *
01501 * @par Lua example
01502 * pauseRecordCache(true)  -- pause
01503 * pauseRecordCache(false) -- resume
01504 *
01505 * \endenglish
01506 */
01507 int pauseRecordCache(bool pause);
01508
01509 /**
01510 * @ingroup RobotManage
01511 * \chinese
01512 *
01513 *
01514 * @param name
01515 *
01516 *
01517 * @param frames
01518 *
01519 * - frames=0      1=1
01520 * - frames>0      frames
01521 *
01522 * @return         >=0    <0
01523 *
01524 * @throws arcs::common_interface::AuboException
01525 *
01526 * @par Lua
01527 * getRecordCache(name: string, frames: number = 0) -> number
01528 *
01529 * @par Lua
01530 * n = getRecordCache("rec")          --
01531 * n = getRecordCache("rec", 2000)    -- 2000
01532 *
01533 * \endchinese
01534 * \english
01535 * Get recorded data from the specified memory cache
01536 *
01537 * @param name
01538 * Cache name
01539 *
01540 * @param frames
01541 * Frame limit:
01542 * - frames=0: get all recorded frames (1 frame = 1 control cycle)
01543 * - frames>0: get up to the latest 'frames' frames
01544 *
01545 * @return Returns the number of frames fetched (>=0) on success;
01546 *         negative error code on failure
01547 *
01548 * @throws arcs::common_interface::AuboException
01549 *
01550 * @par Lua function prototype
01551 * getRecordCache(name: string, frames: number = 0) -> number
01552 *
01553 * @par Lua example
01554 * n = getRecordCache("rec")          -- all frames, returns count
01555 * n = getRecordCache("rec", 2000)    -- latest 2000 frames, returns count
01556 *
01557 * \endenglish
01558 */
01559 int getRecordCache(const std::string &name, size_t frames = 0);

```

```

01560
01561 /**
01562  * @ingroup RobotManage
01563  * \chinese
01564  *
01565  *
01566  * @param file_name
01567  *
01568  * @return 0
01569  * AUBO_BUSY
01570  * AUBO_BAD_STATE
01571  * -AUBO_INVL_ARGUMENT
01572  * -AUBO_BAD_STATE
01573  *
01574  * @throws arcs::common_interface::AuboException
01575  *
01576  * @par Lua
01577  * startRecord(fiel_name: string) -> nil
01578  *
01579  * @par Lua
01580  * startRecord("traje.csv")
01581  *
01582  * \endchinese
01583  * \english
01584  * Start real-time trajectory recording
01585  *
01586  * @param file_name
01587  *
01588  * @return Returns 0 on success; error code on failure
01589  * AUBO_BUSY
01590  * AUBO_BAD_STATE
01591  * -AUBO_INVL_ARGUMENT
01592  * -AUBO_BAD_STATE
01593  *
01594  * @throws arcs::common_interface::AuboException
01595  *
01596  * @par Lua function prototype
01597  * startRecord(fiel_name: string) -> nil
01598  *
01599  * @par Lua example
01600  * startRecord("traje.csv")
01601  *
01602  * \endenglish
01603  */
01604 int startRecord(const std::string &file_name);
01605
01606 /**
01607  * @ingroup RobotManage
01608  * \chinese
01609  *
01610  *
01611  * @return 0
01612  * AUBO_BUSY
01613  * AUBO_BAD_STATE
01614  * -AUBO_BAD_STATE
01615  *
01616  * @throws arcs::common_interface::AuboException
01617  *
01618  * @par Lua
01619  * stopRecord() -> nil
01620  *
01621  * @par Lua
01622  * stopRecord()
01623  *
01624  * @par JSON-RPC
01625  * {"jsonrpc":"2.0","method":"rob1.RobotManage.stopRecord","params":[],"id":1}
01626  *
01627  * @par JSON-RPC
01628  * {"id":1,"jsonrpc":"2.0","result":0}
01629  *
01630  * \endchinese
01631  * \english
01632  * Stop real-time recording
01633  *
01634  * @return Returns 0 on success; error code on failure
01635  * AUBO_BUSY
01636  * AUBO_BAD_STATE
01637  * -AUBO_BAD_STATE
01638  *
01639  * @throws arcs::common_interface::AuboException
01640  *
01641  * @par Lua function prototype
01642  * stopRecord() -> nil
01643  *
01644  * @par Lua example
01645  * stopRecord()
01646  *

```

```

01647 * @par JSON-RPC request example
01648 * {"jsonrpc":"2.0","method":"rob1.RobotManage.stopRecord","params":[],"id":1}
01649 *
01650 * @par JSON-RPC response example
01651 * {"id":1,"jsonrpc":"2.0","result":0}
01652 *
01653 * \endenglish
01654 */
01655 int stopRecord();
01656
01657 /**
01658 * @ingroup RobotManage
01659 * \chinese
01660 *
01661 *
01662 * @param pause
01663 *
01664 * @return 0
01665 * AUBO_BUSY
01666 * AUBO_BAD_STATE
01667 * -AUBO_BAD_STATE
01668 *
01669 * @throws arcs::common_interface::AuboException
01670 *
01671 * @par JSON-RPC
01672 * {"jsonrpc":"2.0","method":"rob1.RobotManage.pauseRecord","params":[true],"id":1}
01673 *
01674 * @par JSON-RPC
01675 * {"id":1,"jsonrpc":"2.0","result":0}
01676 *
01677 * \endchinese
01678 * \english
01679 * Pause real-time recording
01680 *
01681 * @param pause
01682 *
01683 * @return Returns 0 on success; error code on failure
01684 * AUBO_BUSY
01685 * AUBO_BAD_STATE
01686 * -AUBO_BAD_STATE
01687 *
01688 * @throws arcs::common_interface::AuboException
01689 *
01690 * @par JSON-RPC request example
01691 * {"jsonrpc":"2.0","method":"rob1.RobotManage.pauseRecord","params":[true],"id":1}
01692 *
01693 * @par JSON-RPC response example
01694 * {"id":1,"jsonrpc":"2.0","result":0}
01695 *
01696 * \endenglish
01697 */
01698 int pauseRecord(bool pause);
01699
01700 /**
01701 * @ingroup RobotManage
01702 * \chinese
01703 * / ,
01704 *
01705 *
01706 * @param enable
01707 *
01708 * @return 0;
01709 * AUBO_BUSY
01710 * AUBO_REQUEST_IGNORE
01711 * -AUBO_BAD_STATE
01712 *
01713 * @throws arcs::common_interface::AuboException
01714 *
01715 * @par Python
01716 * setLinkModeEnable(self: pyaubo_sdk.RobotManage, arg0: bool) -> int
01717 *
01718 * @par Lua
01719 * setLinkModeEnable(enable: boolean) -> number
01720 *
01721 * @par Lua
01722 * num = setLinkModeEnable(true)
01723 *
01724 * @par JSON-RPC
01725 * {"jsonrpc":"2.0","method":"rob1.RobotManage.setLinkModeEnable","params":[true],"id":1}
01726 *
01727 * @par JSON-RPC
01728 * {"id":1,"jsonrpc":"2.0","result":0}
01729 *
01730 * @par C++
01731 * @code
01732 * auto rpc_cli = std::make_shared<RpcClient>();
01733 * auto robot_name = rpc_cli->getRobotNames().front();

```

```

01734 * rpc_cli->getRobotInterface(robot_name)->getRobotManage()->setLinkModeEnable(true);
01735 * @endcode
01736 * \endchinese
01737 * \english
01738 * Initiate robot enter/exit link mode request.
01739 * Link mode can only be enabled when the operation mode is Automatic or
01740 * None.
01741 *
01742 * @param enable
01743 *
01744 * @return Returns 0 on success; error code on failure
01745 * AUBO_BUSY
01746 * AUBO_REQUEST_IGNORE
01747 * -AUBO_BAD_STATE
01748 *
01749 * @throws arcs::common_interface::AuboException
01750 *
01751 * @par Python function prototype
01752 * setLinkModeEnable(self: pyaubo_sdk.RobotManage, arg0: bool) -> int
01753 *
01754 * @par Lua function prototype
01755 * setLinkModeEnable(enable: boolean) -> number
01756 *
01757 * @par Lua example
01758 * num = setLinkModeEnable(true)
01759 *
01760 * @par JSON-RPC request example
01761 * {"jsonrpc":"2.0","method":"rob1.RobotManage.setLinkModeEnable","params":[true],"id":1}
01762 *
01763 * @par JSON-RPC response example
01764 * {"id":1,"jsonrpc":"2.0","result":0}
01765 *
01766 * @par C++ example
01767 * @code
01768 * auto rpc_cli = std::make_shared<RpcClient>();
01769 * auto robot_name = rpc_cli->getRobotNames().front();
01770 * rpc_cli->getRobotInterface(robot_name)->getRobotManage()->setLinkModeEnable(true);
01771 * @endcode
01772 * \endenglish
01773 */
01774 int setLinkModeEnable(bool enable);
01775
01776 /**
01777 * @ingroup RobotManage
01778 * \chinese
01779 *          IO      IO
01780 *
01781 * @return true; false
01782 *
01783 * @throws arcs::common_interface::AuboException
01784 *
01785 * @par Python
01786 * isLinkModeEnabled(self: pyaubo_sdk.RobotManage) -> bool
01787 *
01788 * @par Lua
01789 * isLinkModeEnabled() -> boolean
01790 *
01791 * @par Lua
01792 * LinkModeEnabled = isLinkModeEnabled()
01793 *
01794 * @par JSON-RPC
01795 * {"jsonrpc":"2.0","method":"rob1.RobotManage.isLinkModeEnabled","params":[],"id":1}
01796 *
01797 * @par JSON-RPC
01798 * {"id":1,"jsonrpc":"2.0","result":false}
01799 *
01800 * @par C++
01801 * @code
01802 * auto rpc_cli = std::make_shared<RpcClient>();
01803 * auto robot_name = rpc_cli->getRobotNames().front();
01804 * bool isEnabled =
01805 * rpc_cli->getRobotInterface(robot_name)->getRobotManage()->isLinkModeEnabled();
01806 * @endcode
01807 * \endchinese
01808 * \english
01809 * Whether the link mode is enabled. In link mode, users can control the
01810 * robot via external IO (users can configure the IO functions).
01811 *
01812 * @return Returns true if enabled; otherwise returns false
01813 *
01814 * @throws arcs::common_interface::AuboException
01815 *
01816 * @par Python function prototype
01817 * isLinkModeEnabled(self: pyaubo_sdk.RobotManage) -> bool
01818 *
01819 * @par Lua function prototype
01820 * isLinkModeEnabled() -> boolean

```

```

01821 *
01822 * @par Lua example
01823 * LinkModeEnabled = isLinkModeEnabled()
01824 *
01825 * @par JSON-RPC request example
01826 * {"jsonrpc":"2.0","method":"rob1.RobotManage.isLinkModeEnabled","params":[],"id":1}
01827 *
01828 * @par JSON-RPC response example
01829 * {"id":1,"jsonrpc":"2.0","result":false}
01830 *
01831 * @par C++ example
01832 * @code
01833 * auto rpc_cli = std::make_shared<RpcClient>();
01834 * auto robot_name = rpc_cli->getRobotNames().front();
01835 * bool isEnabled =
01836 * rpc_cli->getRobotInterface(robot_name)->getRobotManage()->isLinkModeEnabled();
01837 * @endcode
01838 * \endenglish
01839 */
01840 bool isLinkModeEnabled();
01841
01842 /**
01843 * @ingroup RobotManage
01844 * \chinese
01845 *
01846 *
01847 * @return 0; \n
01848 * -AUBO_BAD_STATE: (RuntimeMachine) Stopped,
01849 * AUBO_BAD_STATE 1 \n
01850 * -AUBO_TIMEOUT: AUBO_TIMEOUT 4 \n
01851 *
01852 * @throws arcs::common_interface::AuboException
01853 *
01854 * @par Python
01855 * generateDiagnoseFile(self: pyaubo_sdk.RobotManage, arg0: str) -> int
01856 *
01857 * @par Lua
01858 * generateDiagnoseFile(reason: string) -> nil
01859 *
01860 * @par Lua
01861 * generateDiagnoseFile("reason")
01862 *
01863 * @par JSON-RPC
01864 * {"jsonrpc":"2.0","method":"rob1.RobotManage.generateDiagnoseFile","params":["reason"],"id":1}
01865 *
01866 * @par JSON-RPC
01867 * {"id":1,"jsonrpc":"2.0","result":false}
01868 *
01869 * @par C++
01870 * @code
01871 * auto rpc_cli = std::make_shared<RpcClient>();
01872 * auto robot_name = rpc_cli->getRobotNames().front();
01873 * bool isEnabled =
01874 * rpc_cli->getRobotInterface(robot_name)->getRobotManage()->generateDiagnoseFile("reason");
01875 * @endcode
01876 * \endchinese
01877 * \english
01878 * Manually trigger the generation of a diagnostic file
01879 *
01880 * @return Returns 0 if the command is successfully issued; error code on
01881 * failure. \n -AUBO_BAD_STATE: The current state of the RuntimeMachine is
01882 * not Stopped, the firmware upgrade request is rejected. The value of
01883 * AUBO_BAD_STATE is 1. \n -AUBO_TIMEOUT: Timeout. The value of AUBO_TIMEOUT
01884 * is 4. \n
01885 *
01886 * @throws arcs::common_interface::AuboException
01887 *
01888 * @par Python function prototype
01889 * generateDiagnoseFile(self: pyaubo_sdk.RobotManage, arg0: str) -> int
01890 *
01891 * @par Lua function prototype
01892 * generateDiagnoseFile(reason: string) -> nil
01893 *
01894 * @par Lua example
01895 * generateDiagnoseFile("reason")
01896 *
01897 * @par JSON-RPC request example
01898 * {"jsonrpc":"2.0","method":"rob1.RobotManage.generateDiagnoseFile","params":["reason"],"id":1}
01899 *
01900 * @par JSON-RPC response example
01901 * {"id":1,"jsonrpc":"2.0","result":false}
01902 *
01903 * @par C++ example
01904 * @code
01905 * auto rpc_cli = std::make_shared<RpcClient>();
01906 * auto robot_name = rpc_cli->getRobotNames().front();
01907 * bool isEnabled =

```

```

01908     * rpc_cli->getRobotInterface(robot_name)->getRobotManage()->generateDiagnoseFile("reason");
01909     * @endcode
01910     * \endenglish
01911     */
01912     int generateDiagnoseFile(const std::string &reason);
01913
01914 protected:
01915     void *d_;
01916 };
01917 using RobotManagePtr = std::shared_ptr<RobotManage>;
01918 } // namespace common_interface
01919 } // namespace arcs
01920
01921 #endif // AUBO_SDK_ROBOT_CONTROL_INTERFACE_H

```

12.32 include/aubo/robot/robot_state.h File Reference

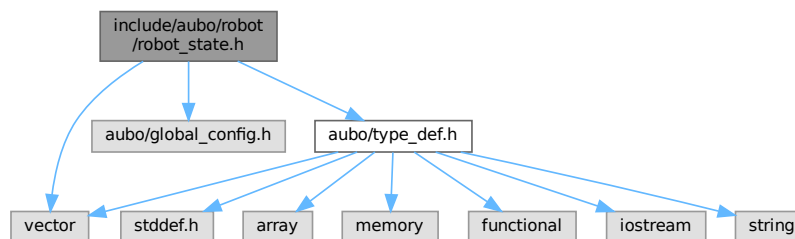
/

```
#include <vector>
```

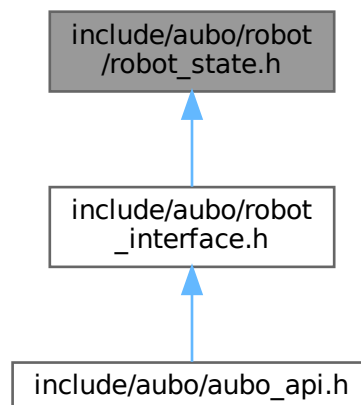
```
#include <aubo/global_config.h>
```

```
#include <aubo/type_def.h>
```

Include dependency graph for robot_state.h:



This graph shows which files directly or indirectly include this file:



Classes

- class [arcs::common_interface::RobotState](#)

Namespaces

- namespace [arcs](#)
- namespace [arcs::common_interface](#)

Typedefs

- using [arcs::common_interface::RobotStatePtr](#) = std::shared_ptr<[RobotState](#)>

12.32.1 Detailed Description

/

Definition in file [robot_state.h](#).

12.33 robot_state.h

[Go to the documentation of this file.](#)

```

00001 /** @file robot_state.h
00002  * @brief
00003  */
00004 #ifndef AUBO_SDK_ROBOT_STATE_INTERFACE_H
00005 #define AUBO_SDK_ROBOT_STATE_INTERFACE_H
00006
00007 #include <vector>
00008
00009 #include <aubo/global_config.h>
00010 #include <aubo/type_def.h>
00011
00012 namespace arcs {
00013 namespace common_interface {
00014
00015 /**
00016  * @defgroup RobotState RobotState ( )
00017  * @ingroup RobotInterface
00018  * RobotState
00019  */
00020 class ARCS_ABI_EXPORT RobotState
00021 {
00022 public:
00023     RobotState();
00024     virtual ~RobotState();
00025
00026     /**
00027      * @ingroup RobotState
00028      * \chinese
00029      *
00030      * @return
00031      *
00032      * @throws arcs::common_interface::AuboException
00033      *
00034      * @par Python
00035      * getRobotModeType(self: pyaubo_sdk.RobotState) ->
00036      * arcs::common_interface::RobotModeType
00037      *
00038      * @par Lua
00039      * getRobotModeType() -> number
00040      *
00041      * @par Lua
00042      * RobotModeType = getRobotModeType()
00043      *
00044      * @par JSON-RPC
00045

```



```

00046 * {"jsonrpc":"2.0","method":"rob1.RobotState.getRobotModeType","params":[],"id":1}
00047 *
00048 * @par JSON-RPC
00049 * {"id":1,"jsonrpc":"2.0","result":"Running"}
00050 *
00051 * \endchinese
00052 * \english
00053 * Get the robot mode state
00054 *
00055 * @return The robot's mode state
00056 *
00057 * @throws arcs::common_interface::AuboException
00058 *
00059 * @par Python function prototype
00060 * getRobotModeType(self: pyaubo_sdk.RobotState) ->
00061 * arcs::common_interface::RobotModeType
00062 *
00063 * @par Lua function prototype
00064 * getRobotModeType() -> number
00065 *
00066 * @par Lua example
00067 * RobotModeType = getRobotModeType()
00068 *
00069 * @par JSON-RPC request example
00070 * {"jsonrpc":"2.0","method":"rob1.RobotState.getRobotModeType","params":[],"id":1}
00071 *
00072 * @par JSON-RPC response example
00073 * {"id":1,"jsonrpc":"2.0","result":"Running"}
00074 * \endenglish
00075 */
00076
00077 RobotModeType getRobotModeType();
00078
00079 /**
00080 * @ingroup RobotState
00081 * \chinese
00082 *
00083 *
00084 * @return
00085 *
00086 * @throws arcs::common_interface::AuboException
00087 *
00088 * @par Python
00089 * getSafetyModeType(self: pyaubo_sdk.RobotState) ->
00090 * arcs::common_interface::SafetyModeType
00091 *
00092 * @par Lua
00093 * getSafetyModeType() -> number
00094 *
00095 * @par Lua
00096 * SafetyModeType = getSafetyModeType()
00097 *
00098 * @par JSON-RPC
00099 * {"jsonrpc":"2.0","method":"rob1.RobotState.getSafetyModeType","params":[],"id":1}
00100 *
00101 * @par JSON-RPC
00102 * {"id":1,"jsonrpc":"2.0","result":"Normal"}
00103 * \endchinese
00104 * \english
00105 * Get the safety mode
00106 *
00107 * @return The safety mode
00108 *
00109 * @throws arcs::common_interface::AuboException
00110 *
00111 * @par Python function prototype
00112 * getSafetyModeType(self: pyaubo_sdk.RobotState) ->
00113 * arcs::common_interface::SafetyModeType
00114 *
00115 * @par Lua function prototype
00116 * getSafetyModeType() -> number
00117 *
00118 * @par Lua example
00119 * SafetyModeType = getSafetyModeType()
00120 *
00121 * @par JSON-RPC request example
00122 * {"jsonrpc":"2.0","method":"rob1.RobotState.getSafetyModeType","params":[],"id":1}
00123 *
00124 * @par JSON-RPC response example
00125 * {"id":1,"jsonrpc":"2.0","result":"Normal"}
00126 * \endenglish
00127 */
00128 SafetyModeType getSafetyModeType();
00129
00130 /**
00131 * @ingroup RobotState
00132 * \chinese

```

```

00133 *
00134 *
00135 * @return true; false
00136 *
00137 * @throws arcs::common_interface::AuboException
00138 *
00139 * @par Lua
00140 * isPowerOn() -> boolean
00141 *
00142 * @par Lua
00143 * PowerOn = isPowerOn()
00144 *
00145 * @par JSON-RPC
00146 * {"jsonrpc":"2.0","method":"rob1.RobotState.isPowerOn","params":[],"id":1}
00147 *
00148 * @par JSON-RPC
00149 * {"id":1,"jsonrpc":"2.0","result":true}
00150 * \endchinese
00151 * \english
00152 * Get the robot power-on state
00153 *
00154 * @return Returns true if the robot is powered on; otherwise returns false
00155 *
00156 * @throws arcs::common_interface::AuboException
00157 *
00158 * @par Lua function prototype
00159 * isPowerOn() -> boolean
00160 *
00161 * @par Lua example
00162 * PowerOn = isPowerOn()
00163 *
00164 * @par JSON-RPC request example
00165 * {"jsonrpc":"2.0","method":"rob1.RobotState.isPowerOn","params":[],"id":1}
00166 *
00167 * @par JSON-RPC response example
00168 * {"id":1,"jsonrpc":"2.0","result":true}
00169 * \endenglish
00170 */
00171 bool isPowerOn();
00172
00173 /**
00174 * @ingroup RobotState
00175 * \chinese
00176 *
00177 *
00178 * @return true; false
00179 *
00180 * @throws arcs::common_interface::AuboException
00181 *
00182 * @par Python
00183 * isSteady(self: pyaubo_sdk.RobotState) -> bool
00184 *
00185 * @par Lua
00186 * isSteady() -> boolean
00187 *
00188 * @par Lua
00189 * Steady = isSteady()
00190 *
00191 * @par JSON-RPC
00192 * {"jsonrpc":"2.0","method":"rob1.RobotState.isSteady","params":[],"id":1}
00193 *
00194 * @par JSON-RPC
00195 * {"id":1,"jsonrpc":"2.0","result":true}
00196 * \endchinese
00197 * \english
00198 * Whether the robot has stopped
00199 *
00200 * @return Returns true if stopped; otherwise returns false
00201 *
00202 * @throws arcs::common_interface::AuboException
00203 *
00204 * @par Python function prototype
00205 * isSteady(self: pyaubo_sdk.RobotState) -> bool
00206 *
00207 * @par Lua function prototype
00208 * isSteady() -> boolean
00209 *
00210 * @par Lua exaple
00211 * Steady = isSteady()
00212 *
00213 * @par JSON-RPC request example
00214 * {"jsonrpc":"2.0","method":"rob1.RobotState.isSteady","params":[],"id":1}
00215 *
00216 * @par JSON-RPC response example
00217 * {"id":1,"jsonrpc":"2.0","result":true}
00218 * \endenglish
00219 */

```

```

00220 bool isSteady();
00221
00222 /**
00223  * @ingroup RobotState
00224  * \chinese
00225  *
00226  *
00227  * @return true; false
00228  *
00229  * @throws arcs::common_interface::AuboException
00230  *
00231  * @par Lua
00232  * isCollisionOccurred() -> boolean
00233  *
00234  * @par Lua
00235  * CollisionOccurred = isCollisionOccurred()
00236  *
00237  * @par JSON-RPC
00238  * {"jsonrpc":"2.0","method":"rob1.RobotState.isCollisionOccurred","params":[],"id":1}
00239  *
00240  * @par JSON-RPC
00241  * {"id":1,"jsonrpc":"2.0","result":false}
00242  * \endchinese
00243  * \english
00244  * Whether a collision has occurred
00245  *
00246  * @return Returns true if a collision occurred; otherwise returns false
00247  *
00248  * @throws arcs::common_interface::AuboException
00249  *
00250  * @par Lua function prototype
00251  * isCollisionOccurred() -> boolean
00252  *
00253  * @par Lua example
00254  * CollisionOccurred = isCollisionOccurred()
00255  *
00256  * @par JSON-RPC request example
00257  * {"jsonrpc":"2.0","method":"rob1.RobotState.isCollisionOccurred","params":[],"id":1}
00258  *
00259  * @par JSON-RPC response example
00260  * {"id":1,"jsonrpc":"2.0","result":false}
00261  * \endenglish
00262  */
00263 bool isCollisionOccurred();
00264
00265 /**
00266  * @ingroup RobotState
00267  * \chinese
00268  *
00269  *
00270  * @return true; false
00271  *
00272  * @throws arcs::common_interface::AuboException
00273  *
00274  * @par Python
00275  * isWithinSafetyLimits(self: pyaubo_sdk.RobotState) -> bool
00276  *
00277  * @par Lua
00278  * isWithinSafetyLimits() -> boolean
00279  *
00280  * @par Lua
00281  * WithinSafetyLimits = isWithinSafetyLimits()
00282  *
00283  * @par JSON-RPC
00284  * {"jsonrpc":"2.0","method":"rob1.RobotState.isWithinSafetyLimits","params":[],"id":1}
00285  *
00286  * @par JSON-RPC
00287  * {"id":1,"jsonrpc":"2.0","result":true}
00288  * \endchinese
00289  * \english
00290  * Whether the robot is within safety limits
00291  *
00292  * @return Returns true if within safety limits; otherwise returns false
00293  *
00294  * @throws arcs::common_interface::AuboException
00295  *
00296  * @par Python function prototype
00297  * isWithinSafetyLimits(self: pyaubo_sdk.RobotState) -> bool
00298  *
00299  * @par Lua function prototype
00300  * isWithinSafetyLimits() -> boolean
00301  *
00302  * @par Lua example
00303  * WithinSafetyLimits = isWithinSafetyLimits()
00304  *
00305  * @par JSON-RPC request example
00306  * {"jsonrpc":"2.0","method":"rob1.RobotState.isWithinSafetyLimits","params":[],"id":1}

```

```

00307 *
00308 * @par JSON-RPC response example
00309 * {"id":1,"jsonrpc":"2.0","result":true}
00310 * \endenglish
00311 */
00312 bool isWithinSafetyLimits();
00313
00314 /**
00315 * @ingroup RobotState
00316 * \chinese
00317 * TCP TCP getActualTcpOffset
00318 *
00319 * (x,y,z,rx,ry,rz)
00320 * x y z TCP m
00321 * rx ry rz TCP ZYX rad
00322 *
00323 * @return TCP , (x,y,z,rx,ry,rz)
00324 *
00325 * @throws arcs::common_interface::AuboException
00326 *
00327 * @par Python
00328 * getTcpPose(self: pyaubo_sdk.RobotState) -> List[float]
00329 *
00330 * @par Lua
00331 * getTcpPose() -> table
00332 *
00333 * @par Lua
00334 * TcpPose = getTcpPose()
00335 *
00336 * @par JSON-RPC
00337 * {"jsonrpc":"2.0","method":"rob1.RobotState.getTcpPose","params":[],"id":1}
00338 *
00339 * @par JSON-RPC
00340 * {"id":1,"jsonrpc":"2.0","result":[0.41777839846910425,-
0.13250000000000012,0.20928451364415995,3.1415792312578987,0.0,1.5707963267948963]}
00341 * \endchinese
00342 * \english
00343 * Get the current TCP pose. The TCP offset can be obtained by
00344 * getActualTcpOffset.
00345 *
00346 * The pose is represented as (x, y, z, rx, ry, rz).
00347 * x, y, z are the position of the Tool Center Point (TCP) in the base
00348 * coordinate system, in meters. rx, ry, rz are the orientation of the TCP
00349 * in the base coordinate system, as ZYX Euler angles in radians.
00350 *
00351 * @return The TCP pose, in the form (x, y, z, rx, ry, rz)
00352 *
00353 * @throws arcs::common_interface::AuboException
00354 *
00355 * @par Python function prototype
00356 * getTcpPose(self: pyaubo_sdk.RobotState) -> List[float]
00357 *
00358 * @par Lua function prototype
00359 * getTcpPose() -> table
00360 *
00361 * @par Lua example
00362 * TcpPose = getTcpPose()
00363 *
00364 * @par JSON-RPC request example
00365 * {"jsonrpc":"2.0","method":"rob1.RobotState.getTcpPose","params":[],"id":1}
00366 *
00367 * @par JSON-RPC response example
00368 * {"id":1,"jsonrpc":"2.0","result":[0.41777839846910425,-
0.13250000000000012,0.20928451364415995,3.1415792312578987,0.0,1.5707963267948963]}
00369 * \endenglish
00370 */
00371 std::vector<double> getTcpPose();
00372
00373 /**
00374 * @ingroup RobotState
00375 * \chinese
00376 * TCP getTcpPose pose TCP
00377 *
00378 * @return TCP
00379 *
00380 * @par Lua
00381 * getActualTcpOffset() -> table
00382 *
00383 * @par Lua
00384 * ActualTcpOffset = getActualTcpOffset()
00385 *
00386 * \endchinese
00387 * \english
00388 * Get the current TCP offset, which is the TCP offset used by the pose
00389 * returned from getTcpPose
00390 *
00391 * @return The current TCP offset

```

```

00392 *
00393 * @par Lua function prototype
00394 * getActualTcpOffset() -> table
00395 *
00396 * @par Lua example
00397 * ActualTcpOffset = getActualTcpOffset()
00398 *
00399 * \endenglish
00400 */
00401 std::vector<double> getActualTcpOffset();
00402
00403 /**
00404 * @ingroup RobotState
00405 * \chinese
00406 *
00407 *     getTcpTargetPose
00408 *
00409 *     (x,y,z,rx,ry,rz)
00410 *     x y z      TCP      m
00411 *     rx ry rz   TCP      ZYX   rad
00412 *
00413 * @return      , (x,y,z,rx,ry,rz)
00414 *
00415 * @throws arcs::common_interface::AuboException
00416 *
00417 * @par Python
00418 * getTargetTcpPose(self: pyaubo_sdk.RobotState) -> List[float]
00419 *
00420 * @par Lua
00421 * getTargetTcpPose() -> table
00422 *
00423 * @par Lua
00424 * TargetTcpPose = getTargetTcpPose()
00425 *
00426 * @par JSON-RPC
00427 * {"jsonrpc":"2.0","method":"rob1.RobotState.getTargetTcpPose","params":[],"id":1}
00428 *
00429 * @par JSON-RPC
00430 * {"id":1,"jsonrpc":"2.0","result":[0.4173932217619493,-
0.13250000000000012,0.43296496133045825,3.141577313781914,0.0,1.5707963267948963]}
00431 * \endchinese
00432 * \english
00433 * Get the next target waypoint.
00434 * Note the difference from getTcpTargetPose; the definition here is
00435 * ambiguous and the naming needs improvement.
00436 *
00437 * The pose is represented as (x, y, z, rx, ry, rz).
00438 * x, y, z are the target position of the Tool Center Point (TCP) in the
00439 * base coordinate system, in meters. rx, ry, rz are the target orientation
00440 * of the TCP in the base coordinate system, as ZYX Euler angles in radians.
00441 *
00442 * @return The current target pose, in the form (x, y, z, rx, ry, rz)
00443 *
00444 * @throws arcs::common_interface::AuboException
00445 *
00446 * @par Python function prototype
00447 * getTargetTcpPose(self: pyaubo_sdk.RobotState) -> List[float]
00448 *
00449 * @par Lua function prototype
00450 * getTargetTcpPose() -> table
00451 *
00452 * @par Lua example
00453 * TargetTcpPose = getTargetTcpPose()
00454 *
00455 * @par JSON-RPC request example
00456 * {"jsonrpc":"2.0","method":"rob1.RobotState.getTargetTcpPose","params":[],"id":1}
00457 *
00458 * @par JSON-RPC response example
00459 * {"id":1,"jsonrpc":"2.0","result":[0.4173932217619493,-
0.13250000000000012,0.43296496133045825,3.141577313781914,0.0,1.5707963267948963]}
00460 * \endenglish
00461 */
00462 std::vector<double> getTargetTcpPose();
00463
00464 /**
00465 * @ingroup RobotState
00466 * \chinese
00467 *     TCP
00468 *
00469 *     (x,y,z,rx,ry,rz)
00470 *     x y z      m
00471 *     rx ry rz   ZYX   rad
00472 *
00473 * @return      , (x,y,z,rx,ry,rz)
00474 *
00475 * @throws arcs::common_interface::AuboException
00476 *

```

```

00477 * @par Python
00478 * getToolPose(self: pyaubo_sdk.RobotState) -> List[float]
00479 *
00480 * @par Lua
00481 * getToolPose() -> table
00482 *
00483 * @par Lua
00484 * ToolPose = getToolPose()
00485 *
00486 * @par JSON-RPC
00487 * {"jsonrpc":"2.0","method":"rob1.RobotState.getToolPose","params":[],"id":1}
00488 *
00489 * @par JSON-RPC
00490 * {"id":1,"jsonrpc":"2.0","result":[0.41777820858878617,-
0.13250000000000012,0.20928410288421018,3.141579231257899,0.0,1.5707963267948963]}
00491 * \endchinese
00492 * \english
00493 * Get the tool pose (without TCP offset)
00494 *
00495 * The pose is represented as (x, y, z, rx, ry, rz).
00496 * x, y, z are the target position of the flange center in the base
00497 * coordinate system, in meters. rx, ry, rz are the target orientation of
00498 * the flange center in the base coordinate system, as ZYX Euler angles in
00499 * radians.
00500 *
00501 * @return The tool pose, in the form (x, y, z, rx, ry, rz)
00502 *
00503 * @throws arcs::common_interface::AuboException
00504 *
00505 * @par Python function prototype
00506 * getToolPose(self: pyaubo_sdk.RobotState) -> List[float]
00507 *
00508 * @par Lua function prototype
00509 * getToolPose() -> table
00510 *
00511 * @par Lua example
00512 * ToolPose = getToolPose()
00513 *
00514 * @par JSON-RPC request example
00515 * {"jsonrpc":"2.0","method":"rob1.RobotState.getToolPose","params":[],"id":1}
00516 *
00517 * @par JSON-RPC response example
00518 * {"id":1,"jsonrpc":"2.0","result":[0.41777820858878617,-
0.13250000000000012,0.20928410288421018,3.141579231257899,0.0,1.5707963267948963]}
00519 * \endenglish
00520 */
00521 std::vector<double> getToolPose();
00522
00523 /**
00524 * @ingroup RobotState
00525 * \chinese
00526 * TCP
00527 *
00528 * @return TCP
00529 *
00530 * @throws arcs::common_interface::AuboException
00531 *
00532 * @par Python
00533 * getTcpSpeed(self: pyaubo_sdk.RobotState) -> List[float]
00534 *
00535 * @par Lua
00536 * getTcpSpeed() -> table
00537 *
00538 * @par Lua
00539 * TcpSpeed = getTcpSpeed()
00540 *
00541 * @par JSON-RPC
00542 * {"jsonrpc":"2.0","method":"rob1.RobotState.getTcpSpeed","params":[],"id":1}
00543 *
00544 * @par JSON-RPC
00545 * {"id":1,"jsonrpc":"2.0","result":[0.0,0.0,0.0,0.0,0.0,0.0]}
00546 * \endchinese
00547 * \english
00548 * Get the TCP speed
00549 *
00550 * @return TCP speed
00551 *
00552 * @throws arcs::common_interface::AuboException
00553 *
00554 * @par Python function prototype
00555 * getTcpSpeed(self: pyaubo_sdk.RobotState) -> List[float]
00556 *
00557 * @par Lua function prototype
00558 * getTcpSpeed() -> table
00559 *
00560 * @par Lua example
00561 * TcpSpeed = getTcpSpeed()

```

```

00562  *
00563  * @par JSON-RPC request example
00564  * {"jsonrpc":"2.0","method":"rob1.RobotState.getTcpSpeed","params":[],"id":1}
00565  *
00566  * @par JSON-RPC response example
00567  * {"id":1,"jsonrpc":"2.0","result":[0.0,0.0,0.0,0.0,0.0,0.0]}
00568  * \endenglish
00569  */
00570 std::vector<double> getTcpSpeed();
00571
00572 /**
00573  * @ingroup RobotState
00574  * \chinese
00575  * TCP /
00576  *
00577  * @return TCP /
00578  *
00579  * @par Python
00580  * getTcpForce(self: pyaubo_sdk.RobotState) -> List[float]
00581  *
00582  * @par Lua
00583  * getTcpForce() -> table
00584  *
00585  * @par Lua
00586  * TcpForce = getTcpForce()
00587  *
00588  * @par JSON-RPC
00589  * {"jsonrpc":"2.0","method":"rob1.RobotState.getTcpForce","params":[],"id":1}
00590  *
00591  * @par JSON-RPC
00592  * {"id":1,"jsonrpc":"2.0","result":[0.0,0.0,0.0,0.0,0.0,0.0]}
00593  * \endchinese
00594  * \english
00595  * Get the TCP force/torque
00596  *
00597  * @return TCP force/torque
00598  *
00599  * @par Python function prototype
00600  * getTcpForce(self: pyaubo_sdk.RobotState) -> List[float]
00601  *
00602  * @par Lua function prototype
00603  * getTcpForce() -> table
00604  *
00605  * @par Lua example
00606  * TcpForce = getTcpForce()
00607  *
00608  * @par JSON-RPC request example
00609  * {"jsonrpc":"2.0","method":"rob1.RobotState.getTcpForce","params":[],"id":1}
00610  *
00611  * @par JSON-RPC response example
00612  * {"id":1,"jsonrpc":"2.0","result":[0.0,0.0,0.0,0.0,0.0,0.0]}
00613  * \endenglish
00614  */
00615 std::vector<double> getTcpForce();
00616
00617 /**
00618  * @ingroup RobotState
00619  * \chinese
00620  *
00621  *
00622  * @return
00623  *
00624  * @throws arcs::common_interface::AuboException
00625  *
00626  * @par Python
00627  * getElbowPosistion(self: pyaubo_sdk.RobotState) -> List[float]
00628  *
00629  * @par Lua
00630  * getElbowPosistion() -> table
00631  *
00632  * @par Lua
00633  * ElbowPosistion = getElbowPosistion()
00634  *
00635  * @par JSON-RPC
00636  * {"jsonrpc":"2.0","method":"rob1.RobotState.getElbowPosistion","params":[],"id":1}
00637  *
00638  * @par JSON-RPC
00639  * {"id":1,"jsonrpc":"2.0","result":[0.07355755887512408,-0.1325,0.43200874126125227,-
1.5707963267948968,0.433006344376404,0.0]}
00640  * \endchinese
00641  * \english
00642  * Get the position of the elbow
00643  *
00644  * @return The position of the elbow
00645  *
00646  * @throws arcs::common_interface::AuboException
00647  *

```

```

00648 * @par Python function prototype
00649 * getElbowPosistion(self: pyaubo_sdk.RobotState) -> List[float]
00650 *
00651 * @par Lua function prototype
00652 * getElbowPosistion() -> table
00653 *
00654 * @par Lua example
00655 * ElbowPosistion = getElbowPosistion()
00656 *
00657 * @par JSON-RPC request example
00658 * {"jsonrpc":"2.0","method":"rob1.RobotState.getElbowPosistion","params":[],"id":1}
00659 *
00660 * @par JSON-RPC response example
00661 * {"id":1,"jsonrpc":"2.0","result":[0.07355755887512408,-0.1325,0.43200874126125227,-
1.5707963267948968,0.433006344376404,0.0]}
00662 * \endenglish
00663 */
00664 std::vector<double> getElbowPosistion();
00665
00666 /**
00667 * @ingroup RobotState
00668 * \chinese
00669 *
00670 *
00671 * @return
00672 *
00673 * @throws arcs::common_interface::AuboException
00674 *
00675 * @par Python
00676 * getElbowVelocity(self: pyaubo_sdk.RobotState) -> List[float]
00677 *
00678 * @par Lua
00679 * getElbowVelocity() -> table
00680 *
00681 * @par Lua
00682 * ElbowVelocity = getElbowVelocity()
00683 *
00684 * @par JSON-RPC
00685 * {"jsonrpc":"2.0","method":"rob1.RobotState.getElbowVelocity","params":[],"id":1}
00686 *
00687 * @par JSON-RPC
00688 * {"id":1,"jsonrpc":"2.0","result":[0.0,0.0,0.0,0.0,0.0,0.0]}
00689 * \endchinese
00690 * \english
00691 * Get the elbow velocity
00692 *
00693 * @return Elbow velocity
00694 *
00695 * @throws arcs::common_interface::AuboException
00696 *
00697 * @par Python function prototype
00698 * getElbowVelocity(self: pyaubo_sdk.RobotState) -> List[float]
00699 *
00700 * @par Lua function prototype
00701 * getElbowVelocity() -> table
00702 *
00703 * @par Lua example
00704 * ElbowVelocity = getElbowVelocity()
00705 *
00706 * @par JSON-RPC request example
00707 * {"jsonrpc":"2.0","method":"rob1.RobotState.getElbowVelocity","params":[],"id":1}
00708 *
00709 * @par JSON-RPC response example
00710 * {"id":1,"jsonrpc":"2.0","result":[0.0,0.0,0.0,0.0,0.0,0.0]}
00711 * \endenglish
00712 */
00713 std::vector<double> getElbowVelocity();
00714
00715 /**
00716 * @ingroup RobotState
00717 * \chinese
00718 * /
00719 *
00720 * @return /
00721 *
00722 * @throws arcs::common_interface::AuboException
00723 *
00724 * @par Python
00725 * getBaseForce(self: pyaubo_sdk.RobotState) -> List[float]
00726 *
00727 * @par Lua
00728 * getBaseForce() -> table
00729 *
00730 * @par Lua
00731 * BaseForce = getBaseForce()
00732 *
00733 * @par JSON-RPC

```



```

00734 * {"jsonrpc":"2.0","method":"rob1.RobotState.getBaseForce","params":[],"id":1}
00735 *
00736 * @par JSON-RPC
00737 * {"id":1,"jsonrpc":"2.0","result":[0.0,0.0,0.0,0.0,0.0,0.0]}
00738 * \endchinese
00739 * \english
00740 * Get the base force/torque
00741 *
00742 * @return Base force/torque
00743 *
00744 * @throws arcs::common_interface::AuboException
00745 *
00746 * @par Python function prototype
00747 * getBaseForce(self: pyaubo_sdk.RobotState) -> List[float]
00748 *
00749 * @par Lua function prototype
00750 * getBaseForce() -> table
00751 *
00752 * @par Lua example
00753 * BaseForce = getBaseForce()
00754 *
00755 * @par JSON-RPC request example
00756 * {"jsonrpc":"2.0","method":"rob1.RobotState.getBaseForce","params":[],"id":1}
00757 *
00758 * @par JSON-RPC response example
00759 * {"id":1,"jsonrpc":"2.0","result":[0.0,0.0,0.0,0.0,0.0,0.0]}
00760 * \endenglish
00761 */
00762 std::vector<double> getBaseForce();
00763
00764 /**
00765 * @ingroup RobotState
00766 * \chinese
00767 * TCP
00768 *
00769 * @return TCP
00770 *
00771 * @throws arcs::common_interface::AuboException
00772 *
00773 * @par Python
00774 * getTcpTargetPose(self: pyaubo_sdk.RobotState) -> List[float]
00775 *
00776 * @par Lua
00777 * getTcpTargetPose() -> table
00778 *
00779 * @par Lua
00780 * TcpTargetPose = getTcpTargetPose()
00781 *
00782 * @par JSON-RPC
00783 * {"jsonrpc":"2.0","method":"rob1.RobotState.getTcpTargetPose","params":[],"id":1}
00784 *
00785 * @par JSON-RPC
00786 * {"id":1,"jsonrpc":"2.0","result":[0.41777829240862013,-
0.132500000000000012,0.2092832117232601,3.1415812372223217,0.0,1.5707963267948963]}
00787 * \endchinese
00788 * \english
00789 * Get the last sent TCP target pose
00790 *
00791 * @return TCP target pose
00792 *
00793 * @throws arcs::common_interface::AuboException
00794 *
00795 * @par Python function prototype
00796 * getTcpTargetPose(self: pyaubo_sdk.RobotState) -> List[float]
00797 *
00798 * @par Lua function prototype
00799 * getTcpTargetPose() -> table
00800 *
00801 * @par Lua example
00802 * TcpTargetPose = getTcpTargetPose()
00803 *
00804 * @par JSON-RPC request example
00805 * {"jsonrpc":"2.0","method":"rob1.RobotState.getTcpTargetPose","params":[],"id":1}
00806 *
00807 * @par JSON-RPC response example
00808 * {"id":1,"jsonrpc":"2.0","result":[0.41777829240862013,-
0.132500000000000012,0.2092832117232601,3.1415812372223217,0.0,1.5707963267948963]}
00809 * \endenglish
00810 */
00811 std::vector<double> getTcpTargetPose();
00812
00813 /**
00814 * @ingroup RobotState
00815 * \chinese
00816 * TCP
00817 *
00818 * @return TCP

```

```

00819 *
00820 * @throws arcs::common_interface::AuboException
00821 *
00822 * @par Python
00823 * getTcpTargetSpeed(self: pyaubo_sdk.RobotState) -> List[float]
00824 *
00825 * @par Lua
00826 * getTcpTargetSpeed() -> table
00827 *
00828 * @par Lua
00829 * TcpTargetSpeed = getTcpTargetSpeed()
00830 *
00831 * @par JSON-RPC
00832 * {"jsonrpc":"2.0","method":"rob1.RobotState.getTcpTargetSpeed","params":[],"id":1}
00833 *
00834 * @par JSON-RPC
00835 * {"id":1,"jsonrpc":"2.0","result":[0.0,0.0,0.0,0.0,0.0,0.0]}
00836 * \endchinese
00837 * \english
00838 * Get the TCP target speed
00839 *
00840 * @return TCP target speed
00841 *
00842 * @throws arcs::common_interface::AuboException
00843 *
00844 * @par Python function prototype
00845 * getTcpTargetSpeed(self: pyaubo_sdk.RobotState) -> List[float]
00846 *
00847 * @par Lua function prototype
00848 * getTcpTargetSpeed() -> table
00849 *
00850 * @par Lua example
00851 * TcpTargetSpeed = getTcpTargetSpeed()
00852 *
00853 * @par JSON-RPC request example
00854 * {"jsonrpc":"2.0","method":"rob1.RobotState.getTcpTargetSpeed","params":[],"id":1}
00855 *
00856 * @par JSON-RPC response example
00857 * {"id":1,"jsonrpc":"2.0","result":[0.0,0.0,0.0,0.0,0.0,0.0]}
00858 * \endenglish
00859 */
00860 std::vector<double> getTcpTargetSpeed();
00861
00862 /**
00863 * @ingroup RobotState
00864 * \chinese
00865 * TCP /
00866 *
00867 * @return TCP /
00868 *
00869 * @throws arcs::common_interface::AuboException
00870 *
00871 * @par Python
00872 * getTcpTargetForce(self: pyaubo_sdk.RobotState) -> List[float]
00873 *
00874 * @par Lua
00875 * getTcpTargetForce() -> table
00876 *
00877 * @par Lua
00878 * TcpTargetForce = getTcpTargetForce()
00879 *
00880 * @par JSON-RPC
00881 * {"jsonrpc":"2.0","method":"rob1.RobotState.getTcpTargetForce","params":[],"id":1}
00882 *
00883 * @par JSON-RPC
00884 * {"id":1,"jsonrpc":"2.0","result":[0.0,0.0,0.0,0.0,0.0,0.0]}
00885 * \endchinese
00886 * \english
00887 * Get the TCP target force/torque
00888 *
00889 * @return TCP target force/torque
00890 *
00891 * @throws arcs::common_interface::AuboException
00892 *
00893 * @par Python function prototype
00894 * getTcpTargetForce(self: pyaubo_sdk.RobotState) -> List[float]
00895 *
00896 * @par Lua function prototype
00897 * getTcpTargetForce() -> table
00898 *
00899 * @par Lua example
00900 * TcpTargetForce = getTcpTargetForce()
00901 *
00902 * @par JSON-RPC request example
00903 * {"jsonrpc":"2.0","method":"rob1.RobotState.getTcpTargetForce","params":[],"id":1}
00904 *
00905 * @par JSON-RPC response example

```

```

00906     * {"id":1,"jsonrpc":"2.0","result":[0.0,0.0,0.0,0.0,0.0,0.0]}
00907     * \endenglish
00908     */
00909     std::vector<double> getTcpTargetForce();
00910
00911     /**
00912     * @ingroup RobotState
00913     * \chinese
00914     *
00915     *
00916     * @return
00917     *
00918     * @throws arcs::common_interface::AuboException
00919     *
00920     * @par Python
00921     * getJointState(self: pyaubo_sdk.RobotState) ->
00922     * List[arcs::common_interface::JointStateType]
00923     *
00924     * @par Lua
00925     * getJointState() -> table
00926     *
00927     * @par Lua
00928     * JointState = getJointState()
00929     *
00930     * @par JSON-RPC
00931     * {"jsonrpc":"2.0","method":"rob1.RobotState.getJointState","params":[],"id":1}
00932     *
00933     * @par JSON-RPC
00934     * {"id":1,"jsonrpc":"2.0","result":["Running","Running","Running","Running","Running","Running","Running"]}
00935     * \endchinese
00936     * \english
00937     * Get the joint state of the manipulator
00938     *
00939     * @return Joint state of the manipulator
00940     *
00941     * @throws arcs::common_interface::AuboException
00942     *
00943     * @par Python function prototype
00944     * getJointState(self: pyaubo_sdk.RobotState) ->
00945     * List[arcs::common_interface::JointStateType]
00946     *
00947     * @par Lua function prototype
00948     * getJointState() -> table
00949     *
00950     * @par Lua example
00951     * JointState = getJointState()
00952     *
00953     * @par JSON-RPC request example
00954     * {"jsonrpc":"2.0","method":"rob1.RobotState.getJointState","params":[],"id":1}
00955     *
00956     * @par JSON-RPC response example
00957     * {"id":1,"jsonrpc":"2.0","result":["Running","Running","Running","Running","Running","Running","Running"]}
00958     * \endenglish
00959     */
00960     std::vector<JointStateType> getJointState();
00961
00962     /**
00963     * @ingroup RobotState
00964     * \chinese
00965     *
00966     *
00967     * @return
00968     *
00969     * @throws arcs::common_interface::AuboException
00970     *
00971     * @par Python
00972     * getJointServoMode(self: pyaubo_sdk.RobotState) ->
00973     * List[arcs::common_interface::JointServoModeType]
00974     *
00975     * @par Lua
00976     * getJointServoMode() -> table
00977     *
00978     * @par Lua
00979     * JointServoMode = getJointServoMode()
00980     *
00981     * @par JSON-RPC
00982     * {"jsonrpc":"2.0","method":"rob1.RobotState.getJointServoMode","params":[],"id":1}
00983     *
00984     * @par JSON-RPC
00985     * {"id":1,"jsonrpc":"2.0","result":["Position","Position","Position","Position","Position","Position","Position"]}
00986     * \endchinese
00987     * \english
00988     * Get the servo state of the joints
00989     *
00990     * @return The servo state of the joints
00991     *
00992     * @throws arcs::common_interface::AuboException

```

```

00993 *
00994 * @par Python function prototype
00995 * getJointServoMode(self: pyaubo_sdk.RobotState) ->
00996 * List[arcs::common_interface::JointServoModeType]
00997 *
00998 * @par Lua function prototype
00999 * getJointServoMode() -> table
01000 *
01001 * @par Lua example
01002 * JointServoMode = getJointServoMode()
01003 *
01004 * @par JSON-RPC request example
01005 * {"jsonrpc":"2.0","method":"rob1.RobotState.getJointServoMode","params":[],"id":1}
01006 *
01007 * @par JSON-RPC response example
01008 * {"id":1,"jsonrpc":"2.0","result":["Position","Position","Position","Position","Position","Position"]}
01009 * \endenglish
01010 */
01011 std::vector<JointServoModeType> getJointServoMode();
01012
01013 /**
01014 * @ingroup RobotState
01015 * \chinese
01016 *
01017 *
01018 * @return
01019 *
01020 * @throws arcs::common_interface::AuboException
01021 *
01022 * @par Python
01023 * getJointPositions(self: pyaubo_sdk.RobotState) -> List[float]
01024 *
01025 * @par Lua
01026 * getJointPositions() -> table
01027 *
01028 * @par Lua
01029 * JointPositions = getJointPositions()
01030 *
01031 * @par JSON-RPC
01032 * {"jsonrpc":"2.0","method":"rob1.RobotState.getJointPositions","params":[],"id":1}
01033 *
01034 * @par JSON-RPC
01035 * {"id":1,"jsonrpc":"2.0","result":[0.0,-
0.26199241371495835,1.7418102574563423,0.4330197667082982,1.5707963267948966,0.0]}
01036 * \endchinese
01037 * \english
01038 * Get the joint positions of the manipulator
01039 *
01040 * @return Joint positions of the manipulator
01041 *
01042 * @throws arcs::common_interface::AuboException
01043 *
01044 * @par Python function prototype
01045 * getJointPositions(self: pyaubo_sdk.RobotState) -> List[float]
01046 *
01047 * @par Lua function prototype
01048 * getJointPositions() -> table
01049 *
01050 * @par Lua example
01051 * JointPositions = getJointPositions()
01052 *
01053 * @par JSON-RPC request example
01054 * {"jsonrpc":"2.0","method":"rob1.RobotState.getJointPositions","params":[],"id":1}
01055 *
01056 * @par JSON-RPC response example
01057 * {"id":1,"jsonrpc":"2.0","result":[0.0,-
0.26199241371495835,1.7418102574563423,0.4330197667082982,1.5707963267948966,0.0]}
01058 * \endenglish
01059 */
01060 std::vector<double> getJointPositions();
01061
01062 /**
01063 * @ingroup RobotState
01064 * \chinese
01065 *
01066 *
01067 * @param steps
01068 * @return
01069 * \endchinese
01070 * \english
01071 * Get the historical joint positions of the manipulator
01072 *
01073 * @param steps Number of steps
01074 * @return Historical joint positions of the manipulator
01075 * \endenglish
01076 */
01077 std::vector<double> getJointPositionsHistory(int steps);

```

```

01078
01079 /**
01080  * @ingroup RobotState
01081  * \chinese
01082  *
01083  *
01084  * @return
01085  *
01086  * @throws arcs::common_interface::AuboException
01087  *
01088  * @par Python
01089  * getJointSpeeds(self: pyaubo_sdk.RobotState) -> List[float]
01090  *
01091  * @par Lua
01092  * getJointSpeeds() -> table
01093  *
01094  * @par Lua
01095  * JointSpeeds = getJointSpeeds()
01096  *
01097  * @par JSON-RPC
01098  * {"jsonrpc":"2.0","method":"rob1.RobotState.getJointSpeeds","params":[],"id":1}
01099  *
01100  * @par JSON-RPC
01101  * {"id":1,"jsonrpc":"2.0","result":[0.0,0.0,0.0,0.0,0.0,0.0]}
01102  * \endchinese
01103  * \english
01104  * Get the joint speeds of the manipulator
01105  *
01106  * @return Joint speeds of the manipulator
01107  *
01108  * @throws arcs::common_interface::AuboException
01109  *
01110  * @par Python function prototype
01111  * getJointSpeeds(self: pyaubo_sdk.RobotState) -> List[float]
01112  *
01113  * @par Lua function prototype
01114  * getJointSpeeds() -> table
01115  *
01116  * @par Lua example
01117  * JointSpeeds = getJointSpeeds()
01118  *
01119  * @par JSON-RPC request example
01120  * {"jsonrpc":"2.0","method":"rob1.RobotState.getJointSpeeds","params":[],"id":1}
01121  *
01122  * @par JSON-RPC response example
01123  * {"id":1,"jsonrpc":"2.0","result":[0.0,0.0,0.0,0.0,0.0,0.0]}
01124  * \endenglish
01125  */
01126 std::vector<double> getJointSpeeds();
01127
01128 /**
01129  * @ingroup RobotState
01130  * \chinese
01131  *
01132  *
01133  * @return
01134  *
01135  * @throws arcs::common_interface::AuboException
01136  *
01137  * @par Python
01138  * getJointAccelerations(self: pyaubo_sdk.RobotState) -> List[float]
01139  *
01140  * @par Lua
01141  * getJointAccelerations() -> table
01142  *
01143  * @par Lua
01144  * JointAccelerations = getJointAccelerations()
01145  *
01146  * @par JSON-RPC
01147  * {"jsonrpc":"2.0","method":"rob1.RobotState.getJointAccelerations","params":[],"id":1}
01148  *
01149  * @par JSON-RPC
01150  * {"id":1,"jsonrpc":"2.0","result":[0.0,0.0,0.0,0.0,0.0,0.0]}
01151  * \endchinese
01152  * \english
01153  * Get the joint accelerations of the manipulator
01154  *
01155  * @return Joint accelerations of the manipulator
01156  *
01157  * @throws arcs::common_interface::AuboException
01158  *
01159  * @par Python function prototype
01160  * getJointAccelerations(self: pyaubo_sdk.RobotState) -> List[float]
01161  *
01162  * @par Lua function prototype
01163  * getJointAccelerations() -> table
01164  *

```

```

01165 * @par Lua example
01166 * JointAccelerations = getJointAccelerations()
01167 *
01168 * @par JSON-RPC request example
01169 * {"jsonrpc":"2.0","method":"rob1.RobotState.getJointAccelerations","params":[],"id":1}
01170 *
01171 * @par JSON-RPC response example
01172 * {"id":1,"jsonrpc":"2.0","result":[0.0,0.0,0.0,0.0,0.0,0.0]}
01173 * \endenglish
01174 */
01175 std::vector<double> getJointAccelerations();
01176
01177 /**
01178 * @ingroup RobotState
01179 * \chinese
01180 *
01181 *
01182 * @return
01183 *
01184 * @throws arcs::common_interface::AuboException
01185 *
01186 * @par Python
01187 * getJointTorqueSensors(self: pyaubo_sdk.RobotState) -> List[float]
01188 *
01189 * @par Lua
01190 * getJointTorqueSensors() -> table
01191 *
01192 * @par Lua
01193 * JointTorqueSensors = getJointTorqueSensors()
01194 *
01195 * @par JSON-RPC
01196 * {"jsonrpc":"2.0","method":"rob1.RobotState.getJointTorqueSensors","params":[],"id":1}
01197 *
01198 * @par JSON-RPC
01199 * {"id":1,"jsonrpc":"2.0","result":[0.0,6275.367736816406,-
7704.2816162109375,3586.9766235351563,503.0364990234375,1506.0882568359375]}
01200 * \endchinese
01201 * \english
01202 * Get the joint torques of the manipulator
01203 *
01204 * @return Joint torques of the manipulator
01205 *
01206 * @throws arcs::common_interface::AuboException
01207 *
01208 * @par Python function prototype
01209 * getJointTorqueSensors(self: pyaubo_sdk.RobotState) -> List[float]
01210 *
01211 * @par Lua function prototype
01212 * getJointTorqueSensors() -> table
01213 *
01214 * @par Lua example
01215 * JointTorqueSensors = getJointTorqueSensors()
01216 *
01217 * @par JSON-RPC request example
01218 * {"jsonrpc":"2.0","method":"rob1.RobotState.getJointTorqueSensors","params":[],"id":1}
01219 *
01220 * @par JSON-RPC response example
01221 * {"id":1,"jsonrpc":"2.0","result":[0.0,6275.367736816406,-
7704.2816162109375,3586.9766235351563,503.0364990234375,1506.0882568359375]}
01222 * \endenglish
01223 */
01224 std::vector<double> getJointTorqueSensors();
01225
01226 /**
01227 * @ingroup RobotState
01228 * \chinese
01229 *
01230 *
01231 * @return
01232 *
01233 * @throws arcs::common_interface::AuboException
01234 *
01235 * @par Python
01236 * getJointContactTorques(self: pyaubo_sdk.RobotState) -> List[float]
01237 *
01238 * @par Lua
01239 * getJointContactTorques() -> table
01240 *
01241 * @par Lua
01242 * JointContactTorques = getJointContactTorques()
01243 *
01244 * @par JSON-RPC
01245 * {"jsonrpc":"2.0","method":"rob1.RobotState.getJointContactTorques","params":[],"id":1}
01246 *
01247 * @par JSON-RPC
01248 * {"id":1,"jsonrpc":"2.0","result":[0.0,0.0,0.0,0.0,0.0,0.0]}
01249 * \endchinese

```

```

01250 * \english
01251 * Get the joint contact torques (external torques) of the manipulator
01252 *
01253 * @return Joint contact torques of the manipulator
01254 *
01255 * @throws arcs::common_interface::AuboException
01256 *
01257 * @par Python function prototype
01258 * getJointContactTorques(self: pyaubo_sdk.RobotState) -> List[float]
01259 *
01260 * @par Lua function prototype
01261 * getJointContactTorques() -> table
01262 *
01263 * @par Lua example
01264 * JointContactTorques = getJointContactTorques()
01265 *
01266 * @par JSON-RPC request example
01267 * {"jsonrpc": "2.0", "method": "rob1.RobotState.getJointContactTorques", "params": [], "id": 1}
01268 *
01269 * @par JSON-RPC response example
01270 * {"id": 1, "jsonrpc": "2.0", "result": [0.0, 0.0, 0.0, 0.0, 0.0, 0.0]}
01271 * \endenglish
01272 */
01273 std::vector<double> getJointContactTorques();
01274
01275 /**
01276 * @ingroup RobotState
01277 * \chinese
01278 *
01279 *
01280 * @return      6      Nm
01281 *
01282 * @throws arcs::common_interface::AuboException
01283 *
01284 * @par Python
01285 * getJointGravityTorques(self: pyaubo_sdk.RobotState) -> List[float]
01286 *
01287 * @par Lua
01288 * getJointGravityTorques() -> table
01289 *
01290 * @par Lua
01291 * local jointGravityTorques = getJointGravityTorques()
01292 *
01293 * @par JSON-RPC
01294 * {"jsonrpc": "2.0", "method": "rob1.RobotState.getJointGravityTorques", "params": [], "id": 1}
01295 *
01296 * @par JSON-RPC
01297 * {"id": 1, "jsonrpc": "2.0", "result": [1.23, -2.34, 3.45, -4.56, 0.78, -0.12]}
01298 * \endchinese
01299 * \english
01300 * Get the joint gravity torque of the manipulator
01301 *
01302 * @return Joint gravity torque of the manipulator, containing torque values
01303 * for 6 joints (unit: Nm)
01304 *
01305 * @throws arcs::common_interface::AuboException
01306 *
01307 * @par Python function prototype
01308 * getJointGravityTorques(self: pyaubo_sdk.RobotState) -> List[float]
01309 *
01310 * @par Lua function prototype
01311 * getJointGravityTorques() -> table
01312 *
01313 * @par Lua example
01314 * local jointGravityTorques = getJointGravityTorques()
01315 *
01316 * @par JSON-RPC request example
01317 * {"jsonrpc": "2.0", "method": "rob1.RobotState.getJointGravityTorques", "params": [], "id": 1}
01318 *
01319 * @par JSON-RPC response example
01320 * {"id": 1, "jsonrpc": "2.0", "result": [1.23, -2.34, 3.45, -4.56, 0.78, -0.12]}
01321 * \endenglish
01322 */
01323 std::vector<double> getJointGravityTorques();
01324
01325 /**
01326 * \chinese
01327 *
01328 *
01329 * @return
01330 *
01331 * @throws arcs::common_interface::AuboException
01332 *
01333 * @par Python
01334 * getBaseForceSensor(self: pyaubo_sdk.RobotState) -> List[float]
01335 *
01336 * @par Lua

```

```

01337 * getBaseForceSensor() -> table
01338 *
01339 * @par Lua
01340 * BaseForceSensor = getBaseForceSensor()
01341 *
01342 * @par JSON-RPC
01343 * {"jsonrpc":"2.0","method":"rob1.RobotState.getBaseForceSensor","params":[],"id":1}
01344 *
01345 * @par JSON-RPC
01346 * {"id":1,"jsonrpc":"2.0","result":[0.0,0.0,0.0,0.0,0.0,0.0]}
01347 * \endchinese
01348 * \english
01349 * Get the base force sensor readings
01350 *
01351 * @return Base force sensor readings
01352 *
01353 * @throws arcs::common_interface::AuboException
01354 *
01355 * @par Python function prototype
01356 * getBaseForceSensor(self: pyaubo_sdk.RobotState) -> List[float]
01357 *
01358 * @par Lua function prototype
01359 * getBaseForceSensor() -> table
01360 *
01361 * @par Lua example
01362 * BaseForceSensor = getBaseForceSensor()
01363 *
01364 * @par JSON-RPC request example
01365 * {"jsonrpc":"2.0","method":"rob1.RobotState.getBaseForceSensor","params":[],"id":1}
01366 *
01367 * @par JSON-RPC response example
01368 * {"id":1,"jsonrpc":"2.0","result":[0.0,0.0,0.0,0.0,0.0,0.0]}
01369 * \endenglish
01370 */
01371 std::vector<double> getBaseForceSensor();
01372
01373 /**
01374 * @ingroup RobotState
01375 * \chinese
01376 * TCP
01377 *
01378 * @return TCP
01379 *
01380 * @throws arcs::common_interface::AuboException
01381 *
01382 * @par Python
01383 * getTcpForceSensors(self: pyaubo_sdk.RobotState) -> List[float]
01384 *
01385 * @par Lua
01386 * getTcpForceSensors() -> table
01387 *
01388 * @par Lua
01389 * TcpForceSensors = getTcpForceSensors()
01390 *
01391 * @par JSON-RPC
01392 * {"jsonrpc":"2.0","method":"rob1.RobotState.getTcpForceSensors","params":[],"id":1}
01393 *
01394 * @par JSON-RPC
01395 * {"id":1,"jsonrpc":"2.0","result":[0.0,0.0,0.0,0.0,0.0,0.0]}
01396 * \endchinese
01397 * \english
01398 * Get the TCP force sensor readings
01399 *
01400 * @return TCP force sensor readings
01401 *
01402 * @throws arcs::common_interface::AuboException
01403 *
01404 * @par Python function prototype
01405 * getTcpForceSensors(self: pyaubo_sdk.RobotState) -> List[float]
01406 *
01407 * @par Lua function prototype
01408 * getTcpForceSensors() -> table
01409 *
01410 * @par Lua example
01411 * TcpForceSensors = getTcpForceSensors()
01412 *
01413 * @par JSON-RPC request example
01414 * {"jsonrpc":"2.0","method":"rob1.RobotState.getTcpForceSensors","params":[],"id":1}
01415 *
01416 * @par JSON-RPC response example
01417 * {"id":1,"jsonrpc":"2.0","result":[0.0,0.0,0.0,0.0,0.0,0.0]}
01418 * \endenglish
01419 */
01420 std::vector<double> getTcpForceSensors();
01421
01422 /**
01423 * @ingroup RobotState

```



```

01424     * \chinese
01425     *
01426     *
01427     * @return
01428     *
01429     * @throws arcs::common_interface::AuboException
01430     *
01431     * @par Python
01432     * getJointCurrents(self: pyaubo_sdk.RobotState) -> List[float]
01433     *
01434     * @par Lua
01435     * getJointCurrents() -> table
01436     *
01437     * @par Lua
01438     * JointCurrents = getJointCurrents()
01439     *
01440     * @par JSON-RPC
01441     * {"jsonrpc":"2.0","method":"rob1.RobotState.getJointCurrents","params":[],"id":1}
01442     *
01443     * @par JSON-RPC
01444     * {"id":1,"jsonrpc":"2.0","result":[0.0,1.25885009765625,-
1.5289306640625,0.71868896484375,0.1007080078125,0.3021240234375]}
01445     * \endchinese
01446     * \english
01447     * Get the joint currents of the manipulator
01448     *
01449     * @return Joint currents of the manipulator
01450     *
01451     * @throws arcs::common_interface::AuboException
01452     *
01453     * @par Python function prototype
01454     * getJointCurrents(self: pyaubo_sdk.RobotState) -> List[float]
01455     *
01456     * @par Lua function prototype
01457     * getJointCurrents() -> table
01458     *
01459     * @par Lua example
01460     * JointCurrents = getJointCurrents()
01461     *
01462     * @par JSON-RPC request example
01463     * {"jsonrpc":"2.0","method":"rob1.RobotState.getJointCurrents","params":[],"id":1}
01464     *
01465     * @par JSON-RPC response example
01466     * {"id":1,"jsonrpc":"2.0","result":[0.0,1.25885009765625,-
1.5289306640625,0.71868896484375,0.1007080078125,0.3021240234375]}
01467     * \endenglish
01468     */
01469     std::vector<double> getJointCurrents();
01470
01471     /**
01472     * @ingroup RobotState
01473     * \chinese
01474     *
01475     *
01476     * @return
01477     *
01478     * @throws arcs::common_interface::AuboException
01479     *
01480     * @par Python
01481     * getJointVoltages(self: pyaubo_sdk.RobotState) -> List[float]
01482     *
01483     * @par Lua
01484     * getJointVoltages() -> table
01485     *
01486     * @par Lua
01487     * JointVoltages = getJointVoltages()
01488     *
01489     * @par JSON-RPC
01490     * {"jsonrpc":"2.0","method":"rob1.RobotState.getJointVoltages","params":[],"id":1}
01491     *
01492     * @par JSON-RPC
01493     * {"id":1,"jsonrpc":"2.0","result":[2.0,2.5,3.0,2.0,2.5,2.0]}
01494     * \endchinese
01495     * \english
01496     * Get the joint voltages of the manipulator
01497     *
01498     * @return Joint voltages of the manipulator
01499     *
01500     * @throws arcs::common_interface::AuboException
01501     *
01502     * @par Python function prototype
01503     * getJointVoltages(self: pyaubo_sdk.RobotState) -> List[float]
01504     *
01505     * @par Lua function prototype
01506     * getJointVoltages() -> table
01507     *
01508     * @par Lua example

```


Generated by Doxygen

```

01681 *
01682 * @par Lua
01683 * JointHardwareVersions = getJointHardwareVersions()
01684 *
01685 * @par JSON-RPC
01686 * {"jsonrpc":"2.0","method":"rob1.RobotState.getJointHardwareVersions","params":[],"id":1}
01687 *
01688 * @par JSON-RPC
01689 * {"id":1,"jsonrpc":"2.0","result":[1000000,1000000,1004000,1004000,1004000,1004000]}
01690 * \endchinese
01691 * \english
01692 * Get the joint hardware versions
01693 *
01694 * @return Joint hardware versions
01695 *
01696 * @throws arcs::common_interface::AuboException
01697 *
01698 * @par Python function prototype
01699 * getJointHardwareVersions(self: pyaubo_sdk.RobotState) -> List[int]
01700 *
01701 * @par Lua function prototype
01702 * getJointHardwareVersions() -> table
01703 *
01704 * @par Lua example
01705 * JointHardwareVersions = getJointHardwareVersions()
01706 *
01707 * @par JSON-RPC request example
01708 * {"jsonrpc":"2.0","method":"rob1.RobotState.getJointHardwareVersions","params":[],"id":1}
01709 *
01710 * @par JSON-RPC response example
01711 * {"id":1,"jsonrpc":"2.0","result":[1000000,1000000,1004000,1004000,1004000,1004000]}
01712 * \endenglish
01713 */
01714 std::vector<int> getJointHardwareVersions();
01715
01716 /**
01717 * @ingroup RobotState
01718 * \chinese
01719 * MasterBoard ID
01720 *
01721 * @return MasterBoard ID
01722 *
01723 * @throws arcs::common_interface::AuboException
01724 *
01725 * @par Python
01726 * getMasterBoardUniqueId(self: pyaubo_sdk.RobotState) -> str
01727 *
01728 * @par Lua
01729 * getMasterBoardUniqueId() -> string
01730 *
01731 * @par Lua
01732 * MasterBoardUniqueId = getMasterBoardUniqueId()
01733 *
01734 * @par JSON-RPC
01735 * {"jsonrpc":"2.0","method":"rob1.RobotState.getMasterBoardUniqueId","params":[],"id":1}
01736 *
01737 * @par JSON-RPC
01738 * {"id":1,"jsonrpc":"2.0","result":"001e0044510f343037323637"}
01739 * \endchinese
01740 * \english
01741 * Get the globally unique ID of the MasterBoard
01742 *
01743 * @return Globally unique ID of the MasterBoard
01744 *
01745 * @throws arcs::common_interface::AuboException
01746 *
01747 * @par Python function prototype
01748 * getMasterBoardUniqueId(self: pyaubo_sdk.RobotState) -> str
01749 *
01750 * @par Lua function prototype
01751 * getMasterBoardUniqueId() -> string
01752 *
01753 * @par Lua example
01754 * MasterBoardUniqueId = getMasterBoardUniqueId()
01755 *
01756 * @par JSON-RPC request example
01757 * {"jsonrpc":"2.0","method":"rob1.RobotState.getMasterBoardUniqueId","params":[],"id":1}
01758 *
01759 * @par JSON-RPC response example
01760 * {"id":1,"jsonrpc":"2.0","result":"001e0044510f343037323637"}
01761 * \endenglish
01762 */
01763 std::string getMasterBoardUniqueId();
01764
01765 /**
01766 * @ingroup RobotState
01767 * \chinese

```

```

01768 * MasterBoard
01769 *
01770 * @return MasterBoard
01771 *
01772 * @throws arcs::common_interface::AuboException
01773 *
01774 * @par Python
01775 * getMasterBoardFirmwareVersion(self: pyaubo_sdk.RobotState) -> int
01776 *
01777 * @par Lua
01778 * getMasterBoardFirmwareVersion() -> number
01779 *
01780 * @par Lua
01781 * MasterBoardFirmwareVersion = getMasterBoardFirmwareVersion()
01782 *
01783 * @par JSON-RPC
01784 * {"jsonrpc":"2.0","method":"rob1.RobotState.getMasterBoardFirmwareVersion","params":[],"id":1}
01785 *
01786 * @par JSON-RPC
01787 * {"id":1,"jsonrpc":"2.0","result":1000004}
01788 * \endchinese
01789 * \english
01790 * Get the MasterBoard firmware version
01791 *
01792 * @return MasterBoard firmware version
01793 *
01794 * @throws arcs::common_interface::AuboException
01795 *
01796 * @par Python function prototype
01797 * getMasterBoardFirmwareVersion(self: pyaubo_sdk.RobotState) -> int
01798 *
01799 * @par Lua function prototype
01800 * getMasterBoardFirmwareVersion() -> number
01801 *
01802 * @par Lua example
01803 * MasterBoardFirmwareVersion = getMasterBoardFirmwareVersion()
01804 *
01805 * @par JSON-RPC request example
01806 * {"jsonrpc":"2.0","method":"rob1.RobotState.getMasterBoardFirmwareVersion","params":[],"id":1}
01807 *
01808 * @par JSON-RPC response example
01809 * {"id":1,"jsonrpc":"2.0","result":1000004}
01810 * \endenglish
01811 */
01812 int getMasterBoardFirmwareVersion();
01813
01814 /**
01815 * @ingroup RobotState
01816 * \chinese
01817 * MasterBoard
01818 *
01819 * @return MasterBoard
01820 *
01821 * @throws arcs::common_interface::AuboException
01822 *
01823 * @par Python
01824 * getMasterBoardHardwareVersion(self: pyaubo_sdk.RobotState) -> int
01825 *
01826 * @par Lua
01827 * getMasterBoardHardwareVersion() -> number
01828 *
01829 * @par Lua
01830 * MasterBoardHardwareVersion = getMasterBoardHardwareVersion()
01831 *
01832 * @par JSON-RPC
01833 * {"jsonrpc":"2.0","method":"rob1.RobotState.getMasterBoardHardwareVersion","params":[],"id":1}
01834 *
01835 * @par JSON-RPC
01836 * {"id":1,"jsonrpc":"2.0","result":1000000}
01837 * \endchinese
01838 * \english
01839 * Get the MasterBoard hardware version
01840 *
01841 * @return MasterBoard hardware version
01842 *
01843 * @throws arcs::common_interface::AuboException
01844 *
01845 * @par Python function prototype
01846 * getMasterBoardHardwareVersion(self: pyaubo_sdk.RobotState) -> int
01847 *
01848 * @par Lua function prototype
01849 * getMasterBoardHardwareVersion() -> number
01850 *
01851 * @par Lua example
01852 * MasterBoardHardwareVersion = getMasterBoardHardwareVersion()
01853 *
01854 * @par JSON-RPC request example

```

```

01855     * {"jsonrpc":"2.0","method":"rob1.RobotState.getMasterBoardHardwareVersion","params":[],"id":1}
01856     *
01857     * @par JSON-RPC response example
01858     * {"id":1,"jsonrpc":"2.0","result":1000000}
01859     * \endenglish
01860     */
01861 int getMasterBoardHardwareVersion();
01862
01863 /**
01864  * @ingroup RobotState
01865  * \chinese
01866  * SlaveBoard ID
01867  *
01868  * @return SlaveBoard ID
01869  *
01870  * @throws arcs::common_interface::AuboException
01871  *
01872  * @par Python
01873  * getSlaveBoardUniqueId(self: pyaubo_sdk.RobotState) -> str
01874  *
01875  * @par Lua
01876  * getSlaveBoardUniqueId() -> string
01877  *
01878  * @par Lua
01879  * SlaveBoardUniqueId = getSlaveBoardUniqueId()
01880  *
01881  * @par JSON-RPC
01882  * {"jsonrpc":"2.0","method":"rob1.RobotState.getSlaveBoardUniqueId","params":[],"id":1}
01883  *
01884  * @par JSON-RPC
01885  * {"id":1,"jsonrpc":"2.0","result":"73657263008000000000000000"}
01886  * \endchinese
01887  * \english
01888  * Get the globally unique ID of the SlaveBoard
01889  *
01890  * @return Globally unique ID of the SlaveBoard
01891  *
01892  * @throws arcs::common_interface::AuboException
01893  *
01894  * @par Python function prototype
01895  * getSlaveBoardUniqueId(self: pyaubo_sdk.RobotState) -> str
01896  *
01897  * @par Lua function prototype
01898  * getSlaveBoardUniqueId() -> string
01899  *
01900  * @par Lua example
01901  * SlaveBoardUniqueId = getSlaveBoardUniqueId()
01902  *
01903  * @par JSON-RPC request example
01904  * {"jsonrpc":"2.0","method":"rob1.RobotState.getSlaveBoardUniqueId","params":[],"id":1}
01905  *
01906  * @par JSON-RPC response example
01907  * {"id":1,"jsonrpc":"2.0","result":"73657263008000000000000000"}
01908  * \endenglish
01909  */
01910 std::string getSlaveBoardUniqueId();
01911
01912 /**
01913  * @ingroup RobotState
01914  * \chinese
01915  * SlaveBoard
01916  *
01917  * @return SlaveBoard
01918  *
01919  * @throws arcs::common_interface::AuboException
01920  *
01921  * @par Python
01922  * getSlaveBoardFirmwareVersion(self: pyaubo_sdk.RobotState) -> int
01923  *
01924  * @par Lua
01925  * getSlaveBoardFirmwareVersion() -> number
01926  *
01927  * @par Lua
01928  * SlaveBoardFirmwareVersion = getSlaveBoardFirmwareVersion()
01929  *
01930  * @par JSON-RPC
01931  * {"jsonrpc":"2.0","method":"rob1.RobotState.getSlaveBoardFirmwareVersion","params":[],"id":1}
01932  *
01933  * @par JSON-RPC
01934  * {"id":1,"jsonrpc":"2.0","result":0}
01935  * \endchinese
01936  * \english
01937  * Get the SlaveBoard firmware version
01938  *
01939  * @return SlaveBoard firmware version
01940  *
01941  * @throws arcs::common_interface::AuboException

```

```

01942 *
01943 * @par Python function prototype
01944 * getSlaveBoardFirmwareVersion(self: pyaubo_sdk.RobotState) -> int
01945 *
01946 * @par Lua function prototype
01947 * getSlaveBoardFirmwareVersion() -> number
01948 *
01949 * @par Lua example
01950 * SlaveBoardFirmwareVersion = getSlaveBoardFirmwareVersion()
01951 *
01952 * @par JSON-RPC request example
01953 * {"jsonrpc":"2.0","method":"rob1.RobotState.getSlaveBoardFirmwareVersion","params":[],"id":1}
01954 *
01955 * @par JSON-RPC response example
01956 * {"id":1,"jsonrpc":"2.0","result":0}
01957 * \endenglish
01958 */
01959 int getSlaveBoardFirmwareVersion();
01960
01961 /**
01962 * @ingroup RobotState
01963 * \chinese
01964 *   SlaveBoard
01965 *
01966 * @return SlaveBoard
01967 *
01968 * @throws arcs::common_interface::AuboException
01969 *
01970 * @par Python
01971 * getStateBoardHardwareVersion(self: pyaubo_sdk.RobotState) -> int
01972 *
01973 * @par Lua
01974 * getSlaveBoardHardwareVersion() -> number
01975 *
01976 * @par Lua
01977 * SlaveBoardHardwareVersion = getSlaveBoardHardwareVersion()
01978 *
01979 * @par JSON-RPC
01980 * {"jsonrpc":"2.0","method":"rob1.RobotState.getSlaveBoardHardwareVersion","params":[],"id":1}
01981 *
01982 * @par JSON-RPC
01983 * {"id":1,"jsonrpc":"2.0","result":6030098}
01984 * \endchinese
01985 * \english
01986 * Get the SlaveBoard hardware version
01987 *
01988 * @return SlaveBoard hardware version
01989 *
01990 * @throws arcs::common_interface::AuboException
01991 *
01992 * @par Python function prototype
01993 * getStateBoardHardwareVersion(self: pyaubo_sdk.RobotState) -> int
01994 *
01995 * @par Lua function prototype
01996 * getSlaveBoardHardwareVersion() -> number
01997 *
01998 * @par Lua example
01999 * SlaveBoardHardwareVersion = getSlaveBoardHardwareVersion()
02000 *
02001 * @par JSON-RPC request example
02002 * {"jsonrpc":"2.0","method":"rob1.RobotState.getStateBoardHardwareVersion","params":[],"id":1}
02003 *
02004 * @par JSON-RPC response example
02005 * {"id":1,"jsonrpc":"2.0","result":6030098}
02006 * \endenglish
02007 */
02008 int getStateBoardHardwareVersion();
02009
02010 /**
02011 * @ingroup RobotState
02012 * \chinese
02013 *   ID
02014 *
02015 * @return ID
02016 *
02017 * @throws arcs::common_interface::AuboException
02018 *
02019 * @par Python
02020 * getToolUniqueId(self: pyaubo_sdk.RobotState) -> str
02021 *
02022 * @par Lua
02023 * getToolUniqueId() -> string
02024 *
02025 * @par Lua
02026 * ToolUniqueId = getToolUniqueId()
02027 *
02028 * @par JSON-RPC

```

```

02029 * {"jsonrpc":"2.0","method":"rob1.RobotState.getToolUniqueId","params":[],"id":1}
02030 *
02031 * @par JSON-RPC
02032 * {"id":1,"jsonrpc":"2.0","result":"397d4e5331541252314d3042"}
02033 * \endchinese
02034 * \english
02035 * Get the globally unique ID of the tool
02036 *
02037 * @return Globally unique ID of the tool
02038 *
02039 * @throws arcs::common_interface::AuboException
02040 *
02041 * @par Python function prototype
02042 * getToolUniqueId(self: pyaubo_sdk.RobotState) -> str
02043 *
02044 * @par Lua function prototype
02045 * getToolUniqueId() -> string
02046 *
02047 * @par Lua example
02048 * ToolUniqueId = getToolUniqueId()
02049 *
02050 * @par JSON-RPC request example
02051 * {"jsonrpc":"2.0","method":"rob1.RobotState.getToolUniqueId","params":[],"id":1}
02052 *
02053 * @par JSON-RPC response example
02054 * {"id":1,"jsonrpc":"2.0","result":"397d4e5331541252314d3042"}
02055 * \endenglish
02056 */
02057 std::string getToolUniqueId();
02058
02059 /**
02060 * @ingroup RobotState
02061 * \chinese
02062 *
02063 *
02064 * @return
02065 *
02066 * @throws arcs::common_interface::AuboException
02067 *
02068 * @par Python
02069 * getToolFirmwareVersion(self: pyaubo_sdk.RobotState) -> int
02070 *
02071 * @par Lua
02072 * getToolFirmwareVersion() -> number
02073 *
02074 * @par Lua
02075 * ToolFirmwareVersion = getToolFirmwareVersion()
02076 *
02077 * @par JSON-RPC
02078 * {"jsonrpc":"2.0","method":"rob1.RobotState.getToolFirmwareVersion","params":[],"id":1}
02079 *
02080 * @par JSON-RPC
02081 * {"id":1,"jsonrpc":"2.0","result":1001003}
02082 * \endchinese
02083 * \english
02084 * Get the tool firmware version
02085 *
02086 * @return Tool firmware version
02087 *
02088 * @throws arcs::common_interface::AuboException
02089 *
02090 * @par Python function prototype
02091 * getToolFirmwareVersion(self: pyaubo_sdk.RobotState) -> int
02092 *
02093 * @par Lua function prototype
02094 * getToolFirmwareVersion() -> number
02095 *
02096 * @par Lua example
02097 * ToolFirmwareVersion = getToolFirmwareVersion()
02098 *
02099 * @par JSON-RPC request example
02100 * {"jsonrpc":"2.0","method":"rob1.RobotState.getToolFirmwareVersion","params":[],"id":1}
02101 *
02102 * @par JSON-RPC response example
02103 * {"id":1,"jsonrpc":"2.0","result":1001003}
02104 * \endenglish
02105 */
02106 int getToolFirmwareVersion();
02107
02108 /**
02109 * @ingroup RobotState
02110 * \chinese
02111 *
02112 *
02113 * @return
02114 *
02115 * @throws arcs::common_interface::AuboException

```



```

02116 *
02117 * @par Python
02118 * getToolHardwareVersion(self: pyaubo_sdk.RobotState) -> int
02119 *
02120 * @par Lua
02121 * getToolHardwareVersion() -> number
02122 *
02123 * @par Lua
02124 * ToolHardwareVersion = getToolHardwareVersion()
02125 *
02126 * @par JSON-RPC
02127 * {"jsonrpc":"2.0","method":"rob1.RobotState.getToolHardwareVersion","params":[],"id":1}
02128 *
02129 * @par JSON-RPC
02130 * {"id":1,"jsonrpc":"2.0","result":1000000}
02131 * \endchinese
02132 * \english
02133 * Get the tool hardware version
02134 *
02135 * @return Tool hardware version
02136 *
02137 * @throws arcs::common_interface::AuboException
02138 *
02139 * @par Python function prototype
02140 * getToolHardwareVersion(self: pyaubo_sdk.RobotState) -> int
02141 *
02142 * @par Lua function prototype
02143 * getToolHardwareVersion() -> number
02144 *
02145 * @par Lua example
02146 * ToolHardwareVersion = getToolHardwareVersion()
02147 *
02148 * @par JSON-RPC request example
02149 * {"jsonrpc":"2.0","method":"rob1.RobotState.getToolHardwareVersion","params":[],"id":1}
02150 *
02151 * @par JSON-RPC response example
02152 * {"id":1,"jsonrpc":"2.0","result":1000000}
02153 * \endenglish
02154 */
02155 int getToolHardwareVersion();
02156
02157 /**
02158 * @ingroup RobotState
02159 * \chinese
02160 *
02161 *
02162 * @return
02163 * 0:
02164 * 1:
02165 * 2:
02166 *
02167 * @throws arcs::common_interface::AuboException
02168 *
02169 * @par Python
02170 * getToolCommMode(self: pyaubo_sdk.RobotState) -> int
02171 *
02172 * @par Lua
02173 * getToolCommMode() -> number
02174 *
02175 * @par Lua
02176 * ToolCommMode = getToolCommMode()
02177 *
02178 * @par JSON-RPC
02179 * {"jsonrpc":"2.0","method":"rob1.RobotState.getToolCommMode","params":[],"id":1}
02180 *
02181 * @par JSON-RPC
02182 * {"id":1,"jsonrpc":"2.0","result":1}
02183 * \endchinese
02184 * \english
02185 * Get the tool communication mode
02186 *
02187 * @return Tool communication mode
02188 * 0: No serial port
02189 * 1: Serial port only
02190 * 2: With force sensor and serial port
02191 *
02192 * @throws arcs::common_interface::AuboException
02193 *
02194 * @par Python function prototype
02195 * getToolCommMode(self: pyaubo_sdk.RobotState) -> int
02196 *
02197 * @par Lua function prototype
02198 * getToolCommMode() -> number
02199 *
02200 * @par Lua example
02201 * ToolCommMode = getToolCommMode()
02202 *

```

```

02203 * @par JSON-RPC request example
02204 * {"jsonrpc":"2.0","method":"rob1.RobotState.getToolCommMode","params":[],"id":1}
02205 *
02206 * @par JSON-RPC response example
02207 * {"id":1,"jsonrpc":"2.0","result":1}
02208 * \endenglish
02209 */
02210 int getToolCommMode();
02211
02212 /**
02213 * @ingroup RobotState
02214 * \chinese
02215 * ID
02216 *
02217 * @return ID
02218 *
02219 * @throws arcs::common_interface::AuboException
02220 *
02221 * @par Python
02222 * getPedestalUniqueId(self: pyaubo_sdk.RobotState) -> str
02223 *
02224 * @par Lua
02225 * getPedestalUniqueId() -> string
02226 *
02227 * @par Lua
02228 * PedestalUniqueId = getPedestalUniqueId()
02229 *
02230 * @par JSON-RPC
02231 * {"jsonrpc":"2.0","method":"rob1.RobotState.getPedestalUniqueId","params":[],"id":1}
02232 *
02233 * @par JSON-RPC
02234 * {"id":1,"jsonrpc":"2.0","result":"205257533543065248544339"}
02235 * \endchinese
02236 * \english
02237 * Get the globally unique ID of the pedestal
02238 *
02239 * @return Globally unique ID of the pedestal
02240 *
02241 * @throws arcs::common_interface::AuboException
02242 *
02243 * @par Python function prototype
02244 * getPedestalUniqueId(self: pyaubo_sdk.RobotState) -> str
02245 *
02246 * @par Lua function prototype
02247 * getPedestalUniqueId() -> string
02248 *
02249 * @par Lua example
02250 * PedestalUniqueId = getPedestalUniqueId()
02251 *
02252 * @par JSON-RPC request example
02253 * {"jsonrpc":"2.0","method":"rob1.RobotState.getPedestalUniqueId","params":[],"id":1}
02254 *
02255 * @par JSON-RPC response example
02256 * {"id":1,"jsonrpc":"2.0","result":"205257533543065248544339"}
02257 * \endenglish
02258 */
02259 std::string getPedestalUniqueId();
02260
02261 /**
02262 * @ingroup RobotState
02263 * \chinese
02264 *
02265 *
02266 * @return
02267 *
02268 * @throws arcs::common_interface::AuboException
02269 *
02270 * @par Python
02271 * getPedestalFirmwareVersion(self: pyaubo_sdk.RobotState) -> int
02272 *
02273 * @par Lua
02274 * getPedestalFirmwareVersion() -> number
02275 *
02276 * @par Lua
02277 * PedestalFirmwareVersion = getPedestalFirmwareVersion()
02278 *
02279 * @par JSON-RPC
02280 * {"jsonrpc":"2.0","method":"rob1.RobotState.getPedestalFirmwareVersion","params":[],"id":1}
02281 *
02282 * @par JSON-RPC
02283 * {"id":1,"jsonrpc":"2.0","result":1000004}
02284 * \endchinese
02285 * \english
02286 * Get the pedestal firmware version
02287 *
02288 * @return Pedestal firmware version
02289 *

```

```

02290     * @throws arcs::common_interface::AuboException
02291     *
02292     * @par Python function prototype
02293     * getPedestalFirmwareVersion(self: pyaubo_sdk.RobotState) -> int
02294     *
02295     * @par Lua function prototype
02296     * getPedestalFirmwareVersion() -> number
02297     *
02298     * @par Lua example
02299     * PedestalFirmwareVersion = getPedestalFirmwareVersion()
02300     *
02301     * @par JSON-RPC request example
02302     * {"jsonrpc":"2.0","method":"rob1.RobotState.getPedestalFirmwareVersion","params":[],"id":1}
02303     *
02304     * @par JSON-RPC response example
02305     * {"id":1,"jsonrpc":"2.0","result":1000004}
02306     * \endenglish
02307     */
02308 int getPedestalFirmwareVersion();
02309
02310 /**
02311  * @ingroup RobotState
02312  * \chinese
02313  *
02314  *
02315  * @return
02316  *
02317  * @throws arcs::common_interface::AuboException
02318  *
02319  * @par Python
02320  * getPedestalHardwareVersion(self: pyaubo_sdk.RobotState) -> int
02321  *
02322  * @par Lua
02323  * getPedestalHardwareVersion() -> number
02324  *
02325  * @par Lua
02326  * PedestalHardwareVersion = getPedestalHardwareVersion()
02327  *
02328  * @par JSON-RPC
02329  * {"jsonrpc":"2.0","method":"rob1.RobotState.getPedestalHardwareVersion","params":[],"id":1}
02330  *
02331  * @par JSON-RPC
02332  * {"id":1,"jsonrpc":"2.0","result":1007000}
02333  * \endchinese
02334  * \english
02335  * Get the pedestal hardware version
02336  *
02337  * @return Pedestal hardware version
02338  *
02339  * @throws arcs::common_interface::AuboException
02340  *
02341  * @par Python function prototype
02342  * getPedestalHardwareVersion(self: pyaubo_sdk.RobotState) -> int
02343  *
02344  * @par Lua function prototype
02345  * getPedestalHardwareVersion() -> number
02346  *
02347  * @par Lua example
02348  * PedestalHardwareVersion = getPedestalHardwareVersion()
02349  *
02350  * @par JSON-RPC request example
02351  * {"jsonrpc":"2.0","method":"rob1.RobotState.getPedestalHardwareVersion","params":[],"id":1}
02352  *
02353  * @par JSON-RPC response example
02354  * {"id":1,"jsonrpc":"2.0","result":1007000}
02355  * \endenglish
02356  */
02357 int getPedestalHardwareVersion();
02358
02359 /**
02360  * @ingroup RobotState
02361  * \chinese
02362  *
02363  *
02364  * @return
02365  *
02366  * @throws arcs::common_interface::AuboException
02367  *
02368  * @par Python
02369  * getJointTargetPositions(self: pyaubo_sdk.RobotState) -> List[float]
02370  *
02371  * @par Lua
02372  * getJointTargetPositions() -> table
02373  *
02374  * @par Lua
02375  * JointTargetPositions = getJointTargetPositions()
02376  *

```

```

02377 * @par JSON-RPC
02378 * {"jsonrpc":"2.0","method":"rob1.RobotState.getJointTargetPositions","params":[],"id":1}
02379 *
02380 * @par JSON-RPC
02381 * {"id":1,"jsonrpc":"2.0","result":[0.0,-
0.2619944355631239,1.7418124015308052,0.4330219266665035,1.5707963267948966,0.0]}
02382 * \endchinese
02383 * \english
02384 * Get the target joint positions (angles) of the manipulator
02385 *
02386 * @return Target joint positions (angles) of the manipulator
02387 *
02388 * @throws arcs::common_interface::AuboException
02389 *
02390 * @par Python function prototype
02391 * getJointTargetPositions(self: pyaubo_sdk.RobotState) -> List[float]
02392 *
02393 * @par Lua function prototype
02394 * getJointTargetPositions() -> table
02395 *
02396 * @par Lua example
02397 * JointTargetPositions = getJointTargetPositions()
02398 *
02399 * @par JSON-RPC request example
02400 * {"jsonrpc":"2.0","method":"rob1.RobotState.getJointTargetPositions","params":[],"id":1}
02401 *
02402 * @par JSON-RPC response example
02403 * {"id":1,"jsonrpc":"2.0","result":[0.0,-
0.2619944355631239,1.7418124015308052,0.4330219266665035,1.5707963267948966,0.0]}
02404 * \endenglish
02405 */
02406 std::vector<double> getJointTargetPositions();
02407
02408 /**
02409 * @ingroup RobotState
02410 * \chinese
02411 *
02412 *
02413 * @return
02414 *
02415 * @throws arcs::common_interface::AuboException
02416 *
02417 * @par Python
02418 * getJointTargetSpeeds(self: pyaubo_sdk.RobotState) -> List[float]
02419 *
02420 * @par Lua
02421 * getJointTargetSpeeds() -> table
02422 *
02423 * @par Lua
02424 * JointTargetSpeeds = getJointTargetSpeeds()
02425 *
02426 * @par JSON-RPC
02427 * {"jsonrpc":"2.0","method":"rob1.RobotState.getJointTargetSpeeds","params":[],"id":1}
02428 *
02429 * @par JSON-RPC
02430 * {"id":1,"jsonrpc":"2.0","result":[0.0,0.00024227101509399773,0.0016521760307419697,0.0026521060731088397,0.0,0.0]}
02431 * \endchinese
02432 * \english
02433 * Get the target joint speeds of the manipulator
02434 *
02435 * @return Target joint speeds of the manipulator
02436 *
02437 * @throws arcs::common_interface::AuboException
02438 *
02439 * @par Python function prototype
02440 * getJointTargetSpeeds(self: pyaubo_sdk.RobotState) -> List[float]
02441 *
02442 * @par Lua function prototype
02443 * getJointTargetSpeeds() -> table
02444 *
02445 * @par Lua example
02446 * JointTargetSpeeds = getJointTargetSpeeds()
02447 *
02448 * @par JSON-RPC request example
02449 * {"jsonrpc":"2.0","method":"rob1.RobotState.getJointTargetSpeeds","params":[],"id":1}
02450 *
02451 * @par JSON-RPC response example
02452 * {"id":1,"jsonrpc":"2.0","result":[0.0,0.00024227101509399773,0.0016521760307419697,0.0026521060731088397,0.0,0.0]}
02453 * \endenglish
02454 */
02455 std::vector<double> getJointTargetSpeeds();
02456
02457 /**
02458 * @ingroup RobotState
02459 * \chinese
02460 *
02461 *

```

```

02462 * @return
02463 *
02464 * @throws arcs::common_interface::AuboException
02465 *
02466 * @par Python
02467 * getJointTargetAccelerations(self: pyaubo_sdk.RobotState) -> List[float]
02468 *
02469 * @par Lua
02470 * getJointTargetAccelerations() -> table
02471 *
02472 * @par Lua
02473 * JointTargetAccelerations = getJointTargetAccelerations()
02474 *
02475 * @par JSON-RPC
02476 * {"jsonrpc":"2.0","method":"rob1.RobotState.getJointTargetAccelerations","params":[],"id":1}
02477 *
02478 * @par JSON-RPC
02479 * {"id":1,"jsonrpc":"2.0","result":[0.0,-0.6737932929246071,-12.610253240108449,0.0,0.0,0.0]}
02480 * \endchinese
02481 * \english
02482 * Get the target joint accelerations of the manipulator
02483 *
02484 * @return Target joint accelerations of the manipulator
02485 *
02486 * @throws arcs::common_interface::AuboException
02487 *
02488 * @par Python function prototype
02489 * getJointTargetAccelerations(self: pyaubo_sdk.RobotState) -> List[float]
02490 *
02491 * @par Lua function prototype
02492 * getJointTargetAccelerations() -> table
02493 *
02494 * @par Lua example
02495 * JointTargetAccelerations = getJointTargetAccelerations()
02496 *
02497 * @par JSON-RPC request example
02498 * {"jsonrpc":"2.0","method":"rob1.RobotState.getJointTargetAccelerations","params":[],"id":1}
02499 *
02500 * @par JSON-RPC response example
02501 * {"id":1,"jsonrpc":"2.0","result":[0.0,-0.6737932929246071,-12.610253240108449,0.0,0.0,0.0]}
02502 * \endenglish
02503 */
02504 std::vector<double> getJointTargetAccelerations();
02505
02506 /**
02507 * @ingroup RobotState
02508 * \chinese
02509 *
02510 *
02511 * @return
02512 *
02513 * @throws arcs::common_interface::AuboException
02514 *
02515 * @par Python
02516 * getJointTargetTorques(self: pyaubo_sdk.RobotState) -> List[float]
02517 *
02518 * @par Lua
02519 * getJointTargetTorques() -> table
02520 *
02521 * @par Lua
02522 * JointTargetTorques = getJointTargetTorques()
02523 *
02524 * @par JSON-RPC
02525 * {"jsonrpc":"2.0","method":"rob1.RobotState.getJointTargetTorques","params":[],"id":1}
02526 *
02527 * @par JSON-RPC
02528 * {"id":1,"jsonrpc":"2.0","result":[0.0,0.0,0.0,0.0,0.0,0.0]}
02529 * \endchinese
02530 * \english
02531 * Get the target joint torques of the manipulator
02532 *
02533 * @return Target joint torques of the manipulator
02534 *
02535 * @throws arcs::common_interface::AuboException
02536 *
02537 * @par Python function prototype
02538 * getJointTargetTorques(self: pyaubo_sdk.RobotState) -> List[float]
02539 *
02540 * @par Lua function prototype
02541 * getJointTargetTorques() -> table
02542 *
02543 * @par Lua example
02544 * JointTargetTorques = getJointTargetTorques()
02545 *
02546 * @par JSON-RPC request example
02547 * {"jsonrpc":"2.0","method":"rob1.RobotState.getJointTargetTorques","params":[],"id":1}
02548 *

```

```

02549     * @par JSON-RPC response example
02550     * {"id":1,"jsonrpc":"2.0","result":[0.0,0.0,0.0,0.0,0.0,0.0]}
02551     * \endenglish
02552     */
02553     std::vector<double> getJointTargetTorques();
02554
02555     /**
02556     * @ingroup RobotState
02557     * \chinese
02558     *
02559     *
02560     * @return
02561     *
02562     * @throws arcs::common_interface::AuboException
02563     *
02564     * @par Python
02565     * getJointTargetCurrents(self: pyaubo_sdk.RobotState) -> List[float]
02566     *
02567     * @par Lua
02568     * getJointTargetCurrents() -> table
02569     *
02570     * @par Lua
02571     * JointTargetCurrents = getJointTargetCurrents()
02572     *
02573     * @par JSON-RPC
02574     * {"jsonrpc":"2.0","method":"rob1.RobotState.getJointTargetCurrents","params":[],"id":1}
02575     *
02576     * @par JSON-RPC
02577     * {"id":1,"jsonrpc":"2.0","result":[0.0,0.0,0.0,0.0,0.0,0.0]}
02578     * \endchinese
02579     * \english
02580     * Get the target joint currents of the manipulator
02581     *
02582     * @return Target joint currents of the manipulator
02583     *
02584     * @throws arcs::common_interface::AuboException
02585     *
02586     * @par Python function prototype
02587     * getJointTargetCurrents(self: pyaubo_sdk.RobotState) -> List[float]
02588     *
02589     * @par Lua function prototype
02590     * getJointTargetCurrents() -> table
02591     *
02592     * @par Lua example
02593     * JointTargetCurrents = getJointTargetCurrents()
02594     *
02595     * @par JSON-RPC request example
02596     * {"jsonrpc":"2.0","method":"rob1.RobotState.getJointTargetCurrents","params":[],"id":1}
02597     *
02598     * @par JSON-RPC response example
02599     * {"id":1,"jsonrpc":"2.0","result":[0.0,0.0,0.0,0.0,0.0,0.0]}
02600     * \endenglish
02601     */
02602     std::vector<double> getJointTargetCurrents();
02603
02604     /**
02605     * @ingroup RobotState
02606     * \chinese
02607     *
02608     *
02609     *
02610     * @return          false;    true
02611     *
02612     * @throws arcs::common_interface::AuboException
02613     *
02614     * @par Python
02615     * isTeachPendantEnabled(self: pyaubo_sdk.RobotState) -> bool
02616     *
02617     * @par Lua
02618     * isTeachPendantEnabled() -> boolean
02619     *
02620     * @par Lua
02621     * TeachPendantEnabled = isTeachPendantEnabled()
02622     *
02623     * @par JSON-RPC
02624     * {"jsonrpc":"2.0","method":"rob1.RobotState.isTeachPendantEnabled","params":[],"id":1}
02625     *
02626     * @par JSON-RPC
02627     * {"id":1,"jsonrpc":"2.0","result":true}
02628     * \endchinese
02629     * \english
02630     * Get whether the teach pendant is enabled.
02631     * Indicates whether the enable button of teach pendant is in the pressed
02632     * state.
02633     *
02634     * @return Returns false if the enable button of teach pendant is pressed;
02635     * otherwise returns true

```

```

02636  *
02637  * @throws arcs::common_interface::AuboException
02638  *
02639  * @par Python function prototype
02640  * isTeachPendantEnabled(self: pyaubo_sdk.RobotState) -> bool
02641  *
02642  * @par Lua function prototype
02643  * isTeachPendantEnabled() -> boolean
02644  *
02645  * @par Lua example
02646  * TeachPendantEnabled = isTeachPendantEnabled()
02647  *
02648  * @par JSON-RPC request example
02649  * {"jsonrpc":"2.0","method":"rob1.RobotState.isTeachPendantEnabled","params":[],"id":1}
02650  *
02651  * @par JSON-RPC response example
02652  * {"id":1,"jsonrpc":"2.0","result":true}
02653  * \endenglish
02654  */
02655 bool isTeachPendantEnabled();
02656
02657 /**
02658  * @ingroup RobotState
02659  * \chinese
02660  *
02661  *
02662  * @return false; true
02663  *
02664  * @throws arcs::common_interface::AuboException
02665  *
02666  * @par Python
02667  * isToolFlangeEnabled(self: pyaubo_sdk.RobotState) -> bool
02668  *
02669  * @par Lua
02670  * isToolFlangeEnabled() -> boolean
02671  *
02672  * @par Lua
02673  * toolEnabled = isToolFlangeEnabled()
02674  *
02675  * @par JSON-RPC
02676  * {"jsonrpc":"2.0","method":"rob1.RobotState.isToolFlangeEnabled","params":[],"id":1}
02677  *
02678  * @par JSON-RPC
02679  * {"id":1,"jsonrpc":"2.0","result":true}
02680  * \endchinese
02681  * \english
02682  * Get whether the tool flange is enabled.
02683  *
02684  * @return Returns false if the tool flange is disabled; otherwise returns
02685  * true
02686  *
02687  * @throws arcs::common_interface::AuboException
02688  *
02689  * @par Python function prototype
02690  * isToolFlangeEnabled(self: pyaubo_sdk.RobotState) -> bool
02691  *
02692  * @par Lua function prototype
02693  * isToolFlangeEnabled() -> boolean
02694  *
02695  * @par Lua example
02696  * toolEnabled = isToolFlangeEnabled()
02697  *
02698  * @par JSON-RPC request example
02699  * {"jsonrpc":"2.0","method":"rob1.RobotState.isToolFlangeEnabled","params":[],"id":1}
02700  *
02701  * @par JSON-RPC response example
02702  * {"id":1,"jsonrpc":"2.0","result":true}
02703  * \endenglish
02704  */
02705 bool isToolFlangeEnabled();
02706
02707 /**
02708  * @ingroup RobotState
02709  * \chinese
02710  *
02711  *
02712  * @return
02713  *
02714  * @throws arcs::common_interface::AuboException
02715  *
02716  * @par Python
02717  * getControlBoxTemperature(self: pyaubo_sdk.RobotState) -> float
02718  *
02719  * @par Lua
02720  * getControlBoxTemperature() -> number
02721  *
02722  * @par Lua

```

```

02723 * ControlBoxTemperature = getControlBoxTemperature()
02724 *
02725 * @par JSON-RPC
02726 * {"jsonrpc":"2.0","method":"rob1.RobotState.getControlBoxTemperature","params":[],"id":1}
02727 *
02728 * @par JSON-RPC
02729 * {"id":1,"jsonrpc":"2.0","result":25.0}
02730 * \endchinese
02731 * \english
02732 * Get the control box temperature
02733 *
02734 * @return Control box temperature
02735 *
02736 * @throws arcs::common_interface::AuboException
02737 *
02738 * @par Python function prototype
02739 * getControlBoxTemperature(self: pyaubo_sdk.RobotState) -> float
02740 *
02741 * @par Lua function prototype
02742 * getControlBoxTemperature() -> number
02743 *
02744 * @par Lua example
02745 * ControlBoxTemperature = getControlBoxTemperature()
02746 *
02747 * @par JSON-RPC request example
02748 * {"jsonrpc":"2.0","method":"rob1.RobotState.getControlBoxTemperature","params":[],"id":1}
02749 *
02750 * @par JSON-RPC response example
02751 * {"id":1,"jsonrpc":"2.0","result":25.0}
02752 * \endenglish
02753 */
02754 double getControlBoxTemperature();
02755
02756 /**
02757 * @ingroup RobotState
02758 * \chinese
02759 *
02760 *
02761 * @return
02762 *
02763 * @throws arcs::common_interface::AuboException
02764 *
02765 * @par Python
02766 * getControlBoxHumidity(self: pyaubo_sdk.RobotState) -> float
02767 *
02768 * @par Lua
02769 * getControlBoxHumidity() -> number
02770 *
02771 * @par Lua
02772 * ControlBoxHumidity = getControlBoxHumidity()
02773 *
02774 * @par JSON-RPC
02775 * {"jsonrpc":"2.0","method":"rob1.RobotState.getControlBoxHumidity","params":[],"id":1}
02776 *
02777 * @par JSON-RPC
02778 * {"id":1,"jsonrpc":"2.0","result":20.0}
02779 * \endchinese
02780 * \english
02781 * Get the control box humidity
02782 *
02783 * @return Control box humidity
02784 *
02785 * @throws arcs::common_interface::AuboException
02786 *
02787 * @par Python function prototype
02788 * getControlBoxHumidity(self: pyaubo_sdk.RobotState) -> float
02789 *
02790 * @par Lua function prototype
02791 * getControlBoxHumidity() -> number
02792 *
02793 * @par Lua example
02794 * ControlBoxHumidity = getControlBoxHumidity()
02795 *
02796 * @par JSON-RPC request example
02797 * {"jsonrpc":"2.0","method":"rob1.RobotState.getControlBoxHumidity","params":[],"id":1}
02798 *
02799 * @par JSON-RPC response example
02800 * {"id":1,"jsonrpc":"2.0","result":20.0}
02801 * \endenglish
02802 */
02803 double getControlBoxHumidity();
02804
02805 /**
02806 * @ingroup RobotState
02807 * \chinese
02808 *
02809 *

```



```

02810     * @return
02811     *
02812     * @throws arcs::common_interface::AuboException
02813     *
02814     * @par Python
02815     * getMainVoltage(self: pyaubo_sdk.RobotState) -> float
02816     *
02817     * @par Lua
02818     * getMainVoltage() -> number
02819     *
02820     * @par Lua
02821     * MainVoltage = getMainVoltage()
02822     *
02823     * @par JSON-RPC
02824     * {"jsonrpc":"2.0","method":"rob1.RobotState.getMainVoltage","params":[],"id":1}
02825     *
02826     * @par JSON-RPC
02827     * {"id":1,"jsonrpc":"2.0","result":52.75}
02828     * \endchinese
02829     * \english
02830     * Get the main bus voltage
02831     *
02832     * @return Main bus voltage
02833     *
02834     * @throws arcs::common_interface::AuboException
02835     *
02836     * @par Python function prototype
02837     * getMainVoltage(self: pyaubo_sdk.RobotState) -> float
02838     *
02839     * @par Lua function prototype
02840     * getMainVoltage() -> number
02841     *
02842     * @par Lua example
02843     * MainVoltage = getMainVoltage()
02844     *
02845     * @par JSON-RPC request example
02846     * {"jsonrpc":"2.0","method":"rob1.RobotState.getMainVoltage","params":[],"id":1}
02847     *
02848     * @par JSON-RPC response example
02849     * {"id":1,"jsonrpc":"2.0","result":52.75}
02850     * \endenglish
02851     */
02852 double getMainVoltage();
02853
02854 /**
02855  * @ingroup RobotState
02856  * \chinese
02857  *
02858  *
02859  * @return
02860  *
02861  * @throws arcs::common_interface::AuboException
02862  *
02863  * @par Python
02864  * getMainCurrent(self: pyaubo_sdk.RobotState) -> float
02865  *
02866  * @par Lua
02867  * getMainCurrent() -> number
02868  *
02869  * @par Lua
02870  * MainCurrent = getMainCurrent()
02871  *
02872  * @par JSON-RPC
02873  * {"jsonrpc":"2.0","method":"rob1.RobotState.getMainCurrent","params":[],"id":1}
02874  *
02875  * @par JSON-RPC
02876  * {"id":1,"jsonrpc":"2.0","result":0.3204345703125}
02877  * \endchinese
02878  * \english
02879  * Get the main bus current
02880  *
02881  * @return Main bus current
02882  *
02883  * @throws arcs::common_interface::AuboException
02884  *
02885  * @par Python function prototype
02886  * getMainCurrent(self: pyaubo_sdk.RobotState) -> float
02887  *
02888  * @par Lua function prototype
02889  * getMainCurrent() -> number
02890  *
02891  * @par Lua example
02892  * MainCurrent = getMainCurrent()
02893  *
02894  * @par JSON-RPC request example
02895  * {"jsonrpc":"2.0","method":"rob1.RobotState.getMainCurrent","params":[],"id":1}
02896  *

```

```

02897 * @par JSON-RPC response example
02898 * {"id":1,"jsonrpc":"2.0","result":0.3204345703125}
02899 * \endenglish
02900 */
02901 double getMainCurrent();
02902
02903 /**
02904 * @ingroup RobotState
02905 * \chinese
02906 *
02907 *
02908 * @return
02909 *
02910 * @throws arcs::common_interface::AuboException
02911 *
02912 * @par Python
02913 * getRobotVoltage(self: pyaubo_sdk.RobotState) -> float
02914 *
02915 * @par Lua
02916 * getRobotVoltage() -> number
02917 *
02918 * @par Lua
02919 * RobotVoltage = getRobotVoltage()
02920 *
02921 * @par JSON-RPC
02922 * {"jsonrpc":"2.0","method":"rob1.RobotState.getRobotVoltage","params":[],"id":1}
02923 *
02924 * @par JSON-RPC
02925 * {"id":1,"jsonrpc":"2.0","result":52.75}
02926 * \endchinese
02927 * \english
02928 * Get the robot voltage
02929 *
02930 * @return Robot voltage
02931 *
02932 * @throws arcs::common_interface::AuboException
02933 *
02934 * @par Python function prototype
02935 * getRobotVoltage(self: pyaubo_sdk.RobotState) -> float
02936 *
02937 * @par Lua function prototype
02938 * getRobotVoltage() -> number
02939 *
02940 * @par Lua example
02941 * RobotVoltage = getRobotVoltage()
02942 *
02943 * @par JSON-RPC request example
02944 * {"jsonrpc":"2.0","method":"rob1.RobotState.getRobotVoltage","params":[],"id":1}
02945 *
02946 * @par JSON-RPC response example
02947 * {"id":1,"jsonrpc":"2.0","result":52.75}
02948 * \endenglish
02949 */
02950 double getRobotVoltage();
02951
02952 /**
02953 * @ingroup RobotState
02954 * \chinese
02955 *
02956 *
02957 * @return
02958 *
02959 * @throws arcs::common_interface::AuboException
02960 *
02961 * @par Python
02962 * getRobotCurrent(self: pyaubo_sdk.RobotState) -> float
02963 *
02964 * @par Lua
02965 * getRobotCurrent() -> number
02966 *
02967 * @par Lua
02968 * RobotCurrent = getRobotCurrent()
02969 *
02970 * @par JSON-RPC
02971 * {"jsonrpc":"2.0","method":"rob1.RobotState.getRobotCurrent","params":[],"id":1}
02972 *
02973 * @par JSON-RPC
02974 * {"id":1,"jsonrpc":"2.0","result":0.3204345703125}
02975 * \endchinese
02976 * \english
02977 * Get the robot current
02978 *
02979 * @return Robot current
02980 *
02981 * @throws arcs::common_interface::AuboException
02982 *
02983 * @par Python function prototype

```

```

02984 * getRobotCurrent(self: pyaubo_sdk.RobotState) -> float
02985 *
02986 * @par Lua function prototype
02987 * getRobotCurrent() -> number
02988 *
02989 * @par Lua example
02990 * RobotCurrent = getRobotCurrent()
02991 *
02992 * @par JSON-RPC request example
02993 * {"jsonrpc":"2.0","method":"rob1.RobotState.getRobotCurrent","params":[],"id":1}
02994 *
02995 * @par JSON-RPC response example
02996 * {"id":1,"jsonrpc":"2.0","result":0.3204345703125}
02997 * \endenglish
02998 */
02999 double getRobotCurrent();
03000
03001 /**
03002 * @ingroup RobotState
03003 * \chinese
03004 *
03005 *
03006 * @return
03007 *
03008 * @throws arcs::common_interface::AuboException
03009 *
03010 * @par JSON-RPC
03011 * {"jsonrpc":"2.0","method":"rob1.RobotState.getSlowDownLevel","params":[],"id":1}
03012 *
03013 * @par JSON-RPC
03014 * {"id":1,"jsonrpc":"2.0","result":0}
03015 * \endchinese
03016 * \english
03017 * Get the robot slow down level
03018 *
03019 * @return Robot slow down level
03020 *
03021 * @throws arcs::common_interface::AuboException
03022 *
03023 * @par JSON-RPC request example
03024 * {"jsonrpc":"2.0","method":"rob1.RobotState.getSlowDownLevel","params":[],"id":1}
03025 *
03026 * @par JSON-RPC response example
03027 * {"id":1,"jsonrpc":"2.0","result":0}
03028 * \endenglish
03029 */
03030 int getSlowDownLevel();
03031
03032 /**
03033 * @ingroup RobotState
03034 * \chinese
03035 *
03036 *
03037 * @param name      selectTcpForceSensor
03038 * @return          true;      false
03039 *
03040 * @throws arcs::common_interface::AuboException
03041 *
03042 * @par JSON-RPC
03043 * {"jsonrpc":"2.0","method":"rob1.RobotState.getTcpForceSensorStatus","params":["tool.KWR75A"],"id":1}
03044 *
03045 * @par JSON-RPC
03046 * {"id":1,"jsonrpc":"2.0","result":0}
03047 * \endchinese
03048 * \english
03049 * Get the communication status of the tool force sensor
03050 *
03051 * @param name      force sensor name, it is consistent with the parameters of
03052 * 'selectTcpForceSensor'
03053 * @return Returns true if communication is normal; otherwise returns false
03054 *
03055 * @throws arcs::common_interface::AuboException
03056 *
03057 * @par JSON-RPC request example
03058 * {"jsonrpc":"2.0","method":"rob1.RobotState.getSlowDownLevel","params":["tool.KWR75A"],"id":1}
03059 *
03060 * @par JSON-RPC response example
03061 * {"id":1,"jsonrpc":"2.0","result":0}
03062 * \endenglish
03063 */
03064 bool getTcpForceSensorStatus(const std::string &name);
03065
03066 protected:
03067 void *d_;
03068 };
03069 using RobotStatePtr = std::shared_ptr<RobotState>;
03070 } // namespace common_interface

```


Typedefs

- using `arcs::common_interface::RobotInterfacePtr` = `std::shared_ptr<RobotInterface>`

12.34.1 Detailed Description

API

Definition in file `robot_interface.h`.

12.35 robot_interface.h

[Go to the documentation of this file.](#)

```

00001 /** @file robot_interface.h
00002  * @brief API
00003  */
00004 #ifndef AUBO_SDK_ROBOT_INTERFACE_H
00005 #define AUBO_SDK_ROBOT_INTERFACE_H
00006
00007 #include <aubo/sync_move.h>
00008 #include <aubo/trace.h>
00009 #include <aubo/robot/motion_control.h>
00010 #include <aubo/robot/force_control.h>
00011 #include <aubo/robot/io_control.h>
00012 #include <aubo/robot/robot_algorithm.h>
00013 #include <aubo/robot/robot_state.h>
00014 #include <aubo/robot/robot_manage.h>
00015 #include <aubo/robot/robot_config.h>
00016 #include <aubo/global_config.h>
00017
00018 namespace arcs {
00019 namespace common_interface {
00020
00021 /**
00022  * @defgroup RobotInterface RobotInterface( )
00023  * \~chinese \~english RobotInterface
00024  */
00025 class ARCS_ABI_EXPORT RobotInterface
00026 {
00027 public:
00028     RobotInterface();
00029     virtual ~RobotInterface();
00030
00031     /**
00032      * @ingroup RobotInterface
00033      * @ref RobotConfig
00034      * \chinese
00035      * RobotConfig
00036      *
00037      * @return RobotConfigPtr
00038      *
00039      * @par Python
00040      * getRobotConfig(self: pyaubo_sdk.RobotInterface) ->
00041      * arcs::common_interface::RobotConfig
00042      *
00043      * @par C++
00044      * @code
00045      * auto rpc_cli = std::make_shared<RpcClient>();
00046      * auto robot_name = rpc_cli->getRobotNames().front();
00047      * RobotConfigPtr ptr =
00048      * rpc_cli->getRobotInterface(robot_name)->getRobotConfig();
00049      * @endcode
00050      * \endchinese
00051      * \english
00052      * Get RobotConfig interface
00053      *
00054      * @return Pointer to RobotConfig object
00055      *
00056      * @par Python function prototype
00057      * getRobotConfig(self: pyaubo_sdk.RobotInterface) ->
00058      * arcs::common_interface::RobotConfig
00059      *
00060      * @par C++ example
00061      * @code

```

```

00062 * auto rpc_cli = std::make_shared<RpcClient>();
00063 * auto robot_name = rpc_cli->getRobotNames().front();
00064 * RobotConfigPtr ptr =
00065 * rpc_cli->getRobotInterface(robot_name)->getRobotConfig();
00066 * @endcode
00067 * \endenglish
00068 */
00069
00070 RobotConfigPtr getRobotConfig();
00071
00072 /**
00073 * @ingroup RobotInterface
00074 * @ref MotionControl
00075 * \chinese
00076 *
00077 *
00078 * @return MotionControlPtr
00079 *
00080 * @par Python
00081 * getMotionControl(self: pyaubo_sdk.RobotInterface) ->
00082 * arcs::common_interface::MotionControl
00083 *
00084 * @par C++
00085 * @code
00086 * auto rpc_cli = std::make_shared<RpcClient>();
00087 * auto robot_name = rpc_cli->getRobotNames().front();
00088 * MotionControlPtr ptr =
00089 * rpc_cli->getRobotInterface(robot_name)->getMotionControl();
00090 * @endcode
00091 * \endchinese
00092 * \english
00093 * Get motion planning interface
00094 *
00095 * @return Pointer to MotionControl object
00096 *
00097 * @par Python function prototype
00098 * getMotionControl(self: pyaubo_sdk.RobotInterface) ->
00099 * arcs::common_interface::MotionControl
00100 *
00101 * @par C++ example
00102 * @code
00103 * auto rpc_cli = std::make_shared<RpcClient>();
00104 * auto robot_name = rpc_cli->getRobotNames().front();
00105 * MotionControlPtr ptr =
00106 * rpc_cli->getRobotInterface(robot_name)->getMotionControl();
00107 * @endcode
00108 * \endenglish
00109 */
00110 MotionControlPtr getMotionControl();
00111
00112 /**
00113 * @ingroup RobotInterface
00114 * @ref ForceControl
00115 * \chinese
00116 *
00117 *
00118 * @return ForceControlPtr
00119 *
00120 * @par Python
00121 * getForceControl(self: pyaubo_sdk.RobotInterface) ->
00122 * arcs::common_interface::ForceControl
00123 *
00124 * @par C++
00125 * @code
00126 * auto rpc_cli = std::make_shared<RpcClient>();
00127 * auto robot_name = rpc_cli->getRobotNames().front();
00128 * ForceControlPtr ptr =
00129 * rpc_cli->getRobotInterface(robot_name)->getForceControl();
00130 * @endcode
00131 * \endchinese
00132 * \english
00133 * Get force control interface
00134 *
00135 * @return Pointer to ForceControl object
00136 *
00137 * @par Python function prototype
00138 * getForceControl(self: pyaubo_sdk.RobotInterface) ->
00139 * arcs::common_interface::ForceControl
00140 *
00141 * @par C++ example
00142 * @code
00143 * auto rpc_cli = std::make_shared<RpcClient>();
00144 * auto robot_name = rpc_cli->getRobotNames().front();
00145 * ForceControlPtr ptr =
00146 * rpc_cli->getRobotInterface(robot_name)->getForceControl();
00147 * @endcode
00148 * \endenglish

```

```

00149     */
00150     ForceControlPtr getForceControl();
00151
00152     /**
00153     * @ingroup RobotInterface
00154     * @ref IoControl
00155     * \chinese
00156     * IO
00157     *
00158     * @return IoControlPtr
00159     *
00160     * @par Python
00161     * getIoControl(self: pyaubo_sdk.RobotInterface) ->
00162     * arcs::common_interface::IoControl
00163     *
00164     * @par C++
00165     * @code
00166     * auto rpc_cli = std::make_shared<RpcClient>();
00167     * auto robot_name = rpc_cli->getRobotNames().front();
00168     * IoControlPtr ptr =
00169     * rpc_cli->getRobotInterface(robot_name)->getIoControl();
00170     * @endcode
00171     * \endchinese
00172     * \english
00173     * Get IO control interface
00174     *
00175     * @return Pointer to IoControl object
00176     *
00177     * @par Python function prototype
00178     * getIoControl(self: pyaubo_sdk.RobotInterface) ->
00179     * arcs::common_interface::IoControl
00180     *
00181     * @par C++ example
00182     * @code
00183     * auto rpc_cli = std::make_shared<RpcClient>();
00184     * auto robot_name = rpc_cli->getRobotNames().front();
00185     * IoControlPtr ptr =
00186     * rpc_cli->getRobotInterface(robot_name)->getIoControl();
00187     * @endcode
00188     * \endenglish
00189     */
00190     IoControlPtr getIoControl();
00191
00192     /**
00193     * @ingroup RobotInterface
00194     * @ref SyncMove
00195     * \chinese
00196     *
00197     *
00198     * @return SyncMovePtr
00199     *
00200     * @par Python
00201     * getSyncMove(self: pyaubo_sdk.RobotInterface) ->
00202     * arcs::common_interface::SyncMove
00203     *
00204     * @par C++
00205     * @code
00206     * auto rpc_cli = std::make_shared<RpcClient>();
00207     * auto robot_name = rpc_cli->getRobotNames().front();
00208     * SyncMovePtr ptr = rpc_cli->getRobotInterface(robot_name)->getSyncMove();
00209     * @endcode
00210     * \endchinese
00211     * \english
00212     * Get synchronized motion interface
00213     *
00214     * @return Pointer to SyncMove object
00215     *
00216     * @par Python function prototype
00217     * getSyncMove(self: pyaubo_sdk.RobotInterface) ->
00218     * arcs::common_interface::SyncMove
00219     *
00220     * @par C++ example
00221     * @code
00222     * auto rpc_cli = std::make_shared<RpcClient>();
00223     * auto robot_name = rpc_cli->getRobotNames().front();
00224     * SyncMovePtr ptr = rpc_cli->getRobotInterface(robot_name)->getSyncMove();
00225     * @endcode
00226     * \endenglish
00227     */
00228     SyncMovePtr getSyncMove();
00229
00230     /**
00231     * @ingroup RobotInterface
00232     * @ref RobotAlgorithm
00233     * \chinese
00234     *
00235     *

```

```

00236 * @return RobotAlgorithmPtr
00237 *
00238 * @par Python
00239 * getRobotAlgorithm(self: pyaubo_sdk.RobotInterface) ->
00240 * arcs::common_interface::RobotAlgorithm
00241 *
00242 * @par C++
00243 * @code
00244 * auto rpc_cli = std::make_shared<RpcClient>();
00245 * auto robot_name = rpc_cli->getRobotNames().front();
00246 * RobotAlgorithmPtr ptr =
00247 * rpc_cli->getRobotInterface(robot_name)->getRobotAlgorithm();
00248 * @endcode
00249 * \endchinese
00250 * \english
00251 * Get robot utility algorithm interface
00252 *
00253 * @return Pointer to RobotAlgorithm object
00254 *
00255 * @par Python function prototype
00256 * getRobotAlgorithm(self: pyaubo_sdk.RobotInterface) ->
00257 * arcs::common_interface::RobotAlgorithm
00258 *
00259 * @par C++ example
00260 * @code
00261 * auto rpc_cli = std::make_shared<RpcClient>();
00262 * auto robot_name = rpc_cli->getRobotNames().front();
00263 * RobotAlgorithmPtr ptr =
00264 * rpc_cli->getRobotInterface(robot_name)->getRobotAlgorithm();
00265 * @endcode
00266 * \endenglish
00267 */
00268 RobotAlgorithmPtr getRobotAlgorithm();
00269
00270 /**
00271 * @ingroup RobotInterface
00272 * @ref RobotManage
00273 * \chinese
00274 * ( )
00275 *
00276 * @return RobotManagePtr
00277 *
00278 * @par Python
00279 * getRobotManage(self: pyaubo_sdk.RobotInterface) ->
00280 * arcs::common_interface::RobotManage
00281 *
00282 * @par C++
00283 * @code
00284 * auto rpc_cli = std::make_shared<RpcClient>();
00285 * auto robot_name = rpc_cli->getRobotNames().front();
00286 * RobotManagePtr ptr =
00287 * rpc_cli->getRobotInterface(robot_name)->getRobotManage();
00288 * @endcode
00289 * \endchinese
00290 * \english
00291 * Get robot management interface (power on, start, stop, etc.)
00292 *
00293 * @return Pointer to RobotManage object
00294 *
00295 * @par Python function prototype
00296 * getRobotManage(self: pyaubo_sdk.RobotInterface) ->
00297 * arcs::common_interface::RobotManage
00298 *
00299 * @par C++ example
00300 * @code
00301 * auto rpc_cli = std::make_shared<RpcClient>();
00302 * auto robot_name = rpc_cli->getRobotNames().front();
00303 * RobotManagePtr ptr =
00304 * rpc_cli->getRobotInterface(robot_name)->getRobotManage();
00305 * @endcode
00306 * \endenglish
00307 */
00308 RobotManagePtr getRobotManage();
00309
00310 /**
00311 * @ingroup RobotInterface
00312 * @ref RobotState
00313 * \chinese
00314 *
00315 *
00316 * @return RobotStatePtr
00317 *
00318 * @par Python
00319 * getRobotState(self: pyaubo_sdk.RobotInterface) ->
00320 * arcs::common_interface::RobotState
00321 *
00322 * @par C++

```



```

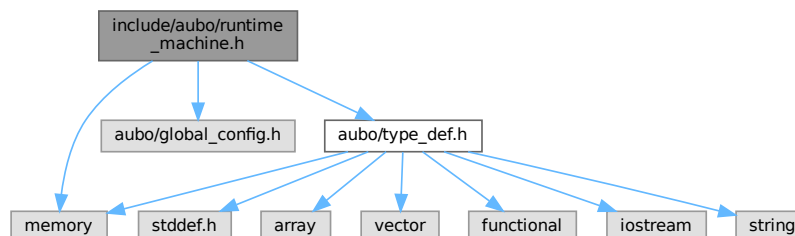
00323 * @code
00324 * auto rpc_cli = std::make_shared<RpcClient>();
00325 * auto robot_name = rpc_cli->getRobotNames().front();
00326 * RobotStatePtr ptr =
00327 * rpc_cli->getRobotInterface(robot_name)->getRobotState();
00328 * @endcode
00329 * \endchinese
00330 * \english
00331 * Get robot state interface
00332 *
00333 * @return Pointer to RobotState object
00334 *
00335 * @par Python function prototype
00336 * getRobotState(self: pyaubo_sdk.RobotInterface) ->
00337 * arcs::common_interface::RobotState
00338 *
00339 * @par C++ example
00340 * @code
00341 * auto rpc_cli = std::make_shared<RpcClient>();
00342 * auto robot_name = rpc_cli->getRobotNames().front();
00343 * RobotStatePtr ptr =
00344 * rpc_cli->getRobotInterface(robot_name)->getRobotState();
00345 * @endcode
00346 * \endenglish
00347 */
00348 RobotStatePtr getRobotState();
00349
00350 /**
00351 * @ingroup RobotInterface
00352 * @ref Trace
00353 * \chinese
00354 *
00355 *
00356 * @return TracePtr
00357 *
00358 * @par Python
00359 * getTrace(self: pyaubo_sdk.RobotInterface) ->
00360 * arcs::common_interface::Trace
00361 *
00362 * @par C++
00363 * @code
00364 * auto rpc_cli = std::make_shared<RpcClient>();
00365 * auto robot_name = rpc_cli->getRobotNames().front();
00366 * TracePtr ptr = rpc_cli->getRobotInterface(robot_name)->getTrace();
00367 * @endcode
00368 * \endchinese
00369 * \english
00370 * Get alarm information interface
00371 *
00372 * @return Pointer to Trace object
00373 *
00374 * @par Python function prototype
00375 * getTrace(self: pyaubo_sdk.RobotInterface) ->
00376 * arcs::common_interface::Trace
00377 *
00378 * @par C++ example
00379 * @code
00380 * auto rpc_cli = std::make_shared<RpcClient>();
00381 * auto robot_name = rpc_cli->getRobotNames().front();
00382 * TracePtr ptr = rpc_cli->getRobotInterface(robot_name)->getTrace();
00383 * @endcode
00384 * \endenglish
00385 */
00386 TracePtr getTrace();
00387
00388 protected:
00389 void *d_;
00390 };
00391 using RobotInterfacePtr = std::shared_ptr<RobotInterface>;
00392
00393 } // namespace common_interface
00394 } // namespace arcs
00395
00396 #endif // AUBO_SDK_ROBOT_INTERFACE_H

```

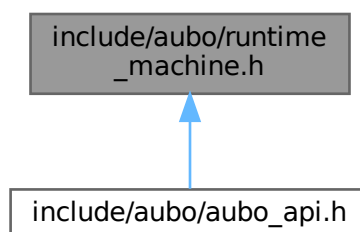
12.36 include/aubo/runtime_machine.h File Reference

Script interpreter runtime interface, allows pausing the script interpreter and setting/removing breakpoints.

```
#include <memory>
#include <aubo/global_config.h>
#include <aubo/type_def.h>
Include dependency graph for runtime_machine.h:
```



This graph shows which files directly or indirectly include this file:



Classes

- class [arcs::common_interface::RuntimeMachine](#)

Namespaces

- namespace [arcs](#)
- namespace [arcs::common_interface](#)

Typedefs

- using [arcs::common_interface::RuntimeMachinePtr](#) = `std::shared_ptr<RuntimeMachine>`

12.36.1 Detailed Description

Script interpreter runtime interface, allows pausing the script interpreter and setting/removing break-points.

Definition in file [runtime_machine.h](#).

12.37 runtime_machine.h

[Go to the documentation of this file.](#)

```

00001 /** @file runtime_machine.h
00002  * @brief Script interpreter runtime interface,
00003  * allows pausing the script interpreter and setting/removing breakpoints.
00004  */
00005 #ifndef AUBO_SDK_RUNTIME_MACHINE_INTERFACE_H
00006 #define AUBO_SDK_RUNTIME_MACHINE_INTERFACE_H
00007
00008 #include <memory>
00009 #include <aubo/global_config.h>
00010 #include <aubo/type_def.h>
00011
00012 namespace arcs {
00013 namespace common_interface {
00014
00015 /**
00016  * @defgroup RuntimeMachine RuntimeMachine ( )
00017  * The RuntimeMachine class
00018  */
00019 class ARCS_ABI_EXPORT RuntimeMachine
00020 {
00021 public:
00022     RuntimeMachine();
00023     virtual ~RuntimeMachine();
00024
00025     /**
00026      * @ingroup RuntimeMachine
00027      * \chinese
00028      * task_id
00029      *
00030      * @par JSON-RPC
00031      * {"jsonrpc":"2.0","method":"RuntimeMachine.newTask","params":[false],"id":1}
00032      *
00033      * @par JSON-RPC
00034      * {"id":1,"jsonrpc":"2.0","result":26}
00035      * \endchinese
00036      * \english
00037      * Returns the task_id
00038      *
00039      * @par JSON-RPC Request Example
00040      * {"jsonrpc":"2.0","method":"RuntimeMachine.newTask","params":[false],"id":1}
00041      *
00042      * @par JSON-RPC Response Example
00043      * {"id":1,"jsonrpc":"2.0","result":26}
00044      * \endenglish
00045      */
00046     int newTask(bool daemon = false);
00047
00048     /**
00049      * @ingroup RuntimeMachine
00050      * \chinese
00051      * task
00052      *
00053      * @par JSON-RPC
00054      * {"jsonrpc":"2.0","method":"RuntimeMachine.deleteTask","params":[26],"id":1}
00055      *
00056      * @par JSON-RPC
00057      * {"id":1,"jsonrpc":"2.0","result":0}
00058      *
00059      * \endchinese
00060      * \english
00061      * Delete a task, which will terminate any ongoing motion.
00062      *
00063      * @par JSON-RPC Request Example
00064      * {"jsonrpc":"2.0","method":"RuntimeMachine.deleteTask","params":[26],"id":1}
00065      *
00066      * @par JSON-RPC Response Example
00067      * {"id":1,"jsonrpc":"2.0","result":0}
00068      *
00069      * \endenglish
00070      */
00071     int deleteTask(int tid);
00072
00073     /**
00074      * @ingroup RuntimeMachine
00075      * \chinese
00076      * task
00077      *
00078      * @param tid
00079      * @return
00080      *
00081      * @par JSON-RPC
00082      * {"jsonrpc":"2.0","method":"RuntimeMachine.detachTask","params":[26],"id":1}

```

```

00083      *
00084      * @par JSON-RPC
00085      * {"id":1,"jsonrpc":"2.0","result":0}
00086      *
00087      * \endchinese
00088      * \english
00089      * Wait for the task to finish naturally
00090      *
00091      * @param tid
00092      * @return
00093      *
00094      * @par JSON-RPC Request Example
00095      * {"jsonrpc":"2.0","method":"RuntimeMachine.detachTask","params":[26],"id":1}
00096      *
00097      * @par JSON-RPC Response Example
00098      * {"id":1,"jsonrpc":"2.0","result":0}
00099      *
00100      * \endenglish
00101      */
00102 int detachTask(int tid);
00103
00104 /**
00105  * @ingroup RuntimeMachine
00106  * \chinese
00107  *
00108  *
00109  * @param tid
00110  * @return
00111  *
00112  * @par JSON-RPC
00113  * {"jsonrpc":"2.0","method":"RuntimeMachine.isTaskAlive","params":[26],"id":1}
00114  *
00115  * @par JSON-RPC
00116  * {"id":1,"jsonrpc":"2.0","result":true}
00117  *
00118  * \endchinese
00119  * \english
00120  * Check if the task is alive
00121  *
00122  * @param tid
00123  * @return
00124  *
00125  * @par JSON-RPC Request Example
00126  * {"jsonrpc":"2.0","method":"RuntimeMachine.isTaskAlive","params":[26],"id":1}
00127  *
00128  * @par JSON-RPC Response Example
00129  * {"id":1,"jsonrpc":"2.0","result":true}
00130  *
00131  * \endenglish
00132  */
00133 bool isTaskAlive(int tid);
00134
00135 /**
00136  * @ingroup RuntimeMachine
00137  * \chinese
00138  *
00139  *
00140  * @param tid
00141  * @return
00142  * \endchinese
00143  * \english
00144  * Get the number of cached instructions in the task
00145  *
00146  * @param tid
00147  * @return
00148  * \endenglish
00149  */
00150 int getTaskQueueSize(int tid);
00151
00152 /**
00153  * @ingroup RuntimeMachine
00154  * \chinese
00155  *
00156  *
00157  * @param tid
00158  * @return
00159  *
00160  * @par JSON-RPC
00161  * {"jsonrpc":"2.0","method":"RuntimeMachine.switchTask","params":[26],"id":1}
00162  *
00163  * @par JSON-RPC
00164  * {"id":1,"jsonrpc":"2.0","result":0}
00165  *
00166  * \endchinese
00167  * \english
00168  * Switch the current thread. After switching, subsequent instructions will
00169  * be inserted into the switched thread.

```

```

00170  *
00171  * @param tid
00172  * @return
00173  *
00174  * @par JSON-RPC Request Example
00175  * {"jsonrpc":"2.0","method":"RuntimeMachine.switchTask","params":[26],"id":1}
00176  *
00177  * @par JSON-RPC Response Example
00178  * {"id":1,"jsonrpc":"2.0","result":0}
00179  *
00180  * \endenglish
00181  */
00182 int switchTask(int tid);
00183
00184 /**
00185  * @ingroup RuntimeMachine
00186  * \chinese
00187  *
00188  *
00189  * @param lineno
00190  * @param comment
00191  * @return
00192  *
00193  * @par JSON-RPC
00194  * {"jsonrpc":"2.0","method":"RuntimeMachine.setLabel","params":[5,"moveJoint"],"id":1}
00195  *
00196  * @par JSON-RPC
00197  * {"id":1,"jsonrpc":"2.0","result":0}
00198  *
00199  * \endchinese
00200  * \english
00201  * Mark the line number and comment of the recorded instruction
00202  *
00203  * @param lineno
00204  * @param comment
00205  * @return
00206  *
00207  * @par JSON-RPC Request Example
00208  * {"jsonrpc":"2.0","method":"RuntimeMachine.setLabel","params":[5,"moveJoint"],"id":1}
00209  *
00210  * @par JSON-RPC Response Example
00211  * {"id":1,"jsonrpc":"2.0","result":0}
00212  *
00213  * \endenglish
00214  */
00215 int setLabel(int lineno, const std::string &comment);
00216
00217 /**
00218  * @ingroup RuntimeMachine
00219  * \chinese
00220  * aubo_control
00221  * setLabel
00222  *
00223  * @param tid ID
00224  * @param lineno
00225  * @param comment
00226  * @return
00227  *
00228  * @par Python
00229  * setPlanContext(self: pyaubo_sdk.RuntimeMachine, arg0: int, arg1: int,
00230  * arg2: str) -> int
00231  *
00232  * @par Lua
00233  * setPlanContext(tid: number, lineno: number, comment: string) -> number
00234  *
00235  * @par JSON-RPC
00236  * {"jsonrpc":"2.0","method":"RuntimeMachine.setPlanContext","params":[26,3,"moveJoint"],"id":1}
00237  *
00238  * @par JSON-RPC
00239  * {"id":1,"jsonrpc":"2.0","result":0}
00240  *
00241  * \endchinese
00242  * \english
00243  * Add a comment to the aubo_control log
00244  * Use setLabel instead
00245  *
00246  * @param tid Thread ID of the instruction
00247  * @param lineno Line number
00248  * @param comment Comment
00249  * @return
00250  *
00251  * @par Python function prototype
00252  * setPlanContext(self: pyaubo_sdk.RuntimeMachine, arg0: int, arg1: int,
00253  * arg2: str) -> int
00254  *
00255  * @par Lua function prototype
00256  * setPlanContext(tid: number, lineno: number, comment: string) -> number

```

```

00257 *
00258 * @par JSON-RPC Request Example
00259 * {"jsonrpc":"2.0","method":"RuntimeMachine.setPlanContext","params":[26,3,"moveJoint"],"id":1}
00260 *
00261 * @par JSON-RPC Response Example
00262 * {"id":1,"jsonrpc":"2.0","result":0}
00263 *
00264 * \endenglish
00265 */
00266 ARCS_DEPRECATED int setPlanContext(int tid, int lineno,
00267                                     const std::string &comment);
00268
00269 /**
00270 * @ingroup RuntimeMachine
00271 * \chinese
00272 *
00273 *
00274 * @return
00275 *
00276 * @par JSON-RPC
00277 * {"jsonrpc":"2.0","method":"RuntimeMachine.nop","params":[],"id":1}
00278 *
00279 * @par JSON-RPC
00280 * {"id":1,"jsonrpc":"2.0","result":0}
00281 *
00282 * \endchinese
00283 * \english
00284 * No operation
00285 *
00286 * @return
00287 *
00288 * @par JSON-RPC Request Example
00289 * {"jsonrpc":"2.0","method":"RuntimeMachine.nop","params":[],"id":1}
00290 *
00291 * @par JSON-RPC Response Example
00292 * {"id":1,"jsonrpc":"2.0","result":0}
00293 *
00294 * \endenglish
00295 */
00296 int nop();
00297
00298 /**
00299 * @ingroup RuntimeMachine
00300 * \chinese
00301 * (INST) , setPersistentParameters
00302 *
00303 * @return ,
00304 * : EXECUTING/FINISHED
00305 *
00306 * @par Python
00307 * getExecutionStatus(self: pyaubo_sdk.RuntimeMachine) -> Tuple[str, str,
00308 * int]
00309 *
00310 * @par Lua
00311 * getExecutionStatus() -> string, string, number
00312 *
00313 * @par JSON-RPC
00314 * {"jsonrpc":"2.0","method":"RuntimeMachine.getExecutionStatus","params":[],"id":1}
00315 *
00316 * @par JSON-RPC
00317 * {"id":1,"jsonrpc":"2.0","result":["confirmSafetyParameters","FINISHED"]}
00318 *
00319 * \endchinese
00320 * \english
00321 * Get the execution status of time-consuming interfaces (INST), such as
00322 * setPersistentParameters
00323 *
00324 * @return Instruction name, execution status
00325 * Execution status: EXECUTING/FINISHED
00326 *
00327 * @par Python function prototype
00328 * getExecutionStatus(self: pyaubo_sdk.RuntimeMachine) -> Tuple[str, str,
00329 * int]
00330 *
00331 * @par Lua function prototype
00332 * getExecutionStatus() -> string, string, number
00333 *
00334 * @par JSON-RPC Request Example
00335 * {"jsonrpc":"2.0","method":"RuntimeMachine.getExecutionStatus","params":[],"id":1}
00336 *
00337 * @par JSON-RPC Response Example
00338 * {"id":1,"jsonrpc":"2.0","result":["confirmSafetyParameters","FINISHED"]}
00339 *
00340 * \endenglish
00341 */
00342 std::tuple<std::string, std::string> getExecutionStatus();
00343 std::tuple<std::string, std::string, int> getExecutionStatus1();

```

```

00344
00345 /**
00346  * @ingroup RuntimeMachine
00347  * \chinese
00348  *
00349  *
00350  * @param lineno
00351  * @return
00352  *
00353  * @par Python
00354  * gotoLine(self: pyaubo_sdk.RuntimeMachine, arg0: int) -> int
00355  *
00356  * @par Lua
00357  * gotoLine(lineno: number) -> number
00358  *
00359  * @par JSON-RPC
00360  * {"jsonrpc":"2.0","method":"RuntimeMachine.gotoLine","params":[10],"id":1}
00361  *
00362  * @par JSON-RPC
00363  * {"id":1,"jsonrpc":"2.0","result":0}
00364  *
00365  * \endchinese
00366  * \english
00367  * Jump to the specified line number
00368  *
00369  * @param lineno
00370  * @return
00371  *
00372  * @par Python function prototype
00373  * gotoLine(self: pyaubo_sdk.RuntimeMachine, arg0: int) -> int
00374  *
00375  * @par Lua function prototype
00376  * gotoLine(lineno: number) -> number
00377  *
00378  * @par JSON-RPC Request Example
00379  * {"jsonrpc":"2.0","method":"RuntimeMachine.gotoLine","params":[10],"id":1}
00380  *
00381  * @par JSON-RPC Response Example
00382  * {"id":1,"jsonrpc":"2.0","result":0}
00383  *
00384  * \endenglish
00385  */
00386 int gotoLine(int lineno);
00387
00388 /**
00389  * @ingroup RuntimeMachine
00390  * \chinese
00391  *
00392  *
00393  * @param tid
00394  * ( -1)      ( -1)
00395  *
00396  * @return
00397  *
00398  * @par Python
00399  * getPlanContext(self: pyaubo_sdk.RuntimeMachine) -> Tuple[int, int, str]
00400  *
00401  * @par Lua
00402  * getPlanContext() -> number
00403  *
00404  * @par JSON-RPC
00405  * {"jsonrpc":"2.0","method":"RuntimeMachine.getPlanContext","params":[-1],"id":1}
00406  *
00407  * @par JSON-RPC
00408  * {"id":1,"jsonrpc":"2.0","result":[-1,0,""]}
00409  *
00410  * \endchinese
00411  * \english
00412  * Get the current runtime context
00413  *
00414  * @param tid Task ID
00415  * If specified (not -1), returns the runtime context of the corresponding
00416  * task; if not specified (is -1), returns the runtime context of the
00417  * currently running thread
00418  *
00419  * @return
00420  *
00421  * @par Python function prototype
00422  * getPlanContext(self: pyaubo_sdk.RuntimeMachine) -> Tuple[int, int, str]
00423  *
00424  * @par Lua function prototype
00425  * getPlanContext() -> number
00426  *
00427  * @par JSON-RPC Request Example
00428  * {"jsonrpc":"2.0","method":"RuntimeMachine.getPlanContext","params":[-1],"id":1}
00429  *
00430  * @par JSON-RPC Response Example

```

```

00431     * {"id":1,"jsonrpc":"2.0","result":[-1,0,""]}
00432     *
00433     * \endenglish
00434     */
00435 std::tuple<int, int, std::string> getPlanContext(int tid = -1);
00436
00437 /**
00438  * @ingroup RuntimeMachine
00439  * \chinese
00440  *
00441  *
00442  * @param tid
00443  *   (-1)          (-1)
00444  *
00445  * @return
00446  *
00447  * @par JSON-RPC
00448  * {"jsonrpc":"2.0","method":"RuntimeMachine.getAdvancePlanContext","params":[-1],"id":1}
00449  *
00450  * @par JSON-RPC
00451  * {"id":1,"jsonrpc":"2.0","result":[-1,-1,""]}
00452  *
00453  * \endchinese
00454  * \english
00455  * Get the context information of the advance planner
00456  *
00457  * @param tid Task ID
00458  * If specified (not -1), returns the context information of the advance
00459  * planner for the corresponding task; if not specified (is -1), returns the
00460  * context information of the advance planner for the currently running
00461  * thread
00462  *
00463  * @return
00464  *
00465  * @par JSON-RPC Request Example
00466  * {"jsonrpc":"2.0","method":"RuntimeMachine.getAdvancePlanContext","params":[-1],"id":1}
00467  *
00468  * @par JSON-RPC Response Example
00469  * {"id":1,"jsonrpc":"2.0","result":[-1,-1,""]}
00470  *
00471  * \endenglish
00472  */
00473 std::tuple<int, int, std::string> getAdvancePlanContext(int tid = -1);
00474
00475 /**
00476  * @ingroup RuntimeMachine
00477  * \chinese
00478  * AdvanceRun
00479  *
00480  * @return
00481  *
00482  * @par JSON-RPC
00483  * {"jsonrpc":"2.0","method":"RuntimeMachine.getAdvancePtr","params":[-1],"id":1}
00484  *
00485  * @par JSON-RPC
00486  * {"id":1,"jsonrpc":"2.0","result":-1}
00487  *
00488  * \endchinese
00489  * \english
00490  * Get the program pointer of AdvanceRun
00491  *
00492  * @return
00493  *
00494  * @par JSON-RPC Request Example
00495  * {"jsonrpc":"2.0","method":"RuntimeMachine.getAdvancePtr","params":[-1],"id":1}
00496  *
00497  * @par JSON-RPC Response Example
00498  * {"id":1,"jsonrpc":"2.0","result":-1}
00499  *
00500  * \endenglish
00501  */
00502 int getAdvancePtr(int tid = -1);
00503
00504 /**
00505  * @ingroup RuntimeMachine
00506  * \chinese
00507  *
00508  *
00509  * @param tid
00510  *   (-1)          (-1)
00511  *
00512  * @return
00513  *
00514  * @par JSON-RPC
00515  * {"jsonrpc":"2.0","method":"RuntimeMachine.getMainPtr","params":[-1],"id":1}
00516  *
00517  * @par JSON-RPC

```



```

00518 * {"id":1,"jsonrpc":"2.0","result":-1}
00519 *
00520 * \endchinese
00521 * \english
00522 * Get the program pointer of robot motion
00523 *
00524 * @param tid Task ID
00525 * If specified (not -1), returns the program pointer of the corresponding
00526 * task; if not specified (is -1), returns the program pointer of the
00527 * currently running thread
00528 *
00529 * @return
00530 *
00531 * @par JSON-RPC Request Example
00532 * {"jsonrpc":"2.0","method":"RuntimeMachine.getMainPtr","params":[-1],"id":1}
00533 *
00534 * @par JSON-RPC Response Example
00535 * {"id":1,"jsonrpc":"2.0","result":-1}
00536 *
00537 * \endenglish
00538 */
00539 int getMainPtr(int tid = -1);
00540
00541 /**
00542 * @ingroup RuntimeMachine
00543 * \chinese
00544 *
00545 *
00546 * @param tid
00547 * @return
00548 *
00549 * @par JSON-RPC
00550 * {"jsonrpc":"2.0","method":"RuntimeMachine.getInterpPtr","params":[26],"id":1}
00551 *
00552 * @par JSON-RPC
00553 * {"id":1,"jsonrpc":"2.0","result":-1}
00554 *
00555 * \endchinese
00556 * \english
00557 * Get the pointer of the most recently interpreted instruction
00558 *
00559 * @param tid
00560 * @return
00561 *
00562 * @par JSON-RPC Request Example
00563 * {"jsonrpc":"2.0","method":"RuntimeMachine.getInterpPtr","params":[26],"id":1}
00564 *
00565 * @par JSON-RPC Response Example
00566 * {"id":1,"jsonrpc":"2.0","result":-1}
00567 *
00568 * \endenglish
00569 */
00570 int getInterpPtr(int tid);
00571
00572 /**
00573 * @ingroup RuntimeMachine
00574 * \chinese
00575 *
00576 * Lua          ${ARCS_WS}/program
00577 *
00578 *
00579 * @param program
00580 * @return
00581 *
00582 * @par JSON-RPC
00583 * {"jsonrpc":"2.0","method":"RuntimeMachine.loadProgram","params":["demo"],"id":1}
00584 *
00585 * @par JSON-RPC
00586 * {"id":1,"jsonrpc":"2.0","result":0}
00587 *
00588 * \endchinese
00589 * \english
00590 * Load a local project file.
00591 * For Lua scripts, only the file name is required (no extension), and it
00592 * will be searched in the ${ARCS_WS}/program directory.
00593 *
00594 * @param program
00595 * @return
00596 *
00597 * @par JSON-RPC Request Example
00598 * {"jsonrpc":"2.0","method":"RuntimeMachine.loadProgram","params":["demo"],"id":1}
00599 *
00600 * @par JSON-RPC Response Example
00601 * {"id":1,"jsonrpc":"2.0","result":0}
00602 *
00603 * \endenglish
00604 */

```

```

00605 int loadProgram(const std::string &program);
00606
00607 /**
00608  * @ingroup RuntimeMachine
00609  * \chinese
00610  *
00611  *
00612  * @param index 0~99
00613  * @param program
00614  * @return
00615  * \endchinese
00616  * \english
00617  * Preload project file
00618  *
00619  * @param index Project index number (0~99)
00620  * @param program Project name
00621  * @return
00622  * \endenglish
00623  */
00624 int preloadProgram(int index, const std::string &program);
00625
00626 /**
00627  * @ingroup RuntimeMachine
00628  * \chinese
00629  *
00630  *
00631  * @param index 0~99
00632  * @return
00633  * \endchinese
00634  * \english
00635  * Get the name of the preloaded project file. Returns an empty string if
00636  * not loaded or index is out of range.
00637  *
00638  * @param index Project index number (0~99)
00639  * @return Project file name
00640  * \endenglish
00641  */
00642 std::string getPreloadProgram(int index);
00643
00644 /**
00645  * @ingroup RuntimeMachine
00646  * \chinese
00647  *
00648  *      preloadProgram
00649  *
00650  *
00651  * @return 0
00652  * \endchinese
00653  * \english
00654  * Clear all preloaded project files.
00655  * Calling this method will release all project indices and their associated
00656  * program names that were previously preloaded via preloadProgram.
00657  *
00658  * @return Returns 0 on success;
00659  * \endenglish
00660  */
00661 int clearPreloadPrograms();
00662
00663 /**
00664  * @ingroup RuntimeMachine
00665  * \chinese
00666  *
00667  *
00668  * @return
00669  *
00670  * @par JSON-RPC
00671  * {"jsonrpc":"2.0","method":"RuntimeMachine.runProgram","params":[],"id":1}
00672  *
00673  * @par JSON-RPC
00674  * {"id":1,"jsonrpc":"2.0","result":0}
00675  *
00676  * \endchinese
00677  * \english
00678  * Run the already loaded project file
00679  *
00680  * @return
00681  *
00682  * @par JSON-RPC Request Example
00683  * {"jsonrpc":"2.0","method":"RuntimeMachine.runProgram","params":[],"id":1}
00684  *
00685  * @par JSON-RPC Response Example
00686  * {"id":1,"jsonrpc":"2.0","result":0}
00687  *
00688  * \endenglish
00689  */
00690 int runProgram();
00691

```

```

00692  /**
00693   * @ingroup RuntimeMachine
00694   * \chinese
00695   *
00696   *
00697   * @return
00698   *
00699   * @par Python
00700   * start(self: pyaubo_sdk.RuntimeMachine) -> int
00701   *
00702   * @par Lua
00703   * start() -> number
00704   *
00705   * @par JSON-RPC
00706   * {"jsonrpc":"2.0","method":"RuntimeMachine.start","params":[],"id":1}
00707   *
00708   * @par JSON-RPC
00709   * {"id":1,"jsonrpc":"2.0","result":0}
00710   *
00711   * \endchinese
00712   * \english
00713   * Start the runtime
00714   *
00715   * @return
00716   *
00717   * @par Python function prototype
00718   * start(self: pyaubo_sdk.RuntimeMachine) -> int
00719   *
00720   * @par Lua function prototype
00721   * start() -> number
00722   *
00723   * @par JSON-RPC Request Example
00724   * {"jsonrpc":"2.0","method":"RuntimeMachine.start","params":[],"id":1}
00725   *
00726   * @par JSON-RPC Response Example
00727   * {"id":1,"jsonrpc":"2.0","result":0}
00728   *
00729   * \endenglish
00730   */
00731  int start();
00732
00733  /**
00734   * @ingroup RuntimeMachine
00735   * \chinese
00736   *           Stopped
00737   *
00738   *           RuntimeMachine::abort
00739   *
00740   * @return
00741   *
00742   * @par Python
00743   * stop(self: pyaubo_sdk.RuntimeMachine) -> int
00744   *
00745   * @par Lua
00746   * stop() -> number
00747   *
00748   * @par JSON-RPC
00749   * {"jsonrpc":"2.0","method":"RuntimeMachine.stop","params":[],"id":1}
00750   *
00751   * @par JSON-RPC
00752   * {"id":1,"jsonrpc":"2.0","result":0}
00753   *
00754   * \endchinese
00755   * \english
00756   * Stop the runtime, i.e., stop script execution. Cannot stop robot motion
00757   * when the runtime is Stopped.
00758   *
00759   * If you want to stop all robot motion, use the RuntimeMachine::abort
00760   * interface.
00761   *
00762   * @return
00763   *
00764   * @par Python function prototype
00765   * stop(self: pyaubo_sdk.RuntimeMachine) -> int
00766   *
00767   * @par Lua function prototype
00768   * stop() -> number
00769   *
00770   * @par JSON-RPC Request Example
00771   * {"jsonrpc":"2.0","method":"RuntimeMachine.stop","params":[],"id":1}
00772   *
00773   * @par JSON-RPC Response Example
00774   * {"id":1,"jsonrpc":"2.0","result":0}
00775   *
00776   * \endenglish
00777   */
00778  int stop();

```

```

00779
00780 /**
00781  * @ingroup RuntimeMachine
00782  * \chinese
00783  *
00784  *
00785  *          RuntimeMachine::stop
00786  *
00787  *          Running          Stopped
00788  *
00789  *
00790  * @return
00791  *
00792  * @par Python
00793  * abort(self: pyaubo_sdk.RuntimeMachine) -> int
00794  *
00795  * @par Lua
00796  * abort() -> number
00797  *
00798  * @par JSON-RPC
00799  * {"jsonrpc":"2.0","method":"RuntimeMachine.abort","params":[],"id":1}
00800  *
00801  * @par JSON-RPC
00802  * {"id":1,"jsonrpc":"2.0","result":0}
00803  *
00804  * \endchinese
00805  * \english
00806  * Abort robot operation.
00807  *
00808  * If you only want to stop the runtime, you can call the
00809  * RuntimeMachine::stop interface.
00810  *
00811  * If the script runtime is in the Running state, aborts the runtime; if the
00812  * runtime is Stopped and the robot is moving, stops the robot motion; if
00813  * force control is enabled, stops force control.
00814  *
00815  * @return
00816  *
00817  * @par Python function prototype
00818  * abort(self: pyaubo_sdk.RuntimeMachine) -> int
00819  *
00820  * @par Lua function prototype
00821  * abort() -> number
00822  *
00823  * @par JSON-RPC Request Example
00824  * {"jsonrpc":"2.0","method":"RuntimeMachine.abort","params":[],"id":1}
00825  *
00826  * @par JSON-RPC Response Example
00827  * {"id":1,"jsonrpc":"2.0","result":0}
00828  *
00829  * \endenglish
00830  */
00831 int abort();
00832
00833 /**
00834  * @ingroup RuntimeMachine
00835  * \chinese
00836  *
00837  *
00838  * @return
00839  *
00840  * @par Python
00841  * pause(self: pyaubo_sdk.RuntimeMachine) -> int
00842  *
00843  * @par Lua
00844  * pause() -> number
00845  *
00846  * @par JSON-RPC
00847  * {"jsonrpc":"2.0","method":"RuntimeMachine.pause","params":[],"id":1}
00848  *
00849  * @par JSON-RPC
00850  * {"id":1,"jsonrpc":"2.0","result":0}
00851  *
00852  * \endchinese
00853  * \english
00854  * Pause the interpreter
00855  *
00856  * @return
00857  *
00858  * @par Python function prototype
00859  * pause(self: pyaubo_sdk.RuntimeMachine) -> int
00860  *
00861  * @par Lua function prototype
00862  * pause() -> number
00863  *
00864  * @par JSON-RPC Request Example
00865  * {"jsonrpc":"2.0","method":"RuntimeMachine.pause","params":[],"id":1}

```

```

00866     *
00867     * @par JSON-RPC Response Example
00868     * {"id":1,"jsonrpc":"2.0","result":0}
00869     *
00870     * \endenglish
00871     */
00872 int pause();
00873
00874 /**
00875  * @ingroup RuntimeMachine
00876  * \chinese
00877  *
00878  *
00879  * @return
00880  *
00881  * @par Python
00882  * step(self: pyaubo_sdk.RuntimeMachine) -> int
00883  *
00884  * @par Lua
00885  * step() -> number
00886  *
00887  * @par JSON-RPC
00888  * {"jsonrpc":"2.0","method":"RuntimeMachine.step","params":[],"id":1}
00889  *
00890  * @par JSON-RPC
00891  * {"id":1,"jsonrpc":"2.0","result":0}
00892  *
00893  * \endchinese
00894  * \english
00895  * Execute a single step
00896  *
00897  * @return
00898  *
00899  * @par Python function prototype
00900  * step(self: pyaubo_sdk.RuntimeMachine) -> int
00901  *
00902  * @par Lua function prototype
00903  * step() -> number
00904  *
00905  * @par JSON-RPC Request Example
00906  * {"jsonrpc":"2.0","method":"RuntimeMachine.step","params":[],"id":1}
00907  *
00908  * @par JSON-RPC Response Example
00909  * {"id":1,"jsonrpc":"2.0","result":0}
00910  *
00911  * \endenglish
00912  */
00913 int step();
00914
00915 /**
00916  * @ingroup RuntimeMachine
00917  * \chinese
00918  *
00919  *
00920  * @return
00921  *
00922  * @par Python
00923  * resume(self: pyaubo_sdk.RuntimeMachine) -> int
00924  *
00925  * @par Lua
00926  * resume() -> number
00927  *
00928  * @par JSON-RPC
00929  * {"jsonrpc":"2.0","method":"RuntimeMachine.resume","params":[],"id":1}
00930  *
00931  * @par JSON-RPC
00932  * {"id":1,"jsonrpc":"2.0","result":0}
00933  *
00934  * \endchinese
00935  * \english
00936  * Resume the interpreter
00937  *
00938  * @return
00939  *
00940  * @par Python function prototype
00941  * resume(self: pyaubo_sdk.RuntimeMachine) -> int
00942  *
00943  * @par Lua function prototype
00944  * resume() -> number
00945  *
00946  * @par JSON-RPC Request Example
00947  * {"jsonrpc":"2.0","method":"RuntimeMachine.resume","params":[],"id":1}
00948  *
00949  * @par JSON-RPC Response Example
00950  * {"id":1,"jsonrpc":"2.0","result":0}
00951  *
00952  * \endenglish

```

```

00953     */
00954     int resume();
00955
00956     /**
00957     * @ingroup RuntimeMachine
00958     * \chinese
00959     * ( )
00960     *
00961     * @return
00962     *
00963     * @par Python
00964     * arbitraryResume(self: pyaubo_sdk.RuntimeMachine) -> int
00965     *
00966     * @par Lua
00967     * arbitraryResume() -> number
00968     *
00969     * @par JSON-RPC
00970     * {"jsonrpc":"2.0","method":"RuntimeMachine.arbitraryResume","params":[],"id":1}
00971     *
00972     * @par JSON-RPC
00973     * {"id":1,"jsonrpc":"2.0","result":0}
00974     *
00975     * \endchinese
00976     * \english
00977     * Resume the interpreter
00978     *
00979     * @return
00980     *
00981     * @par Python function prototype
00982     * arbitraryResume(self: pyaubo_sdk.RuntimeMachine) -> int
00983     *
00984     * @par Lua function prototype
00985     * arbitraryResume() -> number
00986     *
00987     * @par JSON-RPC Request Example
00988     * {"jsonrpc":"2.0","method":"RuntimeMachine.arbitraryResume","params":[],"id":1}
00989     *
00990     * @par JSON-RPC Response Example
00991     * {"id":1,"jsonrpc":"2.0","result":0}
00992     *
00993     * \endenglish
00994     */
00995     int arbitraryResume();
00996
00997     /**
00998     * @ingroup RuntimeMachine
00999     * \chinese
01000     *
01001     *
01002     * @param wait
01003     * @return
01004     *
01005     * @par JSON-RPC
01006     * {"jsonrpc":"2.0","method":"RuntimeMachine.setResumeWait","params":[true],"id":1}
01007     *
01008     * @par JSON-RPC
01009     * {"id":1,"jsonrpc":"2.0","result":0}
01010     *
01011     * \endchinese
01012     * \english
01013     * Wait for the previous sequence to complete before resuming the
01014     * interpreter
01015     *
01016     * @param wait
01017     * @return
01018     *
01019     * @par JSON-RPC Request Example
01020     * {"jsonrpc":"2.0","method":"RuntimeMachine.setResumeWait","params":[true],"id":1}
01021     *
01022     * @par JSON-RPC Response Example
01023     * {"id":1,"jsonrpc":"2.0","result":0}
01024     *
01025     * \endenglish
01026     */
01027     int setResumeWait(bool wait);
01028
01029     /**
01030     * @ingroup RuntimeMachine
01031     * \chinese
01032     * abort
01033     *
01034     * @param timeout    0~5    abort
01035     *                  abort
01036     * @return
01037     *
01038     * @par Python
01039     * enterCritical(self: pyaubo_sdk.RuntimeMachine, arg0: double) -> int

```

```

01040 *
01041 * @par Lua
01042 * enterCritical(timeout: number) -> number
01043 *
01044 * @par JSON-RPC
01045 * {"jsonrpc":"2.0","method":"RuntimeMachine.enterCritical","params":[5.0],"id":1}
01046 *
01047 * @par JSON-RPC
01048 * {"id":1,"jsonrpc":"2.0","result":0}
01049 *
01050 * \endchinese
01051 * \english
01052 * The abort command is deferred during critical sections to avoid
01053 * interrupting internal instructions.
01054 *
01055 * @param timeout (seconds, 0~5): max deferral time for abort. Exceeding it
01056 * forces exit from critical section and triggers abort.
01057 * @return
01058 *
01059 * @par Python function prototype
01060 * enterCritical(self: pyaubo_sdk.RuntimeMachine, arg0: double) -> int
01061 *
01062 * @par Lua function prototype
01063 * enterCritical(timeout: number) -> number
01064 *
01065 * @par JSON-RPC Request Example
01066 * {"jsonrpc":"2.0","method":"RuntimeMachine.enterCritical","params":[5.0],"id":1}
01067 *
01068 * @par JSON-RPC Response Example
01069 * {"id":1,"jsonrpc":"2.0","result":0}
01070 *
01071 * \endenglish
01072 */
01073 int enterCritical(double timeout);
01074
01075 /**
01076 * @ingroup RuntimeMachine
01077 * \chinese
01078 *
01079 *
01080 * @param
01081 * @return
01082 *
01083 * @par Python
01084 * exitCritical(self: pyaubo_sdk.RuntimeMachine) -> int
01085 *
01086 * @par Lua
01087 * exitCritical() -> number
01088 *
01089 * @par JSON-RPC
01090 * {"jsonrpc":"2.0","method":"RuntimeMachine.exitCritical","params":[],"id":1}
01091 *
01092 * @par JSON-RPC
01093 * {"id":1,"jsonrpc":"2.0","result":0}
01094 *
01095 * \endchinese
01096 * \english
01097 * Exit the critical section
01098 *
01099 * @param
01100 * @return
01101 *
01102 * @par Python function prototype
01103 * exitCritical(self: pyaubo_sdk.RuntimeMachine) -> int
01104 *
01105 * @par Lua function prototype
01106 * exitCritical() -> number
01107 *
01108 * @par JSON-RPC Request Example
01109 * {"jsonrpc":"2.0","method":"RuntimeMachine.exitCritical","params":[],"id":1}
01110 *
01111 * @par JSON-RPC Response Example
01112 * {"id":1,"jsonrpc":"2.0","result":0}
01113 *
01114 * \endenglish
01115 */
01116 int exitCritical();
01117
01118 /**
01119 * @ingroup RuntimeMachine
01120 * \chinese
01121 *
01122 *
01123 * @return
01124 *
01125 * @par Python
01126 * getStatus(self: pyaubo_sdk.RuntimeMachine) ->

```

```

01127 * arcs::common_interface::RuntimeState
01128 *
01129 * @par Lua
01130 * getStatus() -> number
01131 *
01132 * @par JSON-RPC
01133 * {"jsonrpc":"2.0","method":"RuntimeMachine.getStatus","params":[],"id":1}
01134 *
01135 * @par JSON-RPC
01136 * {"id":1,"jsonrpc":"2.0","result":"Running"}
01137 *
01138 * \endchinese
01139 * \english
01140 * Get the status of the planner
01141 *
01142 * @return
01143 *
01144 * @par Python function prototype
01145 * getStatus(self: pyaubo_sdk.RuntimeMachine) ->
01146 * arcs::common_interface::RuntimeState
01147 *
01148 * @par Lua function prototype
01149 * getStatus() -> number
01150 *
01151 * @par JSON-RPC Request Example
01152 * {"jsonrpc":"2.0","method":"RuntimeMachine.getStatus","params":[],"id":1}
01153 *
01154 * @par JSON-RPC Response Example
01155 * {"id":1,"jsonrpc":"2.0","result":"Running"}
01156 *
01157 * \endenglish
01158 */
01159 ARCS_DEPRECATED RuntimeState getStatus();
01160 RuntimeState getRuntimeState();
01161
01162 /**
01163 * @ingroup RuntimeMachine
01164 * \chinese
01165 *
01166 *
01167 * @param lineno
01168 * @return
01169 *
01170 * @par Python
01171 * setBreakPoint(self: pyaubo_sdk.RuntimeMachine, arg0: int) -> int
01172 *
01173 * @par Lua
01174 * setBreakPoint(lineno: number) -> number
01175 *
01176 * @par JSON-RPC
01177 * {"jsonrpc":"2.0","method":"RuntimeMachine.setBreakPoint","params":[15],"id":1}
01178 *
01179 * @par JSON-RPC
01180 * {"id":1,"jsonrpc":"2.0","result":0}
01181 *
01182 * \endchinese
01183 * \english
01184 * Set a breakpoint
01185 *
01186 * @param lineno
01187 * @return
01188 *
01189 * @par Python function prototype
01190 * setBreakPoint(self: pyaubo_sdk.RuntimeMachine, arg0: int) -> int
01191 *
01192 * @par Lua function prototype
01193 * setBreakPoint(lineno: number) -> number
01194 *
01195 * @par JSON-RPC Request Example
01196 * {"jsonrpc":"2.0","method":"RuntimeMachine.setBreakPoint","params":[15],"id":1}
01197 *
01198 * @par JSON-RPC Response Example
01199 * {"id":1,"jsonrpc":"2.0","result":0}
01200 *
01201 * \endenglish
01202 */
01203 int setBreakPoint(int lineno);
01204
01205 /**
01206 * @ingroup RuntimeMachine
01207 * \chinese
01208 *
01209 *
01210 * @param lineno
01211 * @return
01212 *
01213 * @par Python

```



```

01214 * removeBreakPoint(self: pyaubo_sdk.RuntimeMachine, arg0: int) -> int
01215 *
01216 * @par Lua
01217 * removeBreakPoint(lineno: number) -> number
01218 *
01219 * @par JSON-RPC
01220 * {"jsonrpc":"2.0","method":"RuntimeMachine.removeBreakPoint","params":[15],"id":1}
01221 *
01222 * @par JSON-RPC
01223 * {"id":1,"jsonrpc":"2.0","result":0}
01224 *
01225 * \endchinese
01226 * \english
01227 * Remove a breakpoint
01228 *
01229 * @param lineno
01230 * @return
01231 *
01232 * @par Python function prototype
01233 * removeBreakPoint(self: pyaubo_sdk.RuntimeMachine, arg0: int) -> int
01234 *
01235 * @par Lua function prototype
01236 * removeBreakPoint(lineno: number) -> number
01237 *
01238 * @par JSON-RPC Request Example
01239 * {"jsonrpc":"2.0","method":"RuntimeMachine.removeBreakPoint","params":[15],"id":1}
01240 *
01241 * @par JSON-RPC Response Example
01242 * {"id":1,"jsonrpc":"2.0","result":0}
01243 *
01244 * \endenglish
01245 */
01246 int removeBreakPoint(int lineno);
01247
01248 /**
01249 * @ingroup RuntimeMachine
01250 * \chinese
01251 *
01252 *
01253 * @return
01254 *
01255 * @par Python
01256 * clearBreakPoints(self: pyaubo_sdk.RuntimeMachine) -> int
01257 *
01258 * @par Lua
01259 * clearBreakPoints() -> number
01260 *
01261 * @par JSON-RPC
01262 * {"jsonrpc":"2.0","method":"RuntimeMachine.clearBreakPoints","params":[],"id":1}
01263 *
01264 * @par JSON-RPC
01265 * {"id":1,"jsonrpc":"2.0","result":0}
01266 *
01267 * \endchinese
01268 * \english
01269 * Clear all breakpoints
01270 *
01271 * @return
01272 *
01273 * @par Python function prototype
01274 * clearBreakPoints(self: pyaubo_sdk.RuntimeMachine) -> int
01275 *
01276 * @par Lua function prototype
01277 * clearBreakPoints() -> number
01278 *
01279 * @par JSON-RPC Request Example
01280 * {"jsonrpc":"2.0","method":"RuntimeMachine.clearBreakPoints","params":[],"id":1}
01281 *
01282 * @par JSON-RPC Response Example
01283 * {"id":1,"jsonrpc":"2.0","result":0}
01284 *
01285 * \endenglish
01286 */
01287 int clearBreakPoints();
01288
01289 /**
01290 * @ingroup RuntimeMachine
01291 * \chinese
01292 *
01293 *
01294 * @param name
01295 * @return
01296 *
01297 * @par Python
01298 * timerStart(self: pyaubo_sdk.RuntimeMachine, arg0: str) -> int
01299 *
01300 * @par Lua

```

```

01301 * timerStart(name: string) -> nil
01302 *
01303 * @par JSON-RPC
01304 * {"jsonrpc":"2.0","method":"RuntimeMachine.timerStart","params":["timer"],"id":1}
01305 *
01306 * @par JSON-RPC
01307 * {"id":1,"jsonrpc":"2.0","result":0}
01308 *
01309 * \endchinese
01310 * \english
01311 * Start the timer
01312 *
01313 * @param name
01314 * @return
01315 *
01316 * @par Python function prototype
01317 * timerStart(self: pyaubo_sdk.RuntimeMachine, arg0: str) -> int
01318 *
01319 * @par Lua function prototype
01320 * timerStart(name: string) -> nil
01321 *
01322 * @par JSON-RPC Request Example
01323 * {"jsonrpc":"2.0","method":"RuntimeMachine.timerStart","params":["timer"],"id":1}
01324 *
01325 * @par JSON-RPC Response Example
01326 * {"id":1,"jsonrpc":"2.0","result":0}
01327 *
01328 * \endenglish
01329 */
01330 int timerStart(const std::string &name);
01331
01332 /**
01333 * @ingroup RuntimeMachine
01334 * \chinese
01335 *
01336 *
01337 * @param name
01338 * @return
01339 *
01340 * @par Python
01341 * timerStop(self: pyaubo_sdk.RuntimeMachine, arg0: str) -> int
01342 *
01343 * @par Lua
01344 * timerStop(name: string) -> nil
01345 *
01346 * @par JSON-RPC
01347 * {"jsonrpc":"2.0","method":"RuntimeMachine.timerStop","params":["timer"],"id":1}
01348 *
01349 * @par JSON-RPC
01350 * {"id":1,"jsonrpc":"2.0","result":0}
01351 *
01352 * \endchinese
01353 * \english
01354 * Stop the timer
01355 *
01356 * @param name
01357 * @return
01358 *
01359 * @par Python function prototype
01360 * timerStop(self: pyaubo_sdk.RuntimeMachine, arg0: str) -> int
01361 *
01362 * @par Lua function prototype
01363 * timerStop(name: string) -> nil
01364 *
01365 * @par JSON-RPC Request Example
01366 * {"jsonrpc":"2.0","method":"RuntimeMachine.timerStop","params":["timer"],"id":1}
01367 *
01368 * @par JSON-RPC Response Example
01369 * {"id":1,"jsonrpc":"2.0","result":0}
01370 *
01371 * \endenglish
01372 */
01373 int timerStop(const std::string &name);
01374
01375 /**
01376 * @ingroup RuntimeMachine
01377 * \chinese
01378 *
01379 *
01380 * @param name
01381 * @return
01382 *
01383 * @par Python
01384 * timerReset(self: pyaubo_sdk.RuntimeMachine, arg0: str) -> int
01385 *
01386 * @par Lua
01387 * timerReset(name: string) -> nil

```

```

01388 *
01389 * @par JSON-RPC
01390 * {"jsonrpc":"2.0","method":"RuntimeMachine.timerReset","params":["timer"],"id":1}
01391 *
01392 * @par JSON-RPC
01393 * {"id":1,"jsonrpc":"2.0","result":0}
01394 *
01395 * \endchinese
01396 * \english
01397 * Reset the timer
01398 *
01399 * @param name
01400 * @return
01401 *
01402 * @par Python function prototype
01403 * timerReset(self: pyaubo_sdk.RuntimeMachine, arg0: str) -> int
01404 *
01405 * @par Lua function prototype
01406 * timerReset(name: string) -> nil
01407 *
01408 * @par JSON-RPC Request Example
01409 * {"jsonrpc":"2.0","method":"RuntimeMachine.timerReset","params":["timer"],"id":1}
01410 *
01411 * @par JSON-RPC Response Example
01412 * {"id":1,"jsonrpc":"2.0","result":0}
01413 *
01414 * \endenglish
01415 */
01416 int timerReset(const std::string &name);
01417
01418 /**
01419 * @ingroup RuntimeMachine
01420 * \chinese
01421 *
01422 *
01423 * @param name
01424 * @return
01425 *
01426 * @par Python
01427 * timerDelete(self: pyaubo_sdk.RuntimeMachine, arg0: str) -> int
01428 *
01429 * @par Lua
01430 * timerDelete(name: string) -> nil
01431 *
01432 * @par JSON-RPC
01433 * {"jsonrpc":"2.0","method":"RuntimeMachine.timerDelete","params":["timer"],"id":1}
01434 *
01435 * @par JSON-RPC
01436 * {"id":1,"jsonrpc":"2.0","result":0}
01437 *
01438 * \endchinese
01439 * \english
01440 * Delete the timer
01441 *
01442 * @param name
01443 * @return
01444 *
01445 * @par Python function prototype
01446 * timerDelete(self: pyaubo_sdk.RuntimeMachine, arg0: str) -> int
01447 *
01448 * @par Lua function prototype
01449 * timerDelete(name: string) -> nil
01450 *
01451 * @par JSON-RPC Request Example
01452 * {"jsonrpc":"2.0","method":"RuntimeMachine.timerDelete","params":["timer"],"id":1}
01453 *
01454 * @par JSON-RPC Response Example
01455 * {"id":1,"jsonrpc":"2.0","result":0}
01456 *
01457 * \endenglish
01458 */
01459 int timerDelete(const std::string &name);
01460
01461 /**
01462 * @ingroup RuntimeMachine
01463 * \chinese
01464 *
01465 *
01466 * @param name
01467 * @return
01468 *
01469 * @par Python
01470 * getTimer(self: pyaubo_sdk.RuntimeMachine, arg0: str) -> float
01471 *
01472 * @par Lua
01473 * getTimer(name: string) -> number
01474 *

```

```

01475 * @par JSON-RPC
01476 * {"jsonrpc":"2.0","method":"RuntimeMachine.getTimer","params":["timer"],"id":1}
01477 *
01478 * @par JSON-RPC
01479 * {"id":1,"jsonrpc":"2.0","result":25.409769612}
01480 *
01481 * \endchinese
01482 * \english
01483 * Get the timer value
01484 *
01485 * @param name
01486 * @return
01487 *
01488 * @par Python function prototype
01489 * getTimer(self: pyaubo_sdk.RuntimeMachine, arg0: str) -> float
01490 *
01491 * @par Lua function prototype
01492 * getTimer(name: string) -> number
01493 *
01494 * @par JSON-RPC Request Example
01495 * {"jsonrpc":"2.0","method":"RuntimeMachine.getTimer","params":["timer"],"id":1}
01496 *
01497 * @par JSON-RPC Response Example
01498 * {"id":1,"jsonrpc":"2.0","result":25.409769612}
01499 *
01500 * \endenglish
01501 */
01502 double getTimer(const std::string &name);
01503
01504 /**
01505 * @ingroup RuntimeMachine
01506 * \chinese
01507 *
01508 *
01509 * @param distance
01510 * @param delay
01511 * @return
01512 *
01513 * @par JSON-RPC
01514 * {"jsonrpc":"2.0","method":"RuntimeMachine.triggBegin","params":[],"id":1}
01515 *
01516 * @par JSON-RPC
01517 * {"id":1,"jsonrpc":"2.0","result":0}
01518 *
01519 * \endchinese
01520 * \english
01521 * Start configuring trigger
01522 *
01523 * @param distance
01524 * @param delay
01525 * @return
01526 *
01527 * @par JSON-RPC Request Example
01528 * {"jsonrpc":"2.0","method":"RuntimeMachine.triggBegin","params":[],"id":1}
01529 *
01530 * @par JSON-RPC Response Example
01531 * {"id":1,"jsonrpc":"2.0","result":0}
01532 *
01533 * \endenglish
01534 */
01535 int triggBegin(double distance, double delay);
01536
01537 /**
01538 * @ingroup RuntimeMachine
01539 * \chinese
01540 *
01541 *
01542 * @return
01543 *
01544 * @par JSON-RPC
01545 * {"jsonrpc":"2.0","method":"RuntimeMachine.triggEnd","params":[],"id":1}
01546 *
01547 * @par JSON-RPC
01548 * {"id":1,"jsonrpc":"2.0","result":0}
01549 *
01550 * \endchinese
01551 * \english
01552 * End configuring trigger
01553 *
01554 * @return
01555 *
01556 * @par JSON-RPC Request Example
01557 * {"jsonrpc":"2.0","method":"RuntimeMachine.triggEnd","params":[],"id":1}
01558 *
01559 * @par JSON-RPC Response Example
01560 * {"id":1,"jsonrpc":"2.0","result":0}
01561 *

```

```

01562     * \endenglish
01563     */
01564     int triggEnd();
01565
01566     /**
01567     * @ingroup RuntimeMachine
01568     * \chinese
01569     *
01570     *
01571     * @param distance
01572     * @param delay
01573     * @param intnum
01574     * @return
01575     * \endchinese
01576     * \english
01577     * Returns the automatically assigned interrupt number
01578     *
01579     * @param distance
01580     * @param delay
01581     * @param intnum
01582     * @return
01583     * \endenglish
01584     */
01585     int triggInterrupt(double distance, double delay);
01586
01587     /**
01588     * @ingroup RuntimeMachine
01589     * \chinese
01590     *
01591     *
01592     * @return
01593     *
01594     * @par JSON-RPC
01595     * {"jsonrpc":"2.0","method":"RuntimeMachine.getTriggInterrupts","params":[],"id":1}
01596     *
01597     * @par JSON-RPC
01598     * {"id":1,"jsonrpc":"2.0","result":[]}
01599     *
01600     * \endchinese
01601     * \english
01602     * Get the list of all interrupt numbers
01603     *
01604     * @return
01605     *
01606     * @par JSON-RPC Request Example
01607     * {"jsonrpc":"2.0","method":"RuntimeMachine.getTriggInterrupts","params":[],"id":1}
01608     *
01609     * @par JSON-RPC Response Example
01610     * {"id":1,"jsonrpc":"2.0","result":[]}
01611     *
01612     * \endenglish
01613     */
01614     std::vector<int> getTriggInterrupts();
01615
01616 protected:
01617     void *d_;
01618 };
01619
01620 using RuntimeMachinePtr = std::shared_ptr<RuntimeMachine>;
01621
01622 } // namespace common_interface
01623 } // namespace arcs
01624 #endif // AUBO_SDK_RUNTIME_MACHINE_H

```

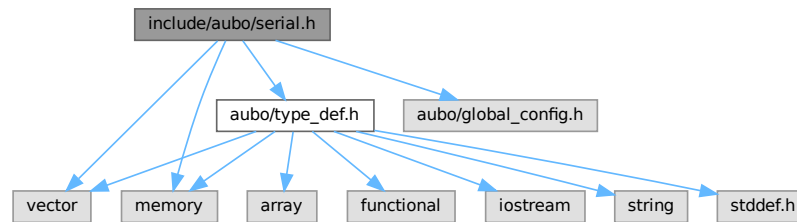
12.38 include/aubo/serial.h File Reference

```

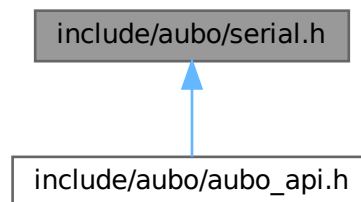
#include <vector>
#include <memory>
#include <aubo/type_def.h>
#include <aubo/global_config.h>

```

Include dependency graph for `serial.h`:



This graph shows which files directly or indirectly include this file:



Classes

- class [arcs::common_interface::Serial](#)

Namespaces

- namespace [arcs](#)
- namespace [arcs::common_interface](#)

Typedefs

- using [arcs::common_interface::SerialPtr](#) = `std::shared_ptr<Serial>`

12.38.1 Detailed Description

Definition in file [serial.h](#).

12.39 serial.h

[Go to the documentation of this file.](#)

```

00001 /** @file serial.h
00002  * @brief
00003  */
00004 #ifndef AUBO_SDK_SERIAL_INTERFACE_H
00005 #define AUBO_SDK_SERIAL_INTERFACE_H
00006
00007 #include <vector>
00008 #include <memory>
00009
00010 #include <aubo/type_def.h>
00011 #include <aubo/global_config.h>
00012
00013 namespace arcs {
00014 namespace common_interface {
00015
00016 /**
00017  * @defgroup Serial Serial ( )
00018  * \~chinese
00019  * \~english Serial control
00020  */
00021 class ARCS_ABI_EXPORT Serial
00022 {
00023 public:
00024     Serial();
00025     virtual ~Serial();
00026
00027     /**
00028      * @ingroup Serial
00029      * \chinese
00030      * TCP/IP
00031      *
00032      * @param device
00033      * @param baud
00034      * @param stop_bits
00035      * @param even
00036      * @param serial_name
00037      * @return
00038      *
00039      * @par Python
00040      * serialOpen(self: pyaubo_sdk.Serial, arg0: str, arg1: int, arg2: float,
00041      * arg3: int, arg4: str) -> int
00042      *
00043      * @par Lua
00044      * serialOpen(device: string, baud: number, stop_bits: number, even: number,
00045      * serial_name: string) -> nil
00046      * \endchinese
00047      * \english
00048      * Open TCP/IP ethernet communication serial
00049      *
00050      * @param device
00051      * @param baud
00052      * @param stop_bits
00053      * @param even
00054      * @param serial_name
00055      * @return
00056      *
00057      * @par Python function prototype
00058      * serialOpen(self: pyaubo_sdk.Serial, arg0: str, arg1: int, arg2: float,
00059      * arg3: int, arg4: str) -> int
00060      *
00061      * @par Lua function prototype
00062      * serialOpen(device: string, baud: number, stop_bits: number, even: number,
00063      * serial_name: string) -> nil
00064      * \endenglish
00065      */
00066
00067     int serialOpen(const std::string &device, int baud, float stop_bits,
00068                   int even, const std::string &serial_name = "serial_0");
00069
00070     /**
00071      * @ingroup Serial
00072      * \chinese
00073      * TCP/IP
00074      *
00075      *
00076      * @param serial_name
00077      * @return
00078      *
00079      * @par Python
00080      * serialClose(self: pyaubo_sdk.Serial, arg0: str) -> int
00081      *
00082      * @par Lua

```

```

00083 * serialClose(serial_name: string) -> nil
00084 * \endchinese
00085 * \english
00086 * Close TCP/IP serial communication
00087 * Close down the serial connection to the server.
00088 *
00089 * @param serial_name
00090 * @return
00091 *
00092 * @par Python function prototype
00093 * serialClose(self: pyaubo_sdk.Serial, arg0: str) -> int
00094 *
00095 * @par Lua function prototype
00096 * serialClose(serial_name: string) -> nil
00097 * \endenglish
00098 */
00099 int serialClose(const std::string &serial_name = "serial_0");
00100
00101 /**
00102 * @ingroup Serial
00103 * \chinese
00104 *          30
00105 *
00106 * @param variable
00107 * @param serial_name
00108 * @return
00109 *
00110 * @par Python
00111 * serialReadByte(self: pyaubo_sdk.Serial, arg0: str, arg1: str) -> int
00112 *
00113 * @par Lua
00114 * serialReadByte(variable: string, serial_name: string) -> number
00115 * \endchinese
00116 * \english
00117 * Reads a number of bytes from the serial. Bytes are in network byte
00118 * order. A maximum of 30 values can be read in one command.
00119 *
00120 * @param variable
00121 * @param serial_name
00122 * @return
00123 *
00124 * @par Python function prototype
00125 * serialReadByte(self: pyaubo_sdk.Serial, arg0: str, arg1: str) -> int
00126 *
00127 * @par Lua function prototype
00128 * serialReadByte(variable: string, serial_name: string) -> number
00129 * \endenglish
00130 */
00131 int serialReadByte(const std::string &variable,
00132                   const std::string &serial_name = "serial_0");
00133
00134 /**
00135 * @ingroup Serial
00136 * \chinese
00137 *          30
00138 *          int  =number+1
00139 *
00140 * @param number
00141 * @param variable
00142 * @param serial_name
00143 * @return
00144 *
00145 * @par Python
00146 * serialReadByteList(self: pyaubo_sdk.Serial, arg0: int, arg1: str, arg2: str) -> int
00147 *
00148 * @par Lua
00149 * serialReadByteList(number: number, variable: string, serial_name: string) -> number
00150 * \endchinese
00151 * \english
00152 * Reads a number of bytes from the serial. Bytes are in network byte
00153 * order. A maximum of 30 values can be read in one command.
00154 * A list of numbers read (list of ints, length=number+1)
00155 *
00156 * @param number Number of bytes to read
00157 * @param variable
00158 * @param serial_name Serial port name
00159 * @return Return value
00160 *
00161 * @par Python function prototype
00162 * serialReadByteList(self: pyaubo_sdk.Serial, arg0: int, arg1: str, arg2: str) -> int
00163 *
00164 * @par Lua function prototype
00165 * serialReadByteList(number: number, variable: string, serial_name: string) -> number
00166 * \endenglish
00167 */
00168 int serialReadByteList(int number, const std::string &variable,
00169                       const std::string &serial_name = "serial_0");

```



```

00170
00171 /**
00172  * @ingroup Serial
00173  * \chinese
00174  *
00175  *
00176  *
00177  * "prefix" "suffix"
00178  * "prefix" "prefix"
00179  * "suffix" "suffix"
00180  * "noise>hello<" prefix=">" suffix="<" "hello"
00181  * "prefix" "suffix" "suffix"
00182  * ">hello<>world<"
00183  *
00184  * @param variable
00185  * @param serial_name
00186  * @param prefix
00187  * @param suffix
00188  * @param interpret_escape
00189  * @return
00190  *
00191  * @par Python
00192  * serialReadString(self: pyaubo_sdk.Serial, arg0: str, arg1: str, arg2: str, arg3: str, arg4: bool) -> int
00193  *
00194  * @par Lua
00195  * serialReadString(variable: string, serial_name: string, prefix: string, suffix: string, interpret_escape: boolean) ->
number
00196  * \endchinese
00197  * \english
00198  * Reads all data from the serial and returns the data as a string.
00199  * Bytes are in network byte order.
00200  *
00201  * The optional parameters "prefix" and "suffix", can be used to express
00202  * what is extracted from the serial. The "prefix" specifies the start
00203  * of the substring (message) extracted from the serial. The data up to
00204  * the end of the "prefix" will be ignored and removed from the serial.
00205  * The "suffix" specifies the end of the substring (message) extracted
00206  * from the serial. Any remaining data on the serial, after the "suffix",
00207  * will be preserved. E.g. if the serial server sends a string
00208  * "noise>hello<", the controller can receive the "hello" by calling this
00209  * script function with the prefix=">" and suffix="<". By using the
00210  * "prefix" and "suffix" it is also possible send multiple string to the
00211  * controller at once, because the suffix defines where the message ends.
00212  * E.g. sending ">hello<>world<"
00213  *
00214  * @param variable
00215  * @param serial_name
00216  * @param prefix
00217  * @param suffix
00218  * @param interpret_escape
00219  * @return
00220  *
00221  * @par Python function prototype
00222  * serialReadString(self: pyaubo_sdk.Serial, arg0: str, arg1: str, arg2: str, arg3: str, arg4: bool) -> int
00223  *
00224  * @par Lua function prototype
00225  * serialReadString(variable: string, serial_name: string, prefix: string, suffix: string, interpret_escape: boolean) ->
number
00226  * \endenglish
00227  */
00228 int serialReadString(const std::string &variable,
00229                     const std::string &serial_name = "serial_0",
00230                     const std::string &prefix = "",
00231                     const std::string &suffix = "",
00232                     bool interpret_escape = false);
00233
00234 /**
00235  * @ingroup Serial
00236  * \chinese
00237  *
00238  * <value> ASCII 10 2 3
00239  *
00240  * @param value
00241  * @param serial_name
00242  * @return
00243  *
00244  * @par Python
00245  * serialSendByte(self: pyaubo_sdk.Serial, arg0: str, arg1: str) -> int
00246  *
00247  * @par Lua
00248  * serialSendByte(value: string, serial_name: string) -> nil
00249  * \endchinese
00250  * \english
00251  * Sends a byte to the server
00252  * Sends the byte <value> through the serial. Expects no response. Can
00253  * be used to send special ASCII characters; 10 is newline, 2 is start of
00254  * text, 3 is end of text.

```

```

00255 *
00256 * @param value
00257 * @param serial_name
00258 * @return
00259 *
00260 * @par Python function prototype
00261 * serialSendByte(self: pyaubo_sdk.Serial, arg0: str, arg1: str) -> int
00262 *
00263 * @par Lua function prototype
00264 * serialSendByte(value: string, serial_name: string) -> nil
00265 * \endenglish
00266 */
00267 int serialSendByte(char value, const std::string &serial_name = "serial_0");
00268
00269 /**
00270 * @ingroup Serial
00271 * \chinese
00272 *     int32_t
00273 *     <value>
00274 *
00275 * @param value
00276 * @param serial_name
00277 * @return
00278 *
00279 * @par Python
00280 * serialSendInt(self: pyaubo_sdk.Serial, arg0: int, arg1: str) -> int
00281 *
00282 * @par Lua
00283 * serialSendInt(value: number, serial_name: string) -> nil
00284 * \endchinese
00285 * \english
00286 * Sends an int (int32_t) to the server
00287 * Sends the int <value> through the serial. Send in network byte order.
00288 * Expects no response.
00289 *
00290 * @param value
00291 * @param serial_name
00292 * @return
00293 *
00294 * @par Python function prototype
00295 * serialSendInt(self: pyaubo_sdk.Serial, arg0: int, arg1: str) -> int
00296 *
00297 * @par Lua function prototype
00298 * serialSendInt(value: number, serial_name: string) -> nil
00299 * \endenglish
00300 */
00301 int serialSendInt(int value, const std::string &serial_name = "serial_0");
00302
00303 /**
00304 * @ingroup Serial
00305 * \chinese
00306 *
00307 *     ASCII      <str>
00308 *
00309 * @param str
00310 * @param serial_name
00311 * @return
00312 *
00313 * @par Python
00314 * serialSendLine(self: pyaubo_sdk.Serial, arg0: str, arg1: str) -> int
00315 *
00316 * @par Lua
00317 * serialSendLine(str: string, serial_name: string) -> nil
00318 * \endchinese
00319 * \english
00320 * Sends a string with a newline character to the server
00321 * Sends the string <str> through the serial in ASCII coding, appending a newline at the end. Expects no response.
00322 *
00323 * @param str
00324 * @param serial_name
00325 * @return
00326 *
00327 * @par Python function prototype
00328 * serialSendLine(self: pyaubo_sdk.Serial, arg0: str, arg1: str) -> int
00329 *
00330 * @par Lua function prototype
00331 * serialSendLine(str: string, serial_name: string) -> nil
00332 * \endenglish
00333 */
00334 int serialSendLine(const std::string &str,
00335                   const std::string &serial_name = "serial_0");
00336
00337 /**
00338 * @ingroup Serial
00339 * \chinese
00340 *
00341 *     ASCII      <str>

```

```

00342  *
00343  * @param str
00344  * @param serial_name
00345  * @return
00346  *
00347  * @par Python
00348  * serialSendString(self: pyaubo_sdk.Serial, arg0: str, arg1: str) -> int
00349  *
00350  * @par Lua
00351  * serialSendString(str: string, serial_name: string) -> nil
00352  * \endchinese
00353  * \english
00354  * Sends a string to the server
00355  * Sends the string <str> through the serial in ASCII coding. Expects no
00356  * response.
00357  *
00358  * @param str
00359  * @param serial_name
00360  * @return
00361  *
00362  * @par Python function prototype
00363  * serialSendString(self: pyaubo_sdk.Serial, arg0: str, arg1: str) -> int
00364  *
00365  * @par Lua function prototype
00366  * serialSendString(str: string, serial_name: string) -> nil
00367  * \endenglish
00368  */
00369 int serialSendString(const std::string &str,
00370                     const std::string &serial_name = "serial_0");
00371
00372 /**
00373  * @ingroup Serial
00374  * \chinese
00375  *
00376  * @param is_check
00377  * @param str
00378  * @param serial_name
00379  * @return
00380  *
00381  * @par Python
00382  * serialSendAllString(self: pyaubo_sdk.Serial, arg0: bool, arg1: List[str], arg2: str) -> int
00383  *
00384  * @par Lua
00385  * serialSendAllString(is_check: boolean, str: table, serial_name: string) -> nil
00386  * \endchinese
00387  * \english
00388  *
00389  * @param is_check Whether to check
00390  * @param str Array of strings
00391  * @param serial_name Serial port name
00392  * @return Return value
00393  *
00394  * @par Python function prototype
00395  * serialSendAllString(self: pyaubo_sdk.Serial, arg0: bool, arg1: List[str], arg2: str) -> int
00396  *
00397  * @par Lua function prototype
00398  * serialSendAllString(is_check: boolean, str: table, serial_name: string) -> nil
00399  * \endenglish
00400  */
00401 int serialSendAllString(bool is_check, const std::vector<char> &str,
00402                        const std::string &serial_name = "serial_0");
00403
00404 protected:
00405 void *d_;
00406 };
00407 using SerialPtr = std::shared_ptr<Serial>;
00408
00409 } // namespace common_interface
00410 } // namespace arcs
00411 #endif

```

12.40 include/aubo/socket.h File Reference

socket

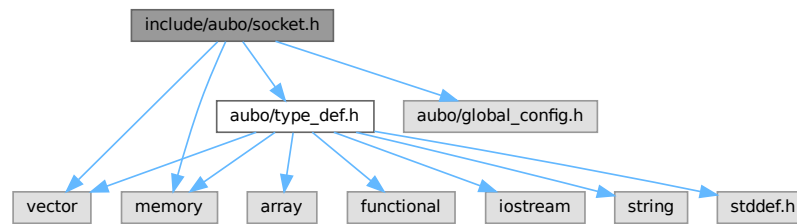
```

#include <vector>
#include <memory>
#include <aubo/type_def.h>

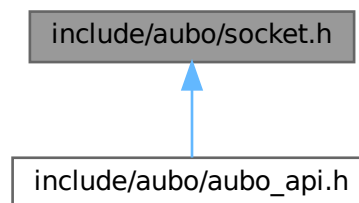
```

```
#include <aubo/global_config.h>
```

Include dependency graph for socket.h:



This graph shows which files directly or indirectly include this file:



Classes

- class [arcs::common_interface::Socket](#)

Namespaces

- namespace [arcs](#)
- namespace [arcs::common_interface](#)

Typedefs

- using [arcs::common_interface::SocketPtr](#) = std::shared_ptr<[Socket](#)>

12.40.1 Detailed Description

socket

Definition in file [socket.h](#).

12.41 socket.h

[Go to the documentation of this file.](#)

```

00001 /** @file socket.h
00002  * @brief socket
00003  */
00004 #ifndef AUBO_SDK_SOCKET_INTERFACE_H
00005 #define AUBO_SDK_SOCKET_INTERFACE_H
00006
00007 #include <vector>
00008 #include <memory>
00009
00010 #include <aubo/type_def.h>
00011 #include <aubo/global_config.h>
00012
00013 namespace arcs {
00014 namespace common_interface {
00015
00016
00017 /**
00018  * @defgroup Socket Socket (socket )
00019  * \-chinese
00020  * \-english Socket control
00021  */
00022 class ARCS_ABI_EXPORT Socket
00023 {
00024 public:
00025     Socket();
00026     virtual ~Socket();
00027
00028     /**
00029      * @ingroup Socket
00030      * \english
00031      * Open TCP/IP ethernet communication socket
00032      *
00033      * Instruction
00034      *
00035      * @param address
00036      * @param port
00037      * @param socket_name
00038      * @return
00039      *
00040      * @par Python function prototype
00041      * socketOpen(self: pyaubo_sdk.Socket, arg0: str, arg1: int, arg2: str) -> int
00042      *
00043      * @par Lua function prototype
00044      * socketOpen(address: string, port: number, socket_name: string) -> nil
00045      *
00046      * @par JSON-RPC request example
00047      * {"jsonrpc":"2.0","method":"Socket.socketOpen","params":["172.16.26.248",8000,"socket_0"],"id":1}
00048      *
00049      * @par JSON-RPC response example
00050      * {"id":1,"jsonrpc":"2.0","result":0}
00051      * \endenglish
00052      * \chinese
00053      * TCP/IP socket
00054      *
00055      *
00056      *
00057      * @param address
00058      * @param port
00059      * @param socket_name
00060      * @return
00061      *
00062      * @par Python
00063      * socketOpen(self: pyaubo_sdk.Socket, arg0: str, arg1: int, arg2: str) -> int
00064      *
00065      * @par Lua
00066      * socketOpen(address: string, port: number, socket_name: string) -> nil
00067      *
00068      * @par JSON-RPC
00069      * {"jsonrpc":"2.0","method":"Socket.socketOpen","params":["172.16.26.248",8000,"socket_0"],"id":1}
00070      *
00071      * @par JSON-RPC
00072      * {"id":1,"jsonrpc":"2.0","result":0}
00073      * \endchinese
00074      */
00075     int socketOpen(const std::string &address, int port,
00076                   const std::string &socket_name = "socket_0");
00077
00078     /**
00079      * @ingroup Socket
00080      * \english
00081      * Closes TCP/IP socket communication
00082      * Closes down the socket connection to the server.

```

```

00083  *
00084  * Instruction
00085  *
00086  * @param socket_name
00087  * @return
00088  *
00089  * @par Python function prototype
00090  * socketClose(self: pyaubo_sdk.Socket, arg0: str) -> int
00091  *
00092  * @par Lua function prototype
00093  * socketClose(socket_name: string) -> nil
00094  *
00095  * @par JSON-RPC request example
00096  * {"jsonrpc":"2.0","method":"Socket.socketClose","params":["socket_0"],"id":1}
00097  *
00098  * @par JSON-RPC response example
00099  * {"id":1,"jsonrpc":"2.0","result":0}
00100  * \endenglish
00101  * \chinese
00102  * TCP/IP socket
00103  *     socket
00104  *
00105  *
00106  *
00107  * @param socket_name
00108  * @return
00109  *
00110  * @par Python
00111  * socketClose(self: pyaubo_sdk.Socket, arg0: str) -> int
00112  *
00113  * @par Lua
00114  * socketClose(socket_name: string) -> nil
00115  *
00116  * @par JSON-RPC
00117  * {"jsonrpc":"2.0","method":"Socket.socketClose","params":["socket_0"],"id":1}
00118  *
00119  * @par JSON-RPC
00120  * {"id":1,"jsonrpc":"2.0","result":0}
00121  * \endchinese
00122  */
00123 int socketClose(const std::string &socket_name = "socket_0");
00124
00125 /**
00126  * @ingroup Socket
00127  * \english
00128  * Reads a number of ascii formatted floats from the socket. A maximum
00129  * of 30 values can be read in one command.
00130  * A list of numbers read (list of floats, length=number+1)
00131  *
00132  * Result will be stored in a register named reg_key. Use getFloatVec
00133  * to retrieve data
00134  *
00135  * @param number
00136  * @param variable
00137  * @param socket_name
00138  * @return
00139  *
00140  * @par Python function prototype
00141  * socketReadAsciiFloat(self: pyaubo_sdk.Socket, arg0: int, arg1: str, arg2:
00142  * str) -> int
00143  *
00144  * @par Lua function prototype
00145  * socketReadAsciiFloat(number: number, variable: string, socket_name:
00146  * string) -> number
00147  * \endenglish
00148  * \chinese
00149  * socket    ASCII    30
00150  *           =number+1
00151  *
00152  *     reg_key    getFloatVec
00153  *
00154  * @param number
00155  * @param variable
00156  * @param socket_name
00157  * @return
00158  *
00159  * @par Python
00160  * socketReadAsciiFloat(self: pyaubo_sdk.Socket, arg0: int, arg1: str, arg2:
00161  * str) -> int
00162  *
00163  * @par Lua
00164  * socketReadAsciiFloat(number: number, variable: string, socket_name:
00165  * string) -> number
00166  * \endchinese
00167  */
00168 int socketReadAsciiFloat(int number, const std::string &variable,
00169                          const std::string &socket_name = "socket_0");

```

```

00170
00171 /**
00172  * @ingroup Socket
00173  * \english
00174  * Reads a number of 32 bit integers from the socket. Bytes are in
00175  * network byte order. A maximum of 30 values can be read in one
00176  * command.
00177  * A list of numbers read (list of ints, length=number+1)
00178  *
00179  * Instruction
00180  *
00181  * std::vector<int>
00182  *
00183  * @param number
00184  * @param variable
00185  * @param socket_name
00186  * @return
00187  *
00188  * @par Python function prototype
00189  * socketReadBinaryInteger(self: pyaubo_sdk.Socket, arg0: int, arg1: str,
00190  * arg2: str) -> int
00191  *
00192  * @par Lua function prototype
00193  * socketReadBinaryInteger(number: number, variable: string, socket_name:
00194  * string) -> number
00195  * \endenglish
00196  * \chinese
00197  * socket      32      30
00198  *           =number+1
00199  *
00200  *
00201  *
00202  * std::vector<int>
00203  *
00204  * @param number
00205  * @param variable
00206  * @param socket_name
00207  * @return
00208  *
00209  * @par Python
00210  * socketReadBinaryInteger(self: pyaubo_sdk.Socket, arg0: int, arg1: str,
00211  * arg2: str) -> int
00212  *
00213  * @par Lua
00214  * socketReadBinaryInteger(number: number, variable: string, socket_name:
00215  * string) -> number
00216  * \endchinese
00217  */
00218 int socketReadBinaryInteger(int number, const std::string &variable,
00219                             const std::string &socket_name = "socket_0");
00220
00221 /**
00222  * @ingroup Socket
00223  * \english
00224  * Reads a number of bytes from the socket. Bytes are in network byte
00225  * order. A maximum of 30 values can be read in one command.
00226  * A list of numbers read (list of ints, length=number+1)
00227  *
00228  * Instruction
00229  *
00230  * std::vector<char>
00231  *
00232  * @param number
00233  * @param variable
00234  * @param socket_name
00235  * @return
00236  *
00237  * @par Python function prototype
00238  * socketReadByteList(self: pyaubo_sdk.Socket, arg0: int, arg1: str, arg2:
00239  * str) -> int
00240  *
00241  * @par Lua function prototype
00242  * socketReadByteList(number: number, variable: string, socket_name: string)
00243  * -> number
00244  * \endenglish
00245  * \chinese
00246  * socket      30
00247  *           =number+1
00248  *
00249  *
00250  *
00251  * std::vector<char>
00252  *
00253  * @param number
00254  * @param variable
00255  * @param socket_name
00256  * @return

```

```

00257 *
00258 * @par Python
00259 * socketReadByteList(self: pyaubo_sdk.Socket, arg0: int, arg1: str, arg2:
00260 * str) -> int
00261 *
00262 * @par Lua
00263 * socketReadByteList(number: number, variable: string, socket_name: string)
00264 * -> number
00265 * \endchinese
00266 */
00267 int socketReadByteList(int number, const std::string &variable,
00268                        const std::string &socket_name = "socket_0");
00269
00270 /**
00271 * @ingroup Socket
00272 * \english
00273 * Reads all data from the socket and returns the data as a string.
00274 * Bytes are in network byte order.
00275 *
00276 * The optional parameters "prefix" and "suffix", can be used to express
00277 * what is extracted from the socket. The "prefix" specifies the start
00278 * of the substring (message) extracted from the socket. The data up to
00279 * the end of the "prefix" will be ignored and removed from the socket.
00280 * The "suffix" specifies the end of the substring (message) extracted
00281 * from the socket. Any remaining data on the socket, after the "suffix",
00282 * will be preserved. E.g. if the socket server sends a string
00283 * "noise>hello<", the controller can receive the "hello" by calling this
00284 * script function with the prefix=">" and suffix="<". By using the
00285 * "prefix" and "suffix" it is also possible send multiple string to the
00286 * controller at once, because the suffix defines where the message ends.
00287 * E.g. sending ">hello<>world<"
00288 *
00289 * Instruction
00290 *
00291 * std::string
00292 *
00293 * @param variable
00294 * @param socket_name
00295 * @param prefix
00296 * @param suffix
00297 * @param interpret_escape
00298 * @return
00299 *
00300 * @par Python function prototype
00301 * socketReadString(self: pyaubo_sdk.Socket, arg0: str, arg1: str, arg2:
00302 * str, arg3: str, arg4: bool) -> int
00303 *
00304 * @par Lua function prototype
00305 * socketReadString(variable: string, socket_name: string, prefix: string,
00306 * suffix: string, interpret_escape: boolean) -> number
00307 *
00308 * @par JSON-RPC request example
00309 * {"jsonrpc": "2.0", "method": "Socket.socketReadString", "params": ["camera", "socket_0", "", "", false], "id": 1}
00310 *
00311 * @par JSON-RPC response example
00312 * {"id": 1, "jsonrpc": "2.0", "result": 0}
00313 * \endenglish
00314 * \chinese
00315 * socket
00316 *
00317 *
00318 * "prefix" "suffix" socket
00319 * "prefix" "prefix" socket
00320 * "suffix" "suffix" socket
00321 * socket "noise>hello<" prefix=">" suffix="<" "hello"
00322 * "prefix" "suffix" "suffix" ">hello<>world<"
00323 *
00324 *
00325 *
00326 * std::string
00327 *
00328 * @param variable
00329 * @param socket_name
00330 * @param prefix
00331 * @param suffix
00332 * @param interpret_escape
00333 * @return
00334 *
00335 * @par Python
00336 * socketReadString(self: pyaubo_sdk.Socket, arg0: str, arg1: str, arg2:
00337 * str, arg3: str, arg4: bool) -> int
00338 *
00339 * @par Lua
00340 * socketReadString(variable: string, socket_name: string, prefix: string,
00341 * suffix: string, interpret_escape: boolean) -> number
00342 *
00343 * @par JSON-RPC

```



```

00344 * {"jsonrpc":"2.0","method":"Socket.socketReadString","params":["camera","socket_0","","",false],"id":1}
00345 *
00346 * @par JSON-RPC
00347 * {"id":1,"jsonrpc":"2.0","result":0}
00348 * \endchinese
00349 */
00350 int socketReadString(const std::string &variable,
00351                     const std::string &socket_name = "socket_0",
00352                     const std::string &prefix = "",
00353                     const std::string &suffix = "",
00354                     bool interpret_escape = false);
00355
00356 /**
00357 * @ingroup Socket
00358 * \english
00359 * Reads all data from the socket and returns the data as a vector of chars.
00360 *
00361 * Instruction
00362 * std::vector<char>
00363 *
00364 * @param variable
00365 * @param socket_name
00366 * @return
00367 *
00368 * @par Python function prototype
00369 * socketReadAllString(self: pyaubo_sdk.Socket, arg0: str, arg1: str) -> int
00370 *
00371 * @par Lua function prototype
00372 * socketReadAllString(variable: string, socket_name: string) -> number
00373 *
00374 * @par JSON-RPC request example
00375 * {"jsonrpc":"2.0","method":"Socket.socketReadAllString","params":["camera","socket_0"],"id":1}
00376 *
00377 * @par JSON-RPC response example
00378 * {"id":1,"jsonrpc":"2.0","result":0}
00379 * \endenglish
00380 * \chinese
00381 * socket      char
00382 *
00383 *
00384 * std::vector<char>
00385 *
00386 * @param variable
00387 * @param socket_name
00388 * @return
00389 *
00390 * @par Python
00391 * socketReadAllString(self: pyaubo_sdk.Socket, arg0: str, arg1: str) -> int
00392 *
00393 * @par Lua
00394 * socketReadAllString(variable: string, socket_name: string) -> number
00395 *
00396 * @par JSON-RPC
00397 * {"jsonrpc":"2.0","method":"Socket.socketReadAllString","params":["camera","socket_0"],"id":1}
00398 *
00399 * @par JSON-RPC
00400 * {"id":1,"jsonrpc":"2.0","result":0}
00401 * \endchinese
00402 */
00403 int socketReadAllString(const std::string &variable,
00404                         const std::string &socket_name = "socket_0");
00405
00406 /**
00407 * @ingroup Socket
00408 * \english
00409 * Sends a byte to the server
00410 * Sends the byte <value> through the socket. Expects no response. Can
00411 * be used to send special ASCII characters; 10 is newline, 2 is start of
00412 * text, 3 is end of text.
00413 *
00414 * Instruction
00415 *
00416 * @param value
00417 * @param socket_name
00418 * @return
00419 *
00420 * @par Python function prototype
00421 * socketSendByte(self: pyaubo_sdk.Socket, arg0: str, arg1: str) -> int
00422 *
00423 * @par Lua function prototype
00424 * socketSendByte(value: string, socket_name: string) -> nil
00425 *
00426 * \endenglish
00427 * \chinese
00428 *
00429 * socket  <value>          ASCII 10  2  3
00430 *

```

```

00431 *
00432 *
00433 * @param value
00434 * @param socket_name
00435 * @return
00436 *
00437 * @par Python
00438 * socketSendByte(self: pyaubo_sdk.Socket, arg0: str, arg1: str) -> int
00439 *
00440 * @par Lua
00441 * socketSendByte(value: string, socket_name: string) -> nil
00442 *
00443 * \endchinese
00444 */
00445 int socketSendByte(char value, const std::string &socket_name = "socket_0");
00446
00447 /**
00448 * @ingroup Socket
00449 * \english
00450 * Sends an int (int32_t) to the server
00451 * Sends the int <value> through the socket. Send in network byte order.
00452 * Expects no response
00453 *
00454 * Instruction
00455 *
00456 * @param value
00457 * @param socket_name
00458 * @return
00459 *
00460 * @par Python function prototype
00461 * socketSendInt(self: pyaubo_sdk.Socket, arg0: int, arg1: str) -> int
00462 *
00463 * @par Lua function prototype
00464 * socketSendInt(value: number, socket_name: string) -> nil
00465 *
00466 * \endenglish
00467 * \chinese
00468 *     int int32_t
00469 *     socket int <value>
00470 *
00471 *
00472 *
00473 * @param value
00474 * @param socket_name
00475 * @return
00476 *
00477 * @par Python
00478 * socketSendInt(self: pyaubo_sdk.Socket, arg0: int, arg1: str) -> int
00479 *
00480 * @par Lua
00481 * socketSendInt(value: number, socket_name: string) -> nil
00482 *
00483 * \endchinese
00484 */
00485 int socketSendInt(int value, const std::string &socket_name = "socket_0");
00486
00487 /**
00488 * @ingroup Socket
00489 * \english
00490 * Sends a string with a newline character to the server
00491 * Sends the string <str> through the socket in ASCII coding. Expects no
00492 * response.
00493 *
00494 * Instruction
00495 *
00496 * @param str
00497 * @param socket_name
00498 * @return
00499 *
00500 * @par Python function prototype
00501 * socketSendLine(self: pyaubo_sdk.Socket, arg0: str, arg1: str) -> int
00502 *
00503 * @par Lua function prototype
00504 * socketSendLine(str: string, socket_name: string) -> nil
00505 *
00506 * @par JSON-RPC request example
00507 * {"jsonrpc":"2.0","method":"Socket.socketSendLine","params":["abcd","socket_0"],"id":1}
00508 *
00509 * @par JSON-RPC response example
00510 * {"id":1,"jsonrpc":"2.0","result":0}
00511 * \endenglish
00512 * \chinese
00513 *
00514 *     socket ASCII     <str>
00515 *
00516 *
00517 *

```

```

00518 * @param str
00519 * @param socket_name
00520 * @return
00521 *
00522 * @par Python
00523 * socketSendLine(self: pyaubo_sdk.Socket, arg0: str, arg1: str) -> int
00524 *
00525 * @par Lua
00526 * socketSendLine(str: string, socket_name: string) -> nil
00527 *
00528 * @par JSON-RPC
00529 * {"jsonrpc":"2.0","method":"Socket.socketSendLine","params":["abcd","socket_0"],"id":1}
00530 *
00531 * @par JSON-RPC
00532 * {"id":1,"jsonrpc":"2.0","result":0}
00533 * \endchinese
00534 */
00535 int socketSendLine(const std::string &str,
00536                   const std::string &socket_name = "socket_0");
00537
00538 /**
00539 * @ingroup Socket
00540 * \english
00541 * Sends a string to the server
00542 * Sends the string <str> through the socket in ASCII coding. Expects no
00543 * response.
00544 *
00545 * Instruction
00546 *
00547 * @param str
00548 * @param socket_name
00549 * @return
00550 *
00551 * @par Python function prototype
00552 * socketSendString(self: pyaubo_sdk.Socket, arg0: str, arg1: str) -> int
00553 *
00554 * @par Lua function prototype
00555 * socketSendString(str: string, socket_name: string) -> nil
00556 *
00557 * @par JSON-RPC request example
00558 * {"jsonrpc":"2.0","method":"Socket.socketSendString","params":["abcd","socket_0"],"id":1}
00559 *
00560 * @par JSON-RPC response example
00561 * {"id":1,"jsonrpc":"2.0","result":0}
00562 * \endenglish
00563 * \chinese
00564 *
00565 * socket ASCII <str>
00566 *
00567 *
00568 *
00569 * @param str
00570 * @param socket_name
00571 * @return
00572 *
00573 * @par Python
00574 * socketSendString(self: pyaubo_sdk.Socket, arg0: str, arg1: str) -> int
00575 *
00576 * @par Lua
00577 * socketSendString(str: string, socket_name: string) -> nil
00578 *
00579 * @par JSON-RPC
00580 * {"jsonrpc":"2.0","method":"Socket.socketSendString","params":["abcd","socket_0"],"id":1}
00581 *
00582 * @par JSON-RPC
00583 * {"id":1,"jsonrpc":"2.0","result":0}
00584 * \endchinese
00585 */
00586 int socketSendString(const std::string &str,
00587                     const std::string &socket_name = "socket_0");
00588
00589 /**
00590 * @ingroup Socket
00591 * \english
00592 * Sends all data in the given vector of chars to the server.
00593 *
00594 * @param is_check Whether to check the sending status
00595 * @param str The data to send as a vector of chars
00596 * @param socket_name The name of the socket
00597 * @return Status code
00598 *
00599 * @par Python function prototype
00600 * socketSendAllString(self: pyaubo_sdk.Socket, arg0: bool, arg1: List[str], arg2: str) -> int
00601 *
00602 * @par Lua function prototype
00603 * socketSendAllString(is_check: boolean, str: table, socket_name: string) -> nil
00604 * \endenglish

```

```

00605 * \chinese
00606 *   char
00607 *
00608 * @param is_check
00609 * @param str      char
00610 * @param socket_name
00611 * @return
00612 *
00613 * @par Python
00614 * socketSendAllString(self: pyaubo_sdk.Socket, arg0: bool, arg1: List[str], arg2: str) -> int
00615 *
00616 * @par Lua
00617 * socketSendAllString(is_check: boolean, str: table, socket_name: string) -> nil
00618 * \endchinese
00619 */
00620 int socketSendAllString(bool is_check, const std::vector<char> &str,
00621                        const std::string &socket_name = "socket_0");
00622
00623 /**
00624 * @ingroup Socket
00625 * \~chinese socket \~english Check if the socket is connected
00626 * @brief socketHasConnected
00627 * @param socket_name
00628 *
00629 * @return
00630 *
00631 * @par Python
00632 * socketHasConnected(self: pyaubo_sdk.Socket, arg0: str) -> bool
00633 *
00634 * @par Lua
00635 * socketHasConnected(socket_name: string) -> boolean
00636 */
00637 bool socketHasConnected(const std::string &socket_name = "socket_0");
00638
00639 protected:
00640     void *d_;
00641 };
00642 using SocketPtr = std::shared_ptr<Socket>;
00643
00644 } // namespace common_interface
00645 } // namespace arcs
00646 #endif

```

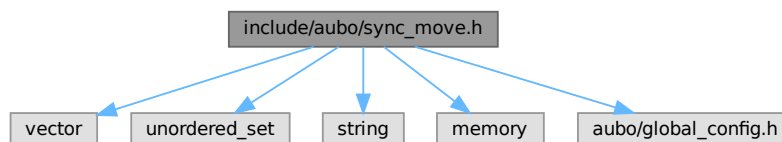
12.42 include/aubo/sync_move.h File Reference

```

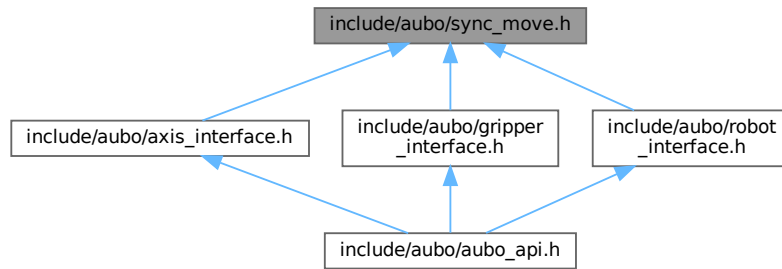
#include <vector>
#include <unordered_set>
#include <string>
#include <memory>
#include <aubo/global_config.h>

```

Include dependency graph for sync_move.h:



This graph shows which files directly or indirectly include this file:



Classes

- class [arcs::common_interface::SyncMove](#)

Namespaces

- namespace [arcs](#)
- namespace [arcs::common_interface](#)

Typedefs

- typedef `std::unordered_set< std::string >` [arcs::common_interface::TaskSet](#)
- using [arcs::common_interface::SyncMovePtr](#) = `std::shared_ptr<SyncMove>`

12.42.1 Detailed Description

1. Independent movements If the different task programs, and their robots, work independently, no synchronization or coordination system is needed. Each task program is then written as if it was the program for a single robot system.
2. Semi coordinated movements Several robots can work with the same work object, without synchronized movements, as long as the work object is not moving. A positioner can move the work object when the robots are not coordinated to it, and the robots can be coordinated to the work object when it is not moving. Switching between moving the object and coordinating the robots is called semi coordinated movements.
3. Coordinated synchronized movements Several robots can work with the same moving work object. The positioner or robot that holds the work object and the robots that work with the work object must have synchronized movements. This means that the RAPID task programs, that handle one mechanical unit each, execute their move instructions simultaneously.

Definition in file [sync_move.h](#).

12.43 sync_move.h

[Go to the documentation of this file.](#)

```

00001 /** @file sync_move.h
00002  * @brief
00003  *
00004  * 1. Independent movements
00005  * If the different task programs, and their robots, work independently, no
00006  * synchronization or coordination is needed. Each task program is then
00007  * written as if it was the program for a single robot system.
00008  *
00009  * 2. Semi coordinated movements
00010  * Several robots can work with the same work object, without synchronized
00011  * movements, as long as the work object is not moving.
00012  * A positioner can move the work object when the robots are not coordinated
00013  * to it, and the robots can be coordinated to the work object when it is not
00014  * moving. Switching between moving the object and coordinating the robots is
00015  * called semi coordinated movements.
00016  *
00017  * 3. Coordinated synchronized movements
00018  * Several robots can work with the same moving work object.
00019  * The positioner or robot that holds the work object and the robots that work
00020  * with the work object must have synchronized movements. This means that the
00021  * RAPID task programs, that handle one mechanical unit each, execute their
00022  * move instructions simultaneously.
00023  */
00024 #ifndef AUBO_SDK_SYNC_MOVE_INTERFACE_H
00025 #define AUBO_SDK_SYNC_MOVE_INTERFACE_H
00026
00027 #include <vector>
00028 #include <unordered_set>
00029 #include <string>
00030 #include <memory>
00031 #include <aubo/global_config.h>
00032
00033 namespace arcs {
00034 namespace common_interface {
00035
00036 typedef std::unordered_set<std::string> TaskSet;
00037
00038 /**
00039  * @defgroup SyncMove SyncMove ( )
00040  * \~chinese API
00041  */
00042 class ARCS_ABI_EXPORT SyncMove
00043 {
00044 public:
00045     SyncMove();
00046     virtual ~SyncMove();
00047
00048     /**
00049      * @ingroup SyncMove
00050      * \chinese
00051      * syncMoveOn
00052      *
00053      * syncMoveOn          syncMoveOn
00054      *
00055      * syncMoveOff      syncMoveOn
00056      *
00057      * @param syncident
00058      * @param taskset
00059      * @return
00060      *
00061      * @par Python
00062      * syncMoveOn(self: pyaubo_sdk.SyncMove, arg0: str, arg1: Set[str]) -> int
00063      *
00064      * @par Lua
00065      * syncMoveOn(syncident: string, taskset: table) -> nil
00066      * @endcoe
00067      * \endchinese
00068      * \english
00069      * syncMoveOn is used to start synchronized movement mode.
00070      *
00071      * A syncMoveOn instruction will wait for the other task programs. When
00072      * all task programs have reached the syncMoveOn, they will continue
00073      * their execution in synchronized movement mode. The move instructions
00074      * in the different task programs are executed simultaneously, until the
00075      * instruction syncMoveOff is executed.
00076      * A stop point must be programmed before the syncMoveOn instruction.
00077      *
00078      * @param syncident
00079      * @param taskset
00080      * @return
00081      *
00082      * @par Python function prototype

```

```

00083     * syncMoveOn(self: pyaubo_sdk.SyncMove, arg0: str, arg1: Set[str]) -> int
00084     *
00085     * @par Lua function prototype
00086     * syncMoveOn(syncident: string, taskset: table) -> nil
00087     * @endcoe
00088     * \endenglish
00089     */
00090     int syncMoveOn(const std::string &syncident, const TaskSet &taskset);
00091
00092     /**
00093     * @ingroup SyncMove
00094     * \chinese
00095     *     ID
00096     *     corner zones    stop points
00097     *     ID
00098     *
00099     *     z10    z50
00100     *
00101     * @param id
00102     * @return
00103     *
00104     * @par Python
00105     * syncMoveSegment(self: pyaubo_sdk.SyncMove, arg0: int) -> bool
00106     *
00107     * @par Lua
00108     * syncMoveSegment(id: number) -> boolean
00109     * @endcoe
00110     * \endchinese
00111     * \english
00112     * Set the ID for the synchronized path segment.
00113     * In synchronized movements mode, all or none of the simultaneous move instructions must be programmed with corner
00114     * zones.
00115     * This means that move instructions with the same ID must either all have corner zones, or all have stop points.
00116     * If a move instruction with a corner zone and a move instruction with a stop point are synchronously executed in their
00117     * respective task program, an error will occur.
00118     * Synchronously executed move instructions can have corner zones of different sizes (e.g. one uses z10 and one uses z50).
00119     *
00120     * @param id
00121     * @return
00122     *
00123     * @par Python prototype
00124     * syncMoveSegment(self: pyaubo_sdk.SyncMove, arg0: int) -> bool
00125     *
00126     * @par Lua prototype
00127     * syncMoveSegment(id: number) -> boolean
00128     * @endcoe
00129     * \endenglish
00130     */
00131     bool syncMoveSegment(int id);
00132
00133     /**
00134     * @ingroup SyncMove
00135     * \chinese
00136     * syncMoveOff
00137     * syncMoveOff    syncMoveOff
00138     * syncMoveOff
00139     *
00140     * @param syncident
00141     * @return
00142     *
00143     * @par Python
00144     * syncMoveOff(self: pyaubo_sdk.SyncMove, arg0: str) -> int
00145     *
00146     * @par Lua
00147     * syncMoveOff(syncident: string) -> nil
00148     * @endcoe
00149     * \endchinese
00150     * \english
00151     * syncMoveOff is used to end synchronized movement mode.
00152     *
00153     * A syncMoveOff instruction will wait for the other task programs. When
00154     * all task programs have reached the syncMoveOff, they will continue
00155     * their execution in unsynchronized mode.
00156     * A stop point must be programmed before the syncMoveOff instruction.
00157     *
00158     * @param syncident
00159     * @return
00160     *
00161     * @par Python prototype
00162     * syncMoveOff(self: pyaubo_sdk.SyncMove, arg0: str) -> int
00163     *
00164     * @par Lua prototype
00165     * syncMoveOff(syncident: string) -> nil
00166     * @endcoe
00167     * \endenglish
00168     */

```

```

00168 int syncMoveOff(const std::string &syncident);
00169
00170 /**
00171  * @ingroup SyncMove
00172  * \chinese
00173  * syncMoveUndo          syncMoveUndo
00174  *
00175  * syncMoveUndo    UNDO          syncMoveUndo
00176  *
00177  * @return
00178  *
00179  * @par Python
00180  * syncMoveUndo(self: pyaubo_sdk.SyncMove) -> int
00181  *
00182  * @par Lua
00183  * syncMoveUndo() -> nil
00184  * @endcoe
00185  * \endchinese
00186  * \english
00187  * syncMoveUndo is used to turn off synchronized movements, even if not
00188  * all the other task programs execute the syncMoveUndo instruction.
00189  *
00190  * syncMoveUndo is intended for UNDO handlers. When the program
00191  * pointer is moved from the procedure, syncMoveUndo is used to turn off
00192  * the synchronization.
00193  *
00194  * @return
00195  *
00196  * @par Python prototype
00197  * syncMoveUndo(self: pyaubo_sdk.SyncMove) -> int
00198  *
00199  * @par Lua prototype
00200  * syncMoveUndo() -> nil
00201  * @endcoe
00202  * \endenglish
00203  */
00204 int syncMoveUndo();
00205
00206 /**
00207  * @ingroup SyncMove
00208  * \chinese
00209  * waitSyncTasks
00210  *
00211  * waitSyncTasks          waitSyncTasks
00212  *
00213  *
00214  * @param syncident
00215  * @param taskset
00216  * @return
00217  *
00218  * @par Python
00219  * waitSyncTasks(self: pyaubo_sdk.SyncMove, arg0: str, arg1: Set[str]) -> int
00220  *
00221  * @par Lua
00222  * waitSyncTasks(syncident: string, taskset: table) -> nil
00223  * @endcoe
00224  * \endchinese
00225  * \english
00226  * waitSyncTasks is used to synchronize several task programs at a specific
00227  * point in the program.
00228  *
00229  * A waitSyncTasks instruction will wait for the other task programs. When all
00230  * task programs have reached the waitSyncTasks instruction, they will continue
00231  * their execution.
00232  *
00233  * @param syncident
00234  * @param taskset
00235  * @return
00236  *
00237  * @par Python prototype
00238  * waitSyncTasks(self: pyaubo_sdk.SyncMove, arg0: str, arg1: Set[str]) -> int
00239  *
00240  * @par Lua prototype
00241  * waitSyncTasks(syncident: string, taskset: table) -> nil
00242  * @endcoe
00243  * \endenglish
00244  */
00245 int waitSyncTasks(const std::string &syncident, const TaskSet &taskset);
00246
00247 /**
00248  * @ingroup SyncMove
00249  * \chinese
00250  * isSyncMoveOn
00251  *
00252  *          “      ”
00253  *
00254  * @return

```



```

00255  *
00256  * @par Python
00257  * isSyncMoveOn(self: pyaubo_sdk.SyncMove) -> bool
00258  *
00259  * @par Lua
00260  * isSyncMoveOn() -> boolean
00261  * \endchinese
00262  * \english
00263  * isSyncMoveOn is used to tell if the mechanical unit group is in synchronized movement mode.
00264  *
00265  * A task that does not control any mechanical unit can find out if the mechanical units defined in the parameter Use
Mechanical Unit Group are in synchronized movement mode.
00266  *
00267  * @return
00268  *
00269  * @par Python prototype
00270  * isSyncMoveOn(self: pyaubo_sdk.SyncMove) -> bool
00271  *
00272  * @par Lua prototype
00273  * isSyncMoveOn() -> boolean
00274  * \endenglish
00275  */
00276 bool isSyncMoveOn();
00277
00278 /**
00279  * @ingroup SyncMove
00280  * \chinese
00281  *
00282  *
00283  * @return
00284  *
00285  * @par Python
00286  * syncMoveSuspend(self: pyaubo_sdk.SyncMove) -> int
00287  *
00288  * @par Lua
00289  * syncMoveSuspend() -> nil
00290  * \endchinese
00291  * \english
00292  * Suspend synchronized movement mode.
00293  *
00294  * @return
00295  *
00296  * @par Python prototype
00297  * syncMoveSuspend(self: pyaubo_sdk.SyncMove) -> int
00298  *
00299  * @par Lua prototype
00300  * syncMoveSuspend() -> nil
00301  * \endenglish
00302  */
00303 int syncMoveSuspend();
00304
00305 /**
00306  * @ingroup SyncMove
00307  * \chinese
00308  *
00309  *
00310  * @return
00311  *
00312  * @par Python
00313  * syncMoveResume(self: pyaubo_sdk.SyncMove) -> int
00314  *
00315  * @par Lua
00316  * syncMoveResume() -> nil
00317  * \endchinese
00318  * \english
00319  * Resume synchronized movement mode.
00320  *
00321  * @return
00322  *
00323  * @par Python prototype
00324  * syncMoveResume(self: pyaubo_sdk.SyncMove) -> int
00325  *
00326  * @par Lua prototype
00327  * syncMoveResume() -> nil
00328  * \endenglish
00329  */
00330 int syncMoveResume();
00331
00332 /**
00333  * @ingroup SyncMove
00334  * \chinese
00335  *      name      pose      ref_frame
00336  *      ref_frame
00337  *      ref_frame      frameAttach()
00338  *
00339  * @param name:
00340  * @param pose:

```

```

00341 * @param ref_frame:          “base”
00342 *
00343 * @return
00344 * \endchinese
00345 * \english
00346 * Add a frame with the name “name” initialized at the specified pose
00347 * expressed in the ref_frame coordinate frame. This command only adds a
00348 * frame to the world, it does not attach it to the ref_frame coordinate
00349 * frame. Use frameAttach() to attach the newly added frame to ref_frame if
00350 * desired.
00351 *
00352 * @param name: name of the frame to be added. The name must not be the same
00353 * as any existing world model object (frame, axis, or axis group),
00354 * otherwise an exception is thrown
00355 * @param pose: initial pose of the new object
00356 * @param ref_frame: name of the world model object whose coordinate frame
00357 * the pose is expressed in. If nothing is provided here, the default is the
00358 * robot “base” frame.
00359 *
00360 * @return
00361 * \endenglish
00362 */
00363 int frameAdd(const std::string &name, const std::vector<double> &pose,
00364             const std::string &ref_name);
00365
00366 /**
00367 * @ingroup SyncMove
00368 * \chinese
00369 *
00370 *
00371 *      “world” “flange” “tcp”
00372 *
00373 *
00374 *
00375 *      MotionPlus parent
00376 *
00377 * @param child:          “world” “flange” “tcp”
00378 * @param parent:
00379 *
00380 * @return
00381 * \endchinese
00382 * \english
00383 * Attaches the child frame to the parent world model object. The relative
00384 * transform between the parent and child will be set such that the child
00385 * does not move in the world when the attachment occurs.
00386 *
00387 * The child cannot be “world”, “flange”, “tcp”, or the same as parent.
00388 *
00389 * This will fail if child or parent is not an existing frame, or this makes
00390 * the attachments form a closed chain.
00391 *
00392 * If being used with the MotionPlus, the parent argument can be the name of
00393 * an external axis or axis group.
00394 *
00395 * @param child: name of the frame to be attached. The name must not be
00396 * “world”, “flange”, or “tcp”.
00397 * @param parent: name of the object that the child frame will be attached
00398 * to.
00399 *
00400 * @return
00401 * \endenglish
00402 */
00403 int frameAttach(const std::string &child, const std::string &parent);
00404
00405 /**
00406 * @ingroup SyncMove
00407 * \chinese
00408 *
00409 *
00410 *      “world” “base” “flange” “tcp”
00411 *
00412 *      “world”
00413 *
00414 * @return
00415 * \endchinese
00416 * \english
00417 * Delete all frames that have been added to the world model.
00418 *
00419 * The “world”, “base”, “flange”, and “tcp” frames cannot be deleted.
00420 *
00421 * Any frames that are attached to the deleted frames will be attached to
00422 * the “world” frame with new frame offsets set such that the detached
00423 * frames do not move in the world.
00424 *
00425 * @return
00426 * \endenglish
00427 */

```

```

00428 int frameDeleteAll();
00429
00430 /**
00431  * @ingroup SyncMove
00432  * \chinese
00433  *
00434  *
00435  * “world” “base” “flange” “tcp”
00436  *
00437  * “world”
00438  *
00439  *
00440  *
00441  * @param name:
00442  *
00443  * @return
00444  * \endchinese
00445  * \english
00446  * Delete the frame with name from the world model.
00447  *
00448  * The “world”, “base”, “flange”, and “tcp” frames cannot be deleted.
00449  *
00450  * Any frames that are attached to the deleted frame will be attached to the
00451  * “world” frame with new frame offsets set such that the detached frame
00452  * does not move in the world.
00453  *
00454  * This command will fail if the frame does not exist.
00455  *
00456  * @param name: name of the frame to be deleted
00457  *
00458  * @return
00459  * \endenglish
00460  */
00461 int frameDelete(const std::string &name);
00462
00463 /**
00464  * @ingroup SyncMove
00465  * \chinese
00466  * name pose pose ref_name
00467  *
00468  * name “world” “flange” “tcp” “tcp” set_tcp()
00469  *
00470  * MotionPlus ref_name
00471  *
00472  * @param name:
00473  * @param pose:
00474  * @param ref_name: pose “base”
00475  *
00476  * @return
00477  * \endchinese
00478  * \english
00479  * Changes the placement of the coordinate frame named name to the new
00480  * placement given by pose that is defined in the ref_name coordinate frame.
00481  *
00482  * This will fail if name is “world”, “flange”, “tcp”, or if the frame does
00483  * not exist. Note: to move the “tcp” frame, use the set_tcp() command
00484  * instead.
00485  *
00486  * If being used with the MotionPlus, the ref_name argument can be the name
00487  * of an external axis or axis group.
00488  *
00489  * @param name: the name of the frame to move
00490  * @param pose: the new placement
00491  * @param ref_name: the coordinate frame that pose is expressed in. The
00492  * default value is the robot’s “base” frame.
00493  *
00494  * @return
00495  * \endenglish
00496  */
00497 int frameMove(const std::string &name, const std::vector<double> &pose,
00498              const std::string &ref_name);
00499
00500 /**
00501  * @ingroup SyncMove
00502  * \chinese
00503  * name rel_frame ref_frame
00504  * ref_frame name rel_frame rel_frame
00505  *
00506  *
00507  *
00508  * MotionPlus
00509  *
00510  * @param name:
00511  * @param rel_frame: “ ”
00512  * @param ref_frame: “ ” rel_frame
00513  *
00514  * @return ref_frame name

```

```

00515 * \endchinese
00516 * \english
00517 * Get the pose of the name frame relative to the rel_frame frame but
00518 * expressed in the coordinates of the ref_frame frame. If ref_frame is not
00519 * provided, then this returns the pose of the name frame relative to and
00520 * expressed in the same frame as rel_frame.
00521 *
00522 * This will fail if any arguments are not an existing frame.
00523 *
00524 * If being used with MotionPlus, all three arguments can also be the names
00525 * of external axes or axis groups.
00526 *
00527 * @param name: name of the frame to query.
00528 * @param rel_frame: short for "relative frame" is the frame where the pose
00529 * is computed relative to
00530 * @param ref_frame: short for "reference frame" is the frame to express the
00531 * coordinates of resulting relative pose in. If this is not provided, then
00532 * it will default to match the value of rel_frame.
00533 *
00534 * @return The pose of the frame expressed in the ref_frame coordinates.
00535 * \endenglish
00536 */
00537 std::vector<double> frameGetPose(const std::string &name,
00538                                const std::string &rel_frame,
00539                                const std::string &ref_frame);
00540
00541 /**
00542 * @ingroup SyncMove
00543 * \chinese
00544 *   from_frame   to_frame
00545 *
00546 *
00547 *
00548 *   MotionPlus
00549 *
00550 * @param pose:
00551 * @param from_frame:
00552 * @param to_frame:
00553 *
00554 * @return to_frame   pose
00555 * \endchinese
00556 * \english
00557 * Convert pose from from_frame to to_frame.
00558 *
00559 * This will fail if either coordinate system argument is not an existing
00560 * frame.
00561 *
00562 * If being used with MotionPlus, all three arguments can also be the names
00563 * of external axes or axis groups.
00564 *
00565 * @param pose: pose to be converted
00566 * @param from_frame: name of reference frame at origin of old coordinate
00567 * system
00568 * @param to_frame: name of reference frame at origin of new coordinate
00569 * system
00570 *
00571 * @return Value of pose expressed in the coordinates of to_frame.
00572 * \endenglish
00573 */
00574 std::vector<double> frameConvertPose(const std::vector<double> &pose,
00575                                     const std::string &from_frame,
00576                                     const std::string &to_frame);
00577
00578 /**
00579 * @ingroup SyncMove
00580 * \chinese
00581 *
00582 *
00583 * @param name:
00584 *
00585 * @return      true   false
00586 * \endchinese
00587 * \english
00588 * Queries for the existence of a frame by the given name.
00589 *
00590 * @param name: name of the frame to be queried.
00591 *
00592 * @return Returns true if there is a frame by the given name, false if not.
00593 * \endenglish
00594 */
00595 bool frameExist(const std::string &name);
00596
00597 /**
00598 * @ingroup SyncMove
00599 * \chinese
00600 *   name
00601 *

```

```

00602     *           "world"
00603     *
00604     * @param name:
00605     *
00606     * @return
00607     * \endchinese
00608     * \english
00609     * Get the parent of the frame named name in the world model.
00610     *
00611     * If the frame is not attached to another frame, then "world" is the
00612     * parent.
00613     *
00614     * @param name: the frame being queried
00615     *
00616     * @return name of the parent as a string
00617     * \endenglish
00618     */
00619     std::string frameGetParent(const std::string &name);
00620
00621     /**
00622     * @ingroup SyncMove
00623     * \chinese
00624     *     MotionPlus
00625     *
00626     * @param name:
00627     *
00628     * @return
00629     * \endchinese
00630     * \english
00631     * Returns a list of immediate child object frame names. Parent-child
00632     * relationships are defined by world model attachments. If being used with
00633     * MotionPlus, the child objects may also be an axis group or an axis.
00634     *
00635     * @param name: the name of the parent object.
00636     *
00637     * @return a list of immediate child object frame names
00638     * \endenglish
00639     */
00640     std::vector<std::string> frameGetChildren(const std::string &name);
00641
00642     /**
00643     * @ingroup SyncMove
00644     * \chinese
00645     *     name      ref_frame      pose
00646     *
00647     *
00648     *
00649     *
00650     *
00651     *
00652     *     6
00653     *
00654     * @param name:
00655     * @param pose:
00656     * @param ref_frame ( ): pose           "base"
00657     *
00658     * @return
00659     * \endchinese
00660     * \english
00661     * Adds a new axis group with the given name to the world model. It is
00662     * placed at the given pose in the reference coordinate frame defined by
00663     * ref_frame.
00664     *
00665     * An axis group can only be attached to the world coordinate frame.
00666     *
00667     * Each axis group has a coordinate frame attached to its base, which can be
00668     * used as an argument to other world model functions by referring the name
00669     * of the group.
00670     *
00671     * At most 6 axis groups can be added to the world model.
00672     *
00673     * @param name: (string) Name of the axis group to add. The name cannot be
00674     * an empty string. Names used by world model objects (e.g., frame, axis
00675     * group, axis, etc.) must be unique.
00676     *
00677     * @param pose: (pose) Pose of the axis group's base, in the reference
00678     * coordinate frame.
00679     *
00680     * @param ref_frame (optional): (string) Name of the reference coordinate
00681     * frame that pose is defined in. This can be any world model entity with a
00682     * coordinate system (e.g., frame, axis group, axis, etc.). The default
00683     * value "base" refers to the robot's base frame.
00684     *
00685     * @return
00686     * \endenglish
00687     */
00688     int axisGroupAdd(const std::string &name, const std::vector<double> &pose,

```

```

00689         const std::string &ref_frame);
00690
00691     /**
00692     * @ingroup SyncMove
00693     * \chinese
00694     *
00695     *
00696     *
00697     *
00698     *
00699     *
00700     * @param name:
00701     *
00702     * @return
00703     * \endchinese
00704     * \english
00705     * Deletes the axis group with the given name from the world model.
00706     *
00707     * All attached axes are also disabled (if live) and deleted.
00708     *
00709     * This function will fail, if this axis group is under control by another function.
00710     *
00711     * @param name: (string) Name of the axis group to delete. Axis group with
00712     * such name must exist.
00713     *
00714     * @return
00715     * \endenglish
00716     */
00717     int axisGroupDelete(const std::string &name);
00718
00719     /**
00720     * @ingroup SyncMove
00721     * \chinese
00722     *   group_name      name      parent   pose   pose      0
00723     *                   type v_limit a_limit q_limits  axis_index
00724     *   pose
00725     *
00726     *
00727     * @param group_name:      axis_group_add()
00728     * @param name:
00729     * @param parent:      group_name
00730     * @param pose:      type 0  z      type 1  z
00731     * @param type: 0      1
00732     * @param v_limit:
00733     * @param a_limit:
00734     * @param q_limits:
00735     * @param axis_index:
00736     * \endchinese
00737     * \english
00738     * Adds an external axis with the given name to the axis group named group_name.
00739     * The axis is attached at the given pose in the reference coordinate frame defined by parent when its axis position is 0.
00740     * The type, max velocity, max acceleration, position limits, and index of this axis are defined by type, v_limit, a_limit,
00741     * q_limits, and axis_index, respectively.
00742     * The pose parameter is typically obtained from a calibration process when the external axis is commissioned.
00743     * This function will fail if this axis group is under control of another function, or if the kinematic chain created by the
00744     * attachment forms a closed chain.
00745     *
00746     * @param group_name: Name of the axis group this new axis is added to. The axis group would have been created
00747     * using axis_group_add(). Axis group with such name must exist.
00748     * @param name: Name of the new axis. The name cannot be an empty string. Names used by world model objects (e.g.,
00749     * frame, axis group, axis, etc.) must be unique.
00750     * @param parent: Name of the parent axis. If it's empty or the same as group_name, the new axis will be attached to
00751     * the base of the axis group. Axis with such name must exist in the axis group.
00752     * @param pose: The zero-position pose, in the parent coordinate frame, this axis will be placed and attached to. This is
00753     * the pose the axis will be (relative to its parent) when its axis position is 0. If type is 0 (rotary), then the z axis of the
00754     * frame corresponds to the axis of rotation. If type is 1 (linear), then the z axis is the axis of translation.
00755     * @param type: Axis type, 0 for rotary, 1 for linear.
00756     * @param v_limit: Maximum velocity.
00757     * @param a_limit: Maximum acceleration.
00758     * @param q_limits: Position limits.
00759     * @param axis_index: Axis index.
00760     * \endenglish
00761     */
00762     int axisGroupAddAxis(const std::string &group_name, const std::string &name,
00763         const std::string &parent,
00764         const std::vector<double> &pose);
00765
00766     /**
00767     * @ingroup SyncMove
00768     * \chinese
00769     *   pose
00770     *
00771     *
00772     *
00773     *
00774     *
00775     * @param name:
00776     * @param pose ( ):      0
00777     * \endchinese

```

```

00769 * \english
00770 * Updates the corresponding properties of axis with name. The pose
00771 * parameter is typically obtained from a calibration process when the
00772 * external axis is commissioned. See here for a guide on a basic routine
00773 * for calibrating a single rotary axis.
00774 *
00775 * This function will fail, if the axis group the axis attached to is
00776 * already being controlled by another command.
00777 * This function will fail, if any attached axis of the axis group is live
00778 * and enabled.
00779 *
00780 * @param name: (string) Name of the axis to update. Axis with such name
00781 * must exist.
00782 *
00783 * @param pose (optional): (pose) New zero-position pose, in the coordinate
00784 * frame of the parent axis (or axis group), of the axis. This is the pose
00785 * of the axis when its axis position is 0.
00786 * \endenglish
00787 */
00788 int axisGroupUpdateAxis(const std::string &name,
00789                        const std::vector<double> &pose);
00790
00791 /**
00792 * @ingroup SyncMove
00793 * \chinese
00794 *     RTDE
00795 *
00796 * @param axis_name: (string)
00797 *
00798 * @return integer:    RTDE
00799 * \endchinese
00800 * \english
00801 * Returns the index of the axis with given axis_name in the RTDE target
00802 * positions and actual positions arrays.
00803 *
00804 * @param axis_name: (string) Name of the axis in query.
00805 * Axis with such name must exist.
00806 *
00807 * @return integer: Index of the axis in the RTDE target positions and
00808 * actual positions arrays.
00809 * \endenglish
00810 */
00811 int axisGroupGetAxisIndex(const std::string &name);
00812
00813 /**
00814 * @ingroup SyncMove
00815 * \chinese
00816 *
00817 *
00818 * @param axis_index: ( )
00819 *
00820 * @return :
00821 * \endchinese
00822 * \english
00823 * Returns the name of the axis with the given axis_index.
00824 *
00825 * @param axis_index: (integer) Index of the axis in query.
00826 * Axis with such index must exist.
00827 *
00828 * @return string: Name of the axis.
00829 * \endenglish
00830 */
00831 std::string axisGroupGetAxisName(int index);
00832
00833 /**
00834 * @ingroup SyncMove
00835 * \chinese
00836 *
00837 *     group_name
00838 *
00839 *
00840 *
00841 * @param group_name ( ): (string)
00842 *
00843 * @return Double[]:
00844 * \endchinese
00845 * \english
00846 * Returns the current target positions of the axis group with group_name.
00847 * If group_name is not provided, the target positions of all external axes
00848 * will be returned.
00849 *
00850 * This function will fail, if the external axis bus is disabled.
00851 *
00852 * @param group_name (optional): (string) Name of the axis group in query.
00853 * Axis group with such name must REALLY exist.
00854 *
00855 * @return Double[]: Target positions of the involved axes, in the order of

```

```

00856     * their external axis indices.
00857     * \endenglish
00858     */
00859     std::vector<double> axisGroupGetTargetPositions(
00860         const std::string &group_name);
00861
00862     /**
00863     * @ingroup SyncMove
00864     * \chinese
00865     *
00866     *     group_name
00867     *
00868     *
00869     *
00870     * @param group_name ( ): (string)
00871     *
00872     * @return Double[]:
00873     * \endchinese
00874     * \english
00875     * Returns the current actual positions of the axis group with group_name.
00876     * If group_name is not provided, the actual positions of all external axes
00877     * will be returned.
00878     *
00879     * This function will fail, if the external axis bus is disabled.
00880     *
00881     * @param group_name (optional): (string) Name of the axis group in query.
00882     * Axis group with such name must exist.
00883     *
00884     * @return Double[]: Actual positions of the involved axes, in the order of
00885     * their external axis indices.
00886     * \endenglish
00887     */
00888     std::vector<double> axisGroupGetActualPositions(
00889         const std::string &group_name);
00890
00891     /**
00892     * @ingroup SyncMove
00893     * \chinese
00894     *     offset    group_name
00895     *
00896     *
00897     *
00898     * @param group_name: (string)
00899     *
00900     * @param offset: (float[])    offset
00901     *
00902     * @return
00903     * \endchinese
00904     * \english
00905     * Shifts the target and actual positions of the axis group group_name by
00906     * the given offset.
00907     *
00908     * This is a software shift that happens in the controller only, it does not
00909     * affect external axis drives. The shift is also applied to any streamed
00910     * target and actual positions published on RTDE.
00911     *
00912     * @param group_name: (string) Name of the axis group to apply the offset
00913     * positions to. Axis group with such name must exist.
00914     *
00915     * @param offset: (float[]) Offsets that the target and actual positions
00916     * should be shifted by. The size of offset must match the number of axes
00917     * attached to the given group.
00918     *
00919     * @return
00920     * \endenglish
00921     */
00922     int axisGroupOffsetPositions(const std::string &group_name,
00923         const std::vector<double> &offset);
00924
00925     /**
00926     * @ingroup SyncMove
00927     * \chinese
00928     *     group_name    q
00929     *     a
00930     *     v
00931     *
00932     *
00933     *
00934     * @param group_name: (string)
00935     * @param q: (float[])    q
00936     * @param a: (float)    (0,1]
00937     * @param v: (float)    (0,1]
00938     *
00939     * :
00940     * \endchinese
00941     * \english
00942     * Moves the axes of axis group named group_name to new positions q, using a

```



```

00943 * trapezoidal velocity profile. Factor a specifying the percentage of the
00944 * max profile accelerations out of the acceleration limits of each axes.
00945 * Factor v specifying the percentage of the max profile velocities out of
00946 * the velocity limits of each axes.
00947 *
00948 * The actual accelerations and velocities are determined by the most
00949 * constraining axis, so that all the axes complete the acceleration,
00950 * cruise, and deceleration phases at the same time.
00951 *
00952 * @param group_name: (string) Name of the axis group to move.
00953 * Axis group with such name must exist.
00954 *
00955 * @param q: (float[]) Target positions in rad (rotary) or in m (linear). If
00956 * the target exceeds the position limits, then it is set to the nearest
00957 * limit. The involved axes are ordered increasingly by their axis indices.
00958 * The size of q must match the number of axes attached to the given group.
00959 *
00960 * @param a: (float) Factor specifying the max accelerations of this move
00961 * out of the acceleration limits. a must be in range of (0,1].
00962 *
00963 * @param v: (float) Factor specifying the max velocities of this move out
00964 * of the velocity limits. v must be in range of (0,1].
00965 *
00966 * Return: n/a
00967 * \endenglish
00968 */
00969 int axisGroupMoveJoint(const std::string &group_name,
00970                       const std::vector<double> &q, double a, double v);
00971
00972 /**
00973 * @ingroup SyncMove
00974 * \chinese
00975 *      a      group_name      qd      t
00976 * @param group_name: (string)
00977 * @param qd: (float[]) qd
00978 * @param a: (float) (0,1]
00979 * @param t ( ): (float) t < 0      t 0
00980 * \endchinese
00981 * \english
00982 * Accelerates the axes of axis group named group_name up to the target
00983 * velocities qd. Factor a specifying the percentage of the max
00984 * accelerations out of the acceleration limits of each axes. The function
00985 * will run for a period of t seconds.
00986 *
00987 * @param group_name: (string) Name of the external axis group to control.
00988 * Axis group with such name must exist.
00989 *
00990 * @param qd: (float[]) Target velocities for the axes in the axis group. If
00991 * the target exceeds the velocity limits, then it is set to the limit. The
00992 * involved axes are ordered increasingly by their axis indices. The size of
00993 * qd must match the number of axes attached to the given group.
00994 *
00995 * @param a: (float) Factor specifying the max accelerations of this move
00996 * out of the acceleration limits. a must be in range of (0,1].
00997 *
00998 * @param t (optional): (float) Duration in seconds before the function
00999 * returns. If t < 0, then the function will return when the target
01000 * velocities are reached. if t 0, then the function will return after
01001 * this duration, regardless of what the achieved axes velocities are.
01002 * \endenglish
01003 */
01004 int axisGroupSpeedJoint(const std::string &group_name,
01005                        const std::vector<double> &qd, double a, double t);
01006
01007 protected:
01008     void *d_;
01009 };
01010
01011 using SyncMovePtr = std::shared_ptr<SyncMove>;
01012 } // namespace common_interface
01013 } // namespace arcs
01014 #endif // AUBO_SDK_SYNC_MOVE_INTERFACE_H

```

12.44 include/aubo/system_info.h File Reference

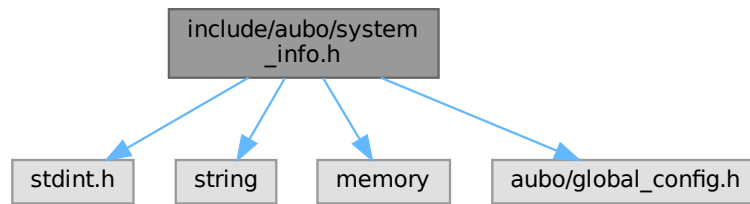
```

#include <stdint.h>
#include <string>
#include <memory>

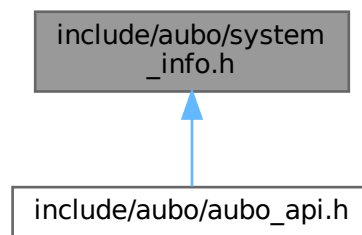
```

```
#include <aubo/global_config.h>
```

Include dependency graph for `system_info.h`:



This graph shows which files directly or indirectly include this file:



Classes

- class [arcs::common_interface::SystemInfo](#)

Namespaces

- namespace [arcs](#)
- namespace [arcs::common_interface](#)

Typedefs

- using [arcs::common_interface::SystemInfoPtr](#) = `std::shared_ptr<SystemInfo>`

12.44.1 Detailed Description

Definition in file [system_info.h](#).

12.45 system_info.h

[Go to the documentation of this file.](#)

```

00001 /** @file system_info.h
00002  * @brief
00003  */
00004 #ifndef AUBO_SDK_SYSTEM_INFO_INTERFACE_H
00005 #define AUBO_SDK_SYSTEM_INFO_INTERFACE_H
00006
00007 #include <stdint.h>
00008 #include <string>
00009 #include <memory>
00010
00011 #include <aubo/global_config.h>
00012
00013 namespace arcs {
00014 namespace common_interface {
00015
00016 /**
00017  * @defgroup SystemInfo SystemInfo ( )
00018  * \~chinese
00019  * \~english SystemInfo
00020  */
00021 class ARCS_ABI_EXPORT SystemInfo
00022 {
00023 public:
00024     SystemInfo();
00025     virtual ~SystemInfo();
00026
00027     /**
00028      * @ingroup SystemInfo
00029      * \chinese
00030      *
00031      *
00032      * @return
00033      *
00034      * @par Python
00035      * getControlSoftwareVersionCode(self: pyaubo_sdk.SystemInfo) -> int
00036      *
00037      * @par Lua
00038      * getControlSoftwareVersionCode() -> number
00039      *
00040      * @par C++
00041      * @code
00042      * int control_version =
00043      * rpc_cli->getSystemInfo()->getControlSoftwareVersionCode();
00044      * @endcode
00045      *
00046      * @par JSON-RPC
00047      * {"jsonrpc":"2.0","method":"SystemInfo.getControlSoftwareVersionCode","params":[],"id":1}
00048      *
00049      * @par JSON-RPC
00050      * {"id":1,"jsonrpc":"2.0","result":28003}
00051      *
00052      * \endchinese
00053      * \english
00054      * Get the controller software version code
00055      *
00056      * @return Returns the controller software version code
00057      *
00058      * @par Python prototype
00059      * getControlSoftwareVersionCode(self: pyaubo_sdk.SystemInfo) -> int
00060      *
00061      * @par Lua prototype
00062      * getControlSoftwareVersionCode() -> number
00063      *
00064      * @par C++ example
00065      * @code
00066      * int control_version =
00067      * rpc_cli->getSystemInfo()->getControlSoftwareVersionCode();
00068      * @endcode
00069      *
00070      * @par JSON-RPC request example
00071      * {"jsonrpc":"2.0","method":"SystemInfo.getControlSoftwareVersionCode","params":[],"id":1}
00072      *
00073      * @par JSON-RPC response example
00074      * {"id":1,"jsonrpc":"2.0","result":28003}
00075      *
00076      * \endenglish
00077      */
00078     int getControlSoftwareVersionCode();
00079
00080     /**
00081      * @ingroup SystemInfo
00082      * \chinese

```

```

00083 *
00084 *
00085 * @return
00086 *
00087 * @par Python
00088 * getControlSoftwareFullVersion(self: pyaubo_sdk.SystemInfo) -> str
00089 *
00090 * @par Lua
00091 * getControlSoftwareFullVersion() -> string
00092 *
00093 * @par C++
00094 * @code
00095 * std::string control_version =
00096 * rpc_cli->getSystemInfo()->getControlSoftwareFullVersion();
00097 * @endcode
00098 *
00099 * @par JSON-RPC
00100 * {"jsonrpc":"2.0","method":"SystemInfo.getControlSoftwareFullVersion","params":[],"id":1}
00101 *
00102 * @par JSON-RPC
00103 * {"id":1,"jsonrpc":"2.0","result":"0.31.0-alpha.16+20alc76"}
00104 *
00105 * \endchinese
00106 * \english
00107 * Get the full controller software version
00108 *
00109 * @return Returns the full controller software version
00110 *
00111 * @par Python prototype
00112 * getControlSoftwareFullVersion(self: pyaubo_sdk.SystemInfo) -> str
00113 *
00114 * @par Lua prototype
00115 * getControlSoftwareFullVersion() -> string
00116 *
00117 * @par C++ example
00118 * @code
00119 * std::string control_version =
00120 * rpc_cli->getSystemInfo()->getControlSoftwareFullVersion();
00121 * @endcode
00122 *
00123 * @par JSON-RPC request example
00124 * {"jsonrpc":"2.0","method":"SystemInfo.getControlSoftwareFullVersion","params":[],"id":1}
00125 *
00126 * @par JSON-RPC response example
00127 * {"id":1,"jsonrpc":"2.0","result":"0.31.0-alpha.16+20alc76"}
00128 *
00129 * \endenglish
00130 */
00131 std::string getControlSoftwareFullVersion();
00132
00133 /**
00134 * @ingroup SystemInfo
00135 * \chinese
00136 *
00137 *
00138 * @return
00139 *
00140 * @par Python
00141 * getInterfaceVersionCode(self: pyaubo_sdk.SystemInfo) -> int
00142 *
00143 * @par Lua
00144 * getInterfaceVersionCode() -> number
00145 *
00146 * @par C++
00147 * @code
00148 * int interface_version =
00149 * rpc_cli->getSystemInfo()->getInterfaceVersionCode();
00150 * @endcode
00151 *
00152 * @par JSON-RPC
00153 * {"jsonrpc":"2.0","method":"SystemInfo.getInterfaceVersionCode","params":[],"id":1}
00154 *
00155 * @par JSON-RPC
00156 * {"id":1,"jsonrpc":"2.0","result":22003}
00157 *
00158 * \endchinese
00159 * \english
00160 * Get the interface version code
00161 *
00162 * @return Returns the interface version code
00163 *
00164 * @par Python prototype
00165 * getInterfaceVersionCode(self: pyaubo_sdk.SystemInfo) -> int
00166 *
00167 * @par Lua prototype
00168 * getInterfaceVersionCode() -> number
00169 *

```

```

00170 * @par C++ example
00171 * @code
00172 * int interface_version =
00173 * rpc_cli->getSystemInfo()->getInterfaceVersionCode();
00174 * @endcode
00175 *
00176 * @par JSON-RPC request example
00177 * {"jsonrpc":"2.0","method":"SystemInfo.getInterfaceVersionCode","params":[],"id":1}
00178 *
00179 * @par JSON-RPC response example
00180 * {"id":1,"jsonrpc":"2.0","result":22003}
00181 *
00182 * \endenglish
00183 */
00184 int getInterfaceVersionCode();
00185
00186 /**
00187 * @ingroup SystemInfo
00188 * \chinese
00189 *
00190 *
00191 * @return
00192 *
00193 * @par Python
00194 * getControlSoftwareBuildDate(self: pyaubo_sdk.SystemInfo) -> str
00195 *
00196 * @par Lua
00197 * getControlSoftwareBuildDate() -> string
00198 *
00199 * @par C++
00200 * @code
00201 * std::string build_date =
00202 * rpc_cli->getSystemInfo()->getControlSoftwareBuildDate();
00203 * @endcode
00204 *
00205 * @par JSON-RPC
00206 * {"jsonrpc":"2.0","method":"SystemInfo.getControlSoftwareBuildDate","params":[],"id":1}
00207 *
00208 * @par JSON-RPC
00209 * {"id":1,"jsonrpc":"2.0","result":"2024-3-5 07:03:20"}
00210 *
00211 * \endchinese
00212 * \english
00213 * Get the controller software build date
00214 *
00215 * @return Returns the controller software build date
00216 *
00217 * @par Python prototype
00218 * getControlSoftwareBuildDate(self: pyaubo_sdk.SystemInfo) -> str
00219 *
00220 * @par Lua prototype
00221 * getControlSoftwareBuildDate() -> string
00222 *
00223 * @par C++ example
00224 * @code
00225 * std::string build_date =
00226 * rpc_cli->getSystemInfo()->getControlSoftwareBuildDate();
00227 * @endcode
00228 *
00229 * @par JSON-RPC request example
00230 * {"jsonrpc":"2.0","method":"SystemInfo.getControlSoftwareBuildDate","params":[],"id":1}
00231 *
00232 * @par JSON-RPC response example
00233 * {"id":1,"jsonrpc":"2.0","result":"2024-3-5 07:03:20"}
00234 *
00235 * \endenglish
00236 */
00237 std::string getControlSoftwareBuildDate();
00238
00239 /**
00240 * @ingroup SystemInfo
00241 * \chinese
00242 * git
00243 *
00244 * @return git
00245 *
00246 * @par Python
00247 * getControlSoftwareVersionHash(self: pyaubo_sdk.SystemInfo) -> str
00248 *
00249 * @par Lua
00250 * getControlSoftwareVersionHash() -> string
00251 *
00252 * @par C++
00253 * @code
00254 * std::string git_version =
00255 * rpc_cli->getSystemInfo()->getControlSoftwareVersionHash();
00256 * @endcode

```

```

00257 *
00258 * @par JSON-RPC
00259 * {"jsonrpc":"2.0","method":"SystemInfo.getControlSoftwareVersionHash","params":[],"id":1}
00260 *
00261 * @par JSON-RPC
00262 * {"id":1,"jsonrpc":"2.0","result":"fa4f64a"}
00263 *
00264 * \endchinese
00265 * \english
00266 * Get the controller software git version
00267 *
00268 * @return Returns the controller software git version
00269 *
00270 * @par Python prototype
00271 * getControlSoftwareVersionHash(self: pyaubo_sdk.SystemInfo) -> str
00272 *
00273 * @par Lua prototype
00274 * getControlSoftwareVersionHash() -> string
00275 *
00276 * @par C++ example
00277 * @code
00278 * std::string git_version =
00279 * rpc_cli->getSystemInfo()->getControlSoftwareVersionHash();
00280 * @endcode
00281 *
00282 * @par JSON-RPC request example
00283 * {"jsonrpc":"2.0","method":"SystemInfo.getControlSoftwareVersionHash","params":[],"id":1}
00284 *
00285 * @par JSON-RPC response example
00286 * {"id":1,"jsonrpc":"2.0","result":"fa4f64a"}
00287 *
00288 * \endenglish
00289 */
00290 std::string getControlSoftwareVersionHash();
00291
00292 /**
00293 * @ingroup SystemInfo
00294 * \chinese
00295 * ( ns )
00296 *
00297 * @return ( ns )
00298 *
00299 * @par Python
00300 * getControlSystemTime(self: pyaubo_sdk.SystemInfo) -> int
00301 *
00302 * @par Lua
00303 * getControlSystemTime() -> number
00304 *
00305 * @par C++
00306 * @code
00307 * std::string system_time =
00308 * rpc_cli->getSystemInfo()->getControlSystemTime();
00309 * @endcode
00310 *
00311 * @par JSON-RPC
00312 * {"jsonrpc":"2.0","method":"SystemInfo.getControlSystemTime","params":[],"id":1}
00313 *
00314 * @par JSON-RPC
00315 * {"id":1,"jsonrpc":"2.0","result":9287799079682}
00316 *
00317 * \endchinese
00318 * \english
00319 * Get the system time (software start time in nanoseconds)
00320 *
00321 * @return Returns the system time (software start time in nanoseconds)
00322 *
00323 * @par Python prototype
00324 * getControlSystemTime(self: pyaubo_sdk.SystemInfo) -> int
00325 *
00326 * @par Lua prototype
00327 * getControlSystemTime() -> number
00328 *
00329 * @par C++ example
00330 * @code
00331 * std::string system_time =
00332 * rpc_cli->getSystemInfo()->getControlSystemTime();
00333 * @endcode
00334 *
00335 * @par JSON-RPC request example
00336 * {"jsonrpc":"2.0","method":"SystemInfo.getControlSystemTime","params":[],"id":1}
00337 *
00338 * @par JSON-RPC response example
00339 * {"id":1,"jsonrpc":"2.0","result":9287799079682}
00340 *
00341 * \endenglish
00342 */
00343 uint64_t getControlSystemTime();

```

```

00344
00345 protected:
00346     void *d_;
00347 };
00348
00349 using SystemInfoPtr = std::shared_ptr<SystemInfo>;
00350
00351 } // namespace common_interface
00352 } // namespace arcs
00353 #endif

```

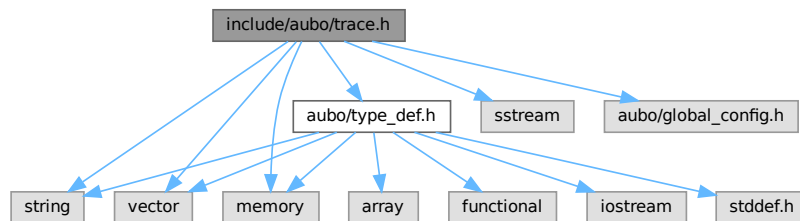
12.46 include/aubo/trace.h File Reference

Interface for injecting logs into the controller's logging system.

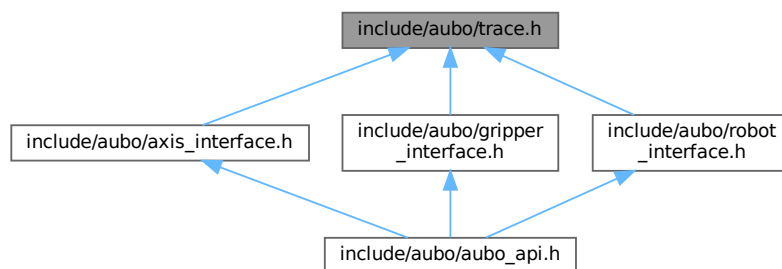
```

#include <string>
#include <vector>
#include <memory>
#include <sstream>
#include <aubo/global_config.h>
#include <aubo/type_def.h>
Include dependency graph for trace.h:

```



This graph shows which files directly or indirectly include this file:



Classes

- class `arcs::common_interface::Trace`

Namespaces

- namespace [arcs](#)
- namespace [arcs::common_interface](#)

Typedefs

- using [arcs::common_interface::TracePtr](#) = std::shared_ptr<[Trace](#)>

12.46.1 Detailed Description

Interface for injecting logs into the controller's logging system.

Definition in file [trace.h](#).

12.47 [trace.h](#)

[Go to the documentation of this file.](#)

```

00001 /** @file trace.h
00002 * \~chinese @brief          \~english @brief Interface for injecting logs into the controller's logging system
00003 */
00004 #ifndef AUBO_SDK_TRACE_INTERFACE_H
00005 #define AUBO_SDK_TRACE_INTERFACE_H
00006
00007 #include <string>
00008 #include <vector>
00009 #include <memory>
00010 #include <sstream>
00011
00012 #include <aubo/global_config.h>
00013 #include <aubo/type_def.h>
00014
00015 namespace arcs {
00016 namespace common_interface {
00017
00018 /**
00019 * @defgroup Trace Trace ( )
00020 * \~chinese          \~english provides a logging system for controller extension programs
00021 */
00022 class ARCS_ABI_EXPORT Trace
00023 {
00024 public:
00025     Trace();
00026     virtual ~Trace();
00027
00028 /**
00029 * @ingroup Trace
00030 * \~chinese  aubo_control    \~english Injects alarm information into the aubo_control log
00031 *
00032 * TraceLevel: \n
00033 * 0 - FATAL \n
00034 * 1 - ERROR \n
00035 * 2 - WARNING \n
00036 * 3 - INFO \n
00037 * 4 - DEBUG \n
00038 *
00039 * \~chinese code    error_stack \~english Code definitions refer to error_stack
00040 *
00041 * @param level
00042 * @param code
00043 * @param args
00044 * @return
00045 *
00046 * \~chinese @par Python    \~english @par Python function prototype
00047 * alarm(self: pyaubo_sdk.Trace, arg0: arcs::common_interface::TraceLevel,
00048 * arg1: int, arg2: List[str]) -> int
00049 *
00050 * \~chinese @par Lua    \~english @par Lua function prototype
00051 * alarm(level: number, code: number, args: table) -> nil
00052 */

```



```

00053 * \~chinese @par JSON-RPC \~english @par JSON-RPC request example
00054 * {"jsonrpc":"2.0","method":"rob1.Trace.alarm","params":["",1,["Error","Trajectory
00055 * planning failed!","1"]],"id":1}
00056 *
00057 * \~chinese @par JSON-RPC \~english @par JSON-RPC response example
00058 * {"id":1,"jsonrpc":"2.0","result":0}
00059 *
00060 */
00061 int alarm(TraceLevel level, int code,
00062           const std::vector<std::string> &args = {});
00063
00064 /**
00065 * @ingroup Trace
00066 * \chinese
00067 *
00068 *
00069 * @param msg
00070 * @return
00071 *
00072 * @par Python
00073 * textmsg(self: pyaubo_sdk.Trace, arg0: str) -> int
00074 *
00075 * @par Lua
00076 * textmsg(msg: string) -> nil
00077 *
00078 * @par JSON-RPC
00079 * {"jsonrpc":"2.0","method":"rob1.Trace.textmsg","params":["test"],"id":1}
00080 *
00081 * @par JSON-RPC
00082 * {"id":1,"jsonrpc":"2.0","result":0}
00083 * \endchinese
00084 * \english
00085 * print message into log
00086 *
00087 * @param msg message information
00088 * @return
00089 *
00090 * @par Python function prototype
00091 * textmsg(self: pyaubo_sdk.Trace, arg0: str) -> int
00092 *
00093 * @par Lua function prototype
00094 * textmsg(msg: string) -> nil
00095 *
00096 * @par JSON-RPC request example
00097 * {"jsonrpc":"2.0","method":"rob1.Trace.textmsg","params":["test"],"id":1}
00098 *
00099 * @par JSON-RPC response example
00100 * {"id":1,"jsonrpc":"2.0","result":0}
00101 * \endenglish
00102 */
00103 int textmsg(const std::string &msg);
00104
00105 /**
00106 * \~chinese \~english Notify the system
00107 *
00108 * @param msg
00109 * @return
00110 */
00111 int notify(const std::string &msg);
00112
00113 /**
00114 * @ingroup Trace
00115 * \chinese
00116 * RTDE
00117 *
00118 * @param level
00119 * @param title
00120 * @param msg
00121 * @param mode \n
00122 * 0: \n
00123 * 1: \n
00124 * 2: bool \n
00125 * 3: int \n
00126 * 4: double \n
00127 * 5: string \n
00128 * @return
00129 *
00130 * @par Python
00131 * popup(self: pyaubo_sdk.Trace, arg0: arcs::common_interface::TraceLevel,
00132 * arg1: str, arg2: str, arg3: int) -> int
00133 *
00134 * @par Lua
00135 * popup(level: number, title: string, msg: string, mode: number) -> nil
00136 *
00137 * @par JSON-RPC
00138 * {"jsonrpc":"2.0","method":"rob1.Trace.popup","params":["","Error","Trajectory
00139 * planning failed!","1"],"id":1}

```

```

00140 *
00141 * @par JSON-RPC
00142 * {"id":1,"jsonrpc":"2.0","result":0}
00143 * \endchinese
00144 * \english
00145 * Send a popup request to the connected RTDE client
00146 *
00147 * @param level
00148 * @param title
00149 * @param msg
00150 * @param mode mode \n
00151 * 0: normal mode \n
00152 * 1: blocking mode \n
00153 * 2: input mode bool \n
00154 * 3: input mode int \n
00155 * 4: input mode double \n
00156 * 5: input mode string \n
00157 * @return
00158 *
00159 * @par Python function prototype
00160 * popup(self: pyaubo_sdk.Trace, arg0: arcs::common_interface::TraceLevel,
00161 * arg1: str, arg2: str, arg3: int) -> int
00162 *
00163 * @par Lua function prototype
00164 * popup(level: number, title: string, msg: string, mode: number) -> nil
00165 *
00166 * @par JSON-RPC request example
00167 * {"jsonrpc":"2.0","method":"rob1.Trace.popup","params":["","Error","Trajectory
00168 * planning failed!"],"id":1}
00169 *
00170 * @par JSON-RPC response example
00171 * {"id":1,"jsonrpc":"2.0","result":0}
00172 * \endenglish
00173 */
00174 int popup(TraceLevel level, const std::string &title,
00175           const std::string &msg, int mode);
00176
00177 /**
00178 * @ingroup Trace
00179 * \chinese
00180 * peek AlarmInfo(    )
00181 *
00182 * last_time 0 AlarmInfo
00183 *
00184 * @param num
00185 * @param last_time
00186 * @return
00187 *
00188 * @par Python
00189 * peek(self: pyaubo_sdk.Trace, arg0: int, arg1: int) ->
00190 * List[arcs::common_interface::RobotMsg]
00191 *
00192 * @par Lua
00193 * peek(num: number, last_time: number) -> table
00194 *
00195 * @par JSON-RPC
00196 * {"jsonrpc":"2.0","method":"rob1.Trace.peek","params":[1,0],"id":1}
00197 *
00198 * @par JSON-RPC
00199 * [{"id":1,"jsonrpc":"2.0","result":[{"args":["RobotModeType.Running"],
00200 * "code":30045,"level":"INFO","source":"rob1","timestamp":5102883064300}]}]
00201 * \endchinese
00202 * \english
00203 * peek the latest AlarmInfo (after the last retrieval)
00204 *
00205 * When last_time is set as 0, retrieve all AlarmInfo
00206 *
00207 * @param num
00208 * @param last_time
00209 * @return
00210 *
00211 * @par Python function prototype
00212 * peek(self: pyaubo_sdk.Trace, arg0: int, arg1: int) ->
00213 * List[arcs::common_interface::RobotMsg]
00214 *
00215 * @par Lua function prototype
00216 * peek(num: number, last_time: number) -> table
00217 *
00218 * @par JSON-RPC request example
00219 * {"jsonrpc":"2.0","method":"rob1.Trace.peek","params":[1,0],"id":1}
00220 *
00221 * @par JSON-RPC response example
00222 * [{"id":1,"jsonrpc":"2.0","result":[{"args":["RobotModeType.Running"],
00223 * "code":30045,"level":"INFO","source":"rob1","timestamp":5102883064300}]}]
00224 * \endenglish
00225 */
00226 RobotMsgVector peek(size_t num, uint64_t last_time = 0);

```

```

00227
00228 protected:
00229     void *d_;
00230 };
00231
00232 using TracePtr = std::shared_ptr<Trace>;
00233
00234 } // namespace common_interface
00235 } // namespace arcs
00236 #endif

```

12.48 include/aubo/type_def.h File Reference

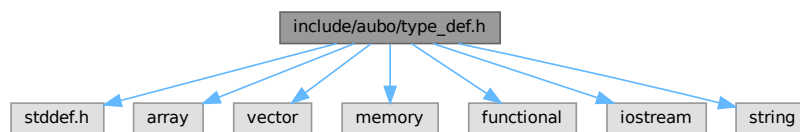
enum type definitions

```

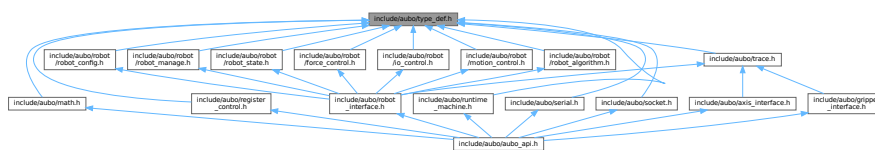
#include <stddef.h>
#include <array>
#include <vector>
#include <memory>
#include <functional>
#include <iostream>
#include <string>

```

Include dependency graph for type_def.h:



This graph shows which files directly or indirectly include this file:



Classes

- struct [arcs::common_interface::RobotSafetyParameterRange](#)
- struct [arcs::common_interface::WObjectData](#)
- struct [arcs::common_interface::VibrationRecalibrationParameter](#)
- struct [arcs::common_interface::CircleParameters](#)

Circular motion parameters definition.

- struct [arcs::common_interface::SpiralParameters](#)
- struct [arcs::common_interface::Enveloping](#)
- struct [arcs::common_interface::TrajConfig](#)

Trajectory configuration for payload identification.

- struct [arcs::common_interface::RobotMsg](#)

- struct [arcs::common_interface::GripperStatus](#)
- struct [arcs::common_interface::RtdeRecipe](#)
RTDE menu.
- class [arcs::common_interface::AuboException](#)
Custom exception class [AuboException](#).

Namespaces

- namespace [arcs](#)
- namespace [arcs::common_interface](#)

Macros

- `#define` [CARTESIAN_DOF](#) 6
Cartesian degree of freedom, 6 for x,y,z,rx,ry,rz.
- `#define` [SAFETY_PARAM_SELECT_NUM](#) 2
normal + reduced
- `#define` [SAFETY_PLANES_NUM](#) 8
Number of safety planes.
- `#define` [SAFETY_CUBIC_NUM](#) 10
Number of safety cubes.
- `#define` [TOOL_CONFIGURATION_NUM](#) 3
Number of tool configurations.
- `#define` [ENUM_AuboErrorCodes_DECLARES](#)
Error codes definition.
- `#define` [ENUM_RuntimeState_DECLARES](#)
The RuntimeState enum.
- `#define` [ENUM_RobotModeType_DECLARES](#)
- `#define` [ENUM_SafetyModeType_DECLARES](#)
- `#define` [ENUM_OperationalModeType_DECLARES](#)
Based on ISO 10218-1:2011(E) Section 5.7 Automatic: In automatic mode, the robot shall execute the task programme and the safeguarding measures shall be functioning.
- `#define` [ENUM_RobotControlModeType_DECLARES](#)
Robot control mode, the final control object.
- `#define` [ENUM_JointServoModeType_DECLARES](#)
- `#define` [ENUM_JointStateType_DECLARES](#)
- `#define` [ENUM_StandardInputAction_DECLARES](#)
- `#define` [ENUM_StandardOutputRunState_DECLARES](#)
- `#define` [ENUM_SafetyInputAction_DECLARES](#)
- `#define` [ENUM_SafetyOutputRunState_DECLARES](#)
- `#define` [ENUM_PayloadIdentifyMoveAxis_DECLARES](#)
- `#define` [ENUM_EnvelopingShape_DECLARES](#)
- `#define` [ENUM_TaskFrameType_DECLARES](#)
- `#define` [ENUM_TraceLevel_DECLARES](#)
- `#define` [ENUM_AxisModeType_DECLARES](#)
- `#define` [ENUM_SafeguedStopType_DECLARES](#)
- `#define` [ENUM_RobotEmergencyStopType_DECLARES](#)
- `#define` [ENUM_ITEM](#)(c, n, ...)
- `#define` [DECL_TO_STRING_FUNC](#)(ENUM)
- `#define` [ENUM_ITEM](#)(c, n, ...)
- `#define` [ENUM_ITEM](#)(n, v, s)
- `#define` [ENUM_ITEM](#)(n, v, s)
- `#define` [ENUM_ITEM](#)(n, v, s)

Typedefs

- using `arcs::common_interface::Vector3d` = `std::array<double, 3>`
- using `arcs::common_interface::Vector4d` = `std::array<double, 4>`
- using `arcs::common_interface::Vector3f` = `std::array<float, 3>`
- using `arcs::common_interface::Vector4f` = `std::array<float, 4>`
- using `arcs::common_interface::Vector6f` = `std::array<float, 6>`
- using `arcs::common_interface::ResultWithErrno` = `std::tuple<std::vector<double>, int>`
- using `arcs::common_interface::ResultWithErrno1` = `std::tuple<std::vector<std::vector<double>>, int>`
- using `arcs::common_interface::ResultWithErrno2` = `std::tuple<std::vector<std::string>, int>`
- using `arcs::common_interface::ResultWithErrno3` = `std::tuple<int, int>`
- using `arcs::common_interface::Payload`
- using `arcs::common_interface::ForceSensorCalibResult`
- using `arcs::common_interface::ForceSensorCalibResultWithError`
- using `arcs::common_interface::DynamicsModel`
- using `arcs::common_interface::Box` = `std::vector<double>`
- using `arcs::common_interface::Cylinder` = `std::vector<double>`
- using `arcs::common_interface::Sphere` = `std::vector<double>`
- using `arcs::common_interface::RobotMsgVector` = `std::vector<RobotMsg>`
- using `arcs::common_interface::GripperStatusVector` = `std::vector<GripperStatus>`

Enumerations

- enum `arcs::common_interface::AuboErrorCodes` : int { `arcs::common_interface::ENUM_AuboErrorCodes_DECLA` }
 - enum class `arcs::common_interface::RuntimeState` : int { `arcs::common_interface::ENUM_RuntimeState_DECLA` }
 - enum class `arcs::common_interface::RobotModeType` : int { `arcs::common_interface::ENUM_RobotModeType_DI` }
 - enum class `arcs::common_interface::AxisModeType` : int { `arcs::common_interface::ENUM_AxisModeType_DECLI` }
 - enum class `arcs::common_interface::SafetyModeType` : int { `arcs::common_interface::ENUM_SafetyModeType_DI` }
 - enum class `arcs::common_interface::OperationalModeType` : int { `arcs::common_interface::ENUM_OperationalMo` }
 - enum class `arcs::common_interface::RobotControlModeType` : int { `arcs::common_interface::ENUM_RobotContro` }
 - enum class `arcs::common_interface::JointServoModeType` : int { `arcs::common_interface::ENUM_JointServoMode` }
 - enum class `arcs::common_interface::JointStateType` : int { `arcs::common_interface::ENUM_JointStateType_DECLC` }
 - enum class `arcs::common_interface::StandardOutputRunState` : int { `arcs::common_interface::ENUM_StandardOu` }
 - enum class `arcs::common_interface::StandardInputAction` : int { `arcs::common_interface::ENUM_StandardInputA` }
- Safety Mode.
- Operational Mode.
- Robot Control Mode.
- Joint Servo Mode.
- Joint State.
- Standard Output Run State.
- The `StandardInputAction` enum.

- enum class `arcs::common_interface::SafetyInputAction` : int { `arcs::common_interface::ENUM_SafetyInputAction_`
}
 - enum class `arcs::common_interface::SafetyOutputRunState` : int { `arcs::common_interface::ENUM_SafetyOutputR`
}
 - enum `arcs::common_interface::TaskFrameType` { `arcs::common_interface::ENUM_TaskFrameType_DECLARES`
}
 - enum `arcs::common_interface::EnvelopingShape` : int { `arcs::common_interface::ENUM_EnvelopingShape_DECL`
}
 - enum `arcs::common_interface::PayloadIdentifyMoveAxis` : int { `arcs::common_interface::ENUM_PayloadIdentifyM`
}
 - enum `arcs::common_interface::TraceLevel` { `arcs::common_interface::ENUM_TraceLevel_DECLARES`
}
 - enum `arcs::common_interface::SafeguedStopType` : int { `arcs::common_interface::ENUM_SafeguedStopType_DE`
}
 - enum `arcs::common_interface::RobotEmergencyStopType` : int { `arcs::common_interface::ENUM_RobotEmergenc`
}
 - enum class `arcs::common_interface::ForceControlState` { `arcs::common_interface::Stopped` ,
`arcs::common_interface::Starting` , `arcs::common_interface::Stropping` , `arcs::common_interface::Running`
}
 - enum class `arcs::common_interface::RefFrameType` { `arcs::common_interface::None` , `arcs::common_interface::Tool`
 , `arcs::common_interface::Path` , `arcs::common_interface::Base` }
 - enum `arcs::common_interface::error_type` {
`arcs::common_interface::parse_error` = -32700 , `arcs::common_interface::invalid_request` = -32600
 , `arcs::common_interface::method_not_found` = -32601 , `arcs::common_interface::invalid_params`
 = -32602 ,
`arcs::common_interface::internal_error` = -32603 , `arcs::common_interface::server_error` = -32500
 , `arcs::common_interface::invalid` = -32400 }
- Error type.
- enum `arcs::common_interface::ExceptionCode` {
`arcs::common_interface::EC_DISCONNECTED` = -1 , `arcs::common_interface::EC_NOT_LOGINED`
 = -2 , `arcs::common_interface::EC_INVALID_SOCKET` = -3 , `arcs::common_interface::EC_REQUEST_BUSY`
 = -4 ,
`arcs::common_interface::EC_SEND_FAILED` = -5 , `arcs::common_interface::EC_RECV_TIMEOUT`
 = -6 , `arcs::common_interface::EC_RECV_ERROR` = -7 , `arcs::common_interface::EC_PARSE_ERROR`
 = -8 ,
`arcs::common_interface::EC_INVALID_REQUEST` = -9 , `arcs::common_interface::EC_METHOD_NOT_FOUN`
 = -10 , `arcs::common_interface::EC_INVALID_PARAMS` = -11 , `arcs::common_interface::EC_INTERNAL_ERR`
 = -12 ,
`arcs::common_interface::EC_SERVER_ERROR` = -13 , `arcs::common_interface::EC_INVALID`
 = -14 }
- Exception code.

Functions

- `std::ostream & arcs::common_interface::operator<<` (`std::ostream &os`, const `RobotSafetyParameterRange`
&`vd`)
- `std::ostream & arcs::common_interface::operator<<` (`std::ostream &os`, `WObjectData` p)
- `std::ostream & arcs::common_interface::operator<<` (`std::ostream &os`, const `VibrationRecalibrationParameter`
p)
- `std::ostream & arcs::common_interface::operator<<` (`std::ostream &os`, `CircleParameters` p)
- `std::ostream & arcs::common_interface::operator<<` (`std::ostream &os`, `SpiralParameters` p)
- `std::ostream & arcs::common_interface::operator<<` (`std::ostream &os`, `Enveloping` p)
- `std::ostream & arcs::common_interface::operator<<` (`std::ostream &os`, `TrajConfig` p)
- `const char * arcs::common_interface::returnValue2Str` (int retval)

12.48.1 Detailed Description

enum type definitions

Definition in file [type_def.h](#).

12.48.2 Macro Definition Documentation

12.48.2.1 CARTESIAN_DOF

```
#define CARTESIAN_DOF 6
```

Cartesion degree of freedom, 6 for x,y,z,rx,ry,rz.

Definition at line 25 of file [type_def.h](#).

12.48.2.2 DECL_TO_STRING_FUNC

```
#define DECL_TO_STRING_FUNC(  
    ENUM)
```

Value:

```
inline std::string toString(ENUM v)
{
    using T = ENUM;
    std::string name = #ENUM " ";
    ENUM_##ENUM##_DECLARES
    return #ENUM ".Unkown";
}
inline std::ostream &operator<<(std::ostream &os, ENUM v) \
{
    os << toString(v);
    return os;
}
```

Definition at line 665 of file [type_def.h](#).

12.48.2.3 ENUM_AuboErrorCodes_DECLARES

```
#define ENUM_AuboErrorCodes_DECLARES
```

Error codes definition.

whole number is warning, negative number is error, 0 is no error and no warning

Definition at line 232 of file [type_def.h](#).

12.48.2.4 ENUM_AxisModeType_DECLARES

```
#define ENUM_AxisModeType_DECLARES
```

Value:

```
ENUM_ITEM(NoController, -1, " ", aubo_control NoController) \
ENUM_ITEM(Disconnected, 0, " ") \
ENUM_ITEM(PowerOff, 1, " ") \
ENUM_ITEM(BrakeReleasing, 2, " ") \
ENUM_ITEM(Idle, 3, " ") \
ENUM_ITEM(Running, 4, " ") \
ENUM_ITEM(Fault, 5, " ")
```

Definition at line 524 of file [type_def.h](#).

12.48.2.5 ENUM_EnvelopingShape_DECLARES

```
#define ENUM_EnvelopingShape_DECLARES
```

Value:

```
ENUM_ITEM(Cube, 1, " ") \
ENUM_ITEM(Column, 2, " ") \
ENUM_ITEM(Stl, 3, " STL ")
```

Definition at line 497 of file [type_def.h](#).

12.48.2.6 ENUM_ITEM [1/5]

```
#define ENUM_ITEM(
    c,
    n,
    ...)

```

Value:

```
if (v == T::c) { \
    return name + #c; \
}
```

Definition at line 546 of file [type_def.h](#).

12.48.2.7 ENUM_ITEM [2/5]

```
#define ENUM_ITEM(
    c,
    n,
    ...)

```

Value:

```
c = n,
```

Definition at line 546 of file [type_def.h](#).

12.48.2.8 ENUM_ITEM [3/5]

```
#define ENUM_ITEM(
    n,
    v,
    s)

```

Value:

```
if (abs(retval) == v) \
    index = n##_INDEX;
```

Definition at line 546 of file [type_def.h](#).

12.48.2.9 ENUM_ITEM [4/5]

```
#define ENUM_ITEM(
    n,
    v,
    s)
```

Value:

n##_INDEX,

Definition at line 546 of file [type_def.h](#).

12.48.2.10 ENUM_ITEM [5/5]

```
#define ENUM_ITEM(
    n,
    v,
    s)
```

Value:

s,

Definition at line 546 of file [type_def.h](#).

12.48.2.11 ENUM_JointServoModeType_DECLARES

```
#define ENUM_JointServoModeType_DECLARES
```

Value:

```
ENUM_ITEM(Unknown, -1, " ") \
ENUM_ITEM(Open, 0, " (open loop mode)") \
ENUM_ITEM(Current, 1, " (current servo mode)") \
ENUM_ITEM(Velocity, 2, " (speed servo mode)") \
ENUM_ITEM(Position, 3, " (position servo mode)") \
ENUM_ITEM(Torque, 4, " (torque servo mode)") \
```

Definition at line 409 of file [type_def.h](#).

12.48.2.12 ENUM_JointStateType_DECLARES

```
#define ENUM_JointStateType_DECLARES
```

Value:

```
ENUM_ITEM(Poweroff, 0, " (node not conected to interface board or already powered off)") \
ENUM_ITEM(Idle, 2, " (node idle)") \
ENUM_ITEM(Fault, 3, " , (node error, node stopped servo move, brake engaged)") \
ENUM_ITEM(Running, 4, " (node servo)") \
ENUM_ITEM(Bootload, 5, " bootload , (node bootloader state, pause all communication)") \
```

Definition at line 417 of file [type_def.h](#).

12.48.2.13 ENUM_OperationalModeType_DECLARES

```
#define ENUM_OperationalModeType_DECLARES
```

Value:

```
ENUM_ITEM(Disabled, 0, " : Operational Mode") \
ENUM_ITEM(Automatic, 1, " : , (auto mode: robot normal operation, speed will not be limited)") \
ENUM_ITEM(Manual, 2, " : (T1), (T2) (manual mode: robot programming teaching mode (T1), robot \
running speed will be limited or robot program verification mode (T2)") \
```

Based on ISO 10218-1:2011(E) Section 5.7 Automatic: In automatic mode, the robot shall execute the task programme and the safeguarding measures shall be functioning.

Automatic operation shall be prevented if any stop condition is detected. Switching from this mode shall result in a stop.

Definition at line 391 of file [type_def.h](#).

12.48.2.14 ENUM_PayloadIdentifyMoveAxis_DECLARES

```
#define ENUM_PayloadIdentifyMoveAxis_DECLARES
```

Value:

```
ENUM_ITEM(Joint_2_6, 0, " 2 6 ") \
ENUM_ITEM(Joint_3_6, 1, " 3 6 ") \
ENUM_ITEM(Joint_4_6, 2, " 4 6 ") \
ENUM_ITEM(Joint_4_5_6, 3, " 4 5 6 ") \
```

Definition at line 491 of file [type_def.h](#).

12.48.2.15 ENUM_RobotControlModeType_DECLARES

```
#define ENUM_RobotControlModeType_DECLARES
```

Value:

```
ENUM_ITEM(Unknown, 0, " (unknown control mode)") \
ENUM_ITEM(Position, 1, " movej (position control)") \
ENUM_ITEM(Speed, 2, " speedj/speedl (speed control)") \
ENUM_ITEM(Servo, 3, " servoj (position control)") \
ENUM_ITEM(Freedrive, 4, " freedrive_mode") \
ENUM_ITEM(Force, 5, " force_mode") \
ENUM_ITEM(Torque, 6, " (joint torque control)") \
ENUM_ITEM(Collision, 7, " (collision mode)") \
```

Robot control mode, the final control object.

Definition at line 399 of file [type_def.h](#).

12.48.2.16 ENUM_RobotEmergencyStopType_DECLARES

```
#define ENUM_RobotEmergencyStopType_DECLARES
```

Value:

```
ENUM_ITEM(RobotEmergencyStopNone, 0, " ") \
ENUM_ITEM(RobotEmergencyStopControlBox, 1, " ( )") \
ENUM_ITEM(RobotEmergencyStopTeachPendant, 2, " ( )") \
ENUM_ITEM(RobotEmergencyStopHandle, 3, " ( )") \
ENUM_ITEM(RobotEmergencyStopEI, 4, " ( IO )") \
```

Definition at line 539 of file [type_def.h](#).

12.48.2.17 ENUM_RobotModeType_DECLARES

```
#define ENUM_RobotModeType_DECLARES
```

Value:

```
ENUM_ITEM(NoController, -1, " , aubo_control NoController") \
ENUM_ITEM(Disconnected, 0, " ( EtherCAT )") \
ENUM_ITEM(ConfirmSafety, 1, " , ") \
ENUM_ITEM(Booting, 2, " ") \
ENUM_ITEM(PowerOff, 3, " ") \
ENUM_ITEM(PowerOn, 4, " , ( ), ") \
ENUM_ITEM(Idle, 5, " , ( ), , ") \
ENUM_ITEM(BrakeReleasing, 6, " , ") \
ENUM_ITEM(BackDrive, 7, " , ") \
ENUM_ITEM(Running, 8, " , , ") \
ENUM_ITEM(Maintaince, 9, " : , ") \
ENUM_ITEM(Error, 10, " ") \
ENUM_ITEM(PowerOffing, 11, " ")
```

The RobotModeType enum

Hardware related

Definition at line 347 of file [type_def.h](#).

12.48.2.18 ENUM_RuntimeState_DECLARES

```
#define ENUM_RuntimeState_DECLARES
```

Value:

```
ENUM_ITEM(Running, 0, " ") \
ENUM_ITEM(Retracting, 1, " ") \
ENUM_ITEM(Pausing, 2, " ") \
ENUM_ITEM(Paused, 3, " ") \
ENUM_ITEM(Stepping, 4, " ") \
ENUM_ITEM(Stopping, 5, " ( )") \
ENUM_ITEM(Stopped, 6, " ") \
ENUM_ITEM(Aborting, 7, " ( )")
```

The RuntimeState enum.

Definition at line 324 of file [type_def.h](#).

12.48.2.19 ENUM_SafeguedStopType_DECLARES

```
#define ENUM_SafeguedStopType_DECLARES
```

Value:

```
ENUM_ITEM(None, 0, " ") \
ENUM_ITEM(SafeguedStopIOInput, 1, " (IO )") \
ENUM_ITEM(SafeguedStop3PE, 2, " ( )") \
ENUM_ITEM(SafeguedStopOperational, 3, " ( )")
```

Definition at line 533 of file [type_def.h](#).

12.48.2.20 ENUM_SafetyInputAction_DECLARES

```
#define ENUM_SafetyInputAction_DECLARES
```

Value:

```
ENUM_ITEM(Unassigned, 0, " ") \
ENUM_ITEM(EmergencyStop, 1, " ") \
ENUM_ITEM(SafeguardStop, 2, " ", " ") \
ENUM_ITEM(SafeguardReset, 3, " ", " ") \
ENUM_ITEM(ThreePositionSwitch, 4, "3 ") \
ENUM_ITEM(OperationalMode, 5, " ") \
ENUM_ITEM(HandGuide, 6, " ") \
ENUM_ITEM(ReducedMode, 7, " 1( ) 0 ") \
ENUM_ITEM(AutomaticModeSafeguardStop, 8, " ( )") \
ENUM_ITEM(AutomaticModeSafeguardReset, 9, " ( )")
```

Definition at line 468 of file [type_def.h](#).

12.48.2.21 ENUM_SafetyModeType_DECLARES

```
#define ENUM_SafetyModeType_DECLARES
```

Value:

```
ENUM_ITEM(Undefined, 0, " ") \
ENUM_ITEM(Normal, 1, " ") \
ENUM_ITEM(ReducedMode, 2, " ") \
ENUM_ITEM(Recovery, 3, " ", "recovery ") \
ENUM_ITEM(Violation, 4, " ", " ") \
ENUM_ITEM(ProtectiveStop, 5, " ", " ") \
ENUM_ITEM(SafeguardStop, 6, "IO ", " ") \
ENUM_ITEM(SystemEmergencyStop, 7, " ( ), ") \
ENUM_ITEM(RobotEmergencyStop, 8, " ", " ") \
ENUM_ITEM(Fault, 9, " ", " ")
```

Definition at line 362 of file [type_def.h](#).

12.48.2.22 ENUM_SafetyOutputRunState_DECLARES

```
#define ENUM_SafetyOutputRunState_DECLARES
```

Value:

```
ENUM_ITEM(Unassigned, 0, " ") \
ENUM_ITEM(SystemEmergencyStop, 1, " ") \
ENUM_ITEM(NotSystemEmergencyStop, 2, " ") \
ENUM_ITEM(RobotMoving, 3, " 0.1rad/s") \
ENUM_ITEM(RobotNotMoving, 4, " 0.1rad/s") \
ENUM_ITEM(ReducedMode, 5, " ") \
ENUM_ITEM(NotReducedMode, 6, " ") \
ENUM_ITEM(SafeHome, 7, " Home ") \
ENUM_ITEM(RobotNotStopping, 8, " ")
```

Definition at line 480 of file [type_def.h](#).

12.48.2.23 ENUM_StandardInputAction_DECLARES

```
#define ENUM_StandardInputAction_DECLARES
```

Value:

```
ENUM_ITEM(Default, 0, " ") \
ENUM_ITEM(Handguide, 1, " ") \
ENUM_ITEM(GoHome, 2, " ") \
ENUM_ITEM(StartProgram, 3, " ") \
ENUM_ITEM(StopProgram, 4, " ") \
ENUM_ITEM(PauseProgram, 5, " ") \
ENUM_ITEM(PopupDismiss, 6, " ") \
ENUM_ITEM(PowerOn, 7, " / ") \
ENUM_ITEM(PowerOff, 8, " / ") \
ENUM_ITEM(ResumeProgram, 9, " ") \
ENUM_ITEM(SlowDown1, 10, " 1 ") \
ENUM_ITEM(SlowDown2, 11, " 2 ") \
ENUM_ITEM(SafeStop, 12, " ") \
ENUM_ITEM(RunningGuard, 13, " ") \
ENUM_ITEM(MoveToFirstPoint, 14, " ") \
ENUM_ITEM(xSlowDown1, 15, " 1 ") \
ENUM_ITEM(xSlowDown2, 16, " 2 ") \
ENUM_ITEM(ConveyorTrack, 17, " ") \
ENUM_ITEM(xConveyorTrack, 18, " ") \
ENUM_ITEM(UnlockProtectiveStop, 19, " ") \
ENUM_ITEM(ArbitraryResumeProgram, 20, " ")
```

Definition at line 424 of file [type_def.h](#).

12.48.2.24 ENUM_StandardOutputRunState_DECLARES

```
#define ENUM_StandardOutputRunState_DECLARES
```

Value:

```
ENUM_ITEM(None, 0, " ") \
ENUM_ITEM(StopLow, 1, " ") \
ENUM_ITEM(StopHigh, 2, " ") \
ENUM_ITEM(RunningHigh, 3, " ") \
ENUM_ITEM(PausedHigh, 4, " ") \
ENUM_ITEM(AtHome, 5, " ") \
ENUM_ITEM(Handguiding, 6, " ") \
ENUM_ITEM(PowerOn, 7, " ") \
ENUM_ITEM(RobotEmergencyStop, 8, " ") \
ENUM_ITEM(SystemEmergencyStop, 9, " ") \
ENUM_ITEM(InternalEmergencyStop, 8, " ") \
ENUM_ITEM(ExternalEmergencyStop, 9, " ") \
ENUM_ITEM(SystemError, 10, " ") \
ENUM_ITEM(NotSystemError, 11, " ") \
ENUM_ITEM(RobotOperable, 12, " ") \
ENUM_ITEM(OperationalMode, 13, " ") \
ENUM_ITEM(SafeguardStop, 14, " ") \
ENUM_ITEM(ProtectiveStop, 15, " ") \
ENUM_ITEM(ProgramAborted, 16, " ")
```

Definition at line 447 of file [type_def.h](#).

12.48.2.25 ENUM_TaskFrameType_DECLARES

```
#define ENUM_TaskFrameType_DECLARES
```

Value:

```
ENUM_ITEM(NONE, 0, " ") \
ENUM_ITEM(POINT_FORCE, 1, " , y TCP , x z " \
" , y TCP , x z " \
" TCP 10mm" \
" X, X Y , X Y , " \
" Z, Y-X Y-Z , ") \
ENUM_ITEM(FRAME_FORCE, 2, " SIMPLE_FORC") \
ENUM_ITEM(MOTION_FORCE, 3, " , x TCP x-y y , x-y ") \
ENUM_ITEM(TOOL_FORCE, 4, " ")
```

Definition at line 502 of file [type_def.h](#).

12.48.2.26 ENUM_TraceLevel_DECLARES

```
#define ENUM_TraceLevel_DECLARES
```

Value:

```
ENUM_ITEM(FATAL, 0, "") \
ENUM_ITEM(ERROR, 1, "") \
ENUM_ITEM(WARNING, 2, "") \
ENUM_ITEM(INFO, 3, "") \
ENUM_ITEM(DEBUG, 4, "")
```

Definition at line 517 of file [type_def.h](#).

12.48.2.27 SAFETY_CUBIC_NUM

```
#define SAFETY_CUBIC_NUM 10
```

Number of safety cubes.

Definition at line 28 of file [type_def.h](#).

Referenced by [arcs::common_interface::RobotSafetyParameterRange::RobotSafetyParameterRange\(\)](#).

12.48.2.28 SAFETY_PARAM_SELECT_NUM

```
#define SAFETY_PARAM_SELECT_NUM 2
```

normal + reduced

Definition at line 26 of file [type_def.h](#).

Referenced by [arcs::common_interface::RobotSafetyParameterRange::RobotSafetyParameterRange\(\)](#).

12.48.2.29 SAFETY_PLANES_NUM

```
#define SAFETY_PLANES_NUM 8
```

Number of safety planes.

Definition at line 27 of file [type_def.h](#).

Referenced by [arcs::common_interface::RobotSafetyParameterRange::RobotSafetyParameterRange\(\)](#).

12.48.2.30 TOOL_CONFIGURATION_NUM

```
#define TOOL_CONFIGURATION_NUM 3
```

Number of tool configurations.

Definition at line 29 of file [type_def.h](#).

Referenced by [arcs::common_interface::RobotSafetyParameterRange::RobotSafetyParameterRange\(\)](#).

12.49 type_def.h

[Go to the documentation of this file.](#)

```

00001 /** @file type_def.h
00002  * \~chinese @brief \~english @brief enum type definitions
00003  */
00004 #ifndef AUBO_SDK_TYPE_DEF_H
00005 #define AUBO_SDK_TYPE_DEF_H
00006
00007 #include <stddef.h>
00008 #include <array>
00009 #include <vector>
00010 #include <memory>
00011 #include <functional>
00012 #include <iostream>
00013 #include <string>
00014
00015 #ifdef _MSC_VER
00016 #pragma execution_character_set("utf-8")
00017 #endif
00018
00019 // clang-format off
00020
00021 namespace arcs {
00022 namespace common_interface {
00023
00024 /// Cartesion degree of freedom, 6 for x,y,z,rx,ry,rz
00025 #define CARTESIAN_DOF 6
00026 #define SAFETY_PARAM_SELECT_NUM 2 ///< \~chinese + \~english normal + reduced
00027 #define SAFETY_PLANES_NUM 8 ///< \~chinese \~english Number of safety planes
00028 #define SAFETY_CUBIC_NUM 10 ///< \~chinese \~english Number of safety cubes
00029 #define TOOL_CONFIGURATION_NUM 3 ///< \~chinese \~english Number of tool configurations
00030
00031 using Vector3d = std::array<double, 3>;
00032 using Vector4d = std::array<double, 4>;
00033 using Vector3f = std::array<float, 3>;
00034 using Vector4f = std::array<float, 4>;
00035 using Vector6f = std::array<float, 6>;
00036
00037 struct RobotSafetyParameterRange
00038 {
00039     RobotSafetyParameterRange()
00040     {
00041         for (int i = 0; i < SAFETY_PARAM_SELECT_NUM; i++) {
00042             params[i].power = 0.;
00043             params[i].momentum = 0.;
00044             params[i].stop_time = 0.;
00045             params[i].stop_distance = 0.;
00046             params[i].reduced_entry_time = 0.;
00047             params[i].reduced_entry_distance = 0.;
00048             params[i].tcp_speed = 0.;
00049             params[i].elbow_speed = 0.;
00050             params[i].tcp_force = 0.;
00051             params[i].elbow_force = 0.;
00052             std::fill(params[i].qmin.begin(), params[i].qmin.end(), 0.);
00053             std::fill(params[i].qmax.begin(), params[i].qmax.end(), 0.);
00054             std::fill(params[i].qddmax.begin(), params[i].qddmax.end(), 0.);
00055             std::fill(params[i].joint_torque.begin(),
00056                     params[i].joint_torque.end(), 0.);
00057             params[i].tool_orientation.fill(0.);
00058             params[i].tool_deviation = 0.;
00059             for (int j = 0; j < SAFETY_PLANES_NUM; j++) {
00060                 params[i].planes[j].fill(0.);
00061                 params[i].restrict_elbow[j] = 0;
00062             }
00063         }
00064         for (int i = 0; i < SAFETY_PLANES_NUM; i++) {
00065             trigger_planes[i].plane.fill(0.);
00066             trigger_planes[i].restrict_elbow = 0;
00067         }
00068         for (int i = 0; i < SAFETY_CUBIC_NUM; i++) {
00069             cubic[i].orig.fill(0.);
00070             cubic[i].size.fill(0.);
00071             cubic[i].restrict_elbow = 0;
00072         }
00073         for (int i = 0; i < TOOL_CONFIGURATION_NUM; i++) {
00074             tools[i].fill(0.);
00075         }
00076
00077         tool_inclination = 0.;
00078         tool_azimuth = 0.;
00079         std::fill(safety_home.begin(), safety_home.end(), 0.);
00080
00081         safety_input_emergency_stop = 0;
00082         safety_input_safeguard_stop = 0;

```

```

00083     safety_input_safeguard_reset = 0;
00084     safety_input_auto_safeguard_stop = 0;
00085     safety_input_auto_safeguard_reset = 0;
00086     safety_input_three_position_switch = 0;
00087     safety_input_operational_mode = 0;
00088     safety_input_reduced_mode = 0;
00089     safety_input_handguide = 0;
00090
00091     safety_output_emergency_stop = 0;
00092     safety_output_not_emergency_stop = 0;
00093     safety_output_robot_moving = 0;
00094     safety_output_robot_steady = 0;
00095     safety_output_reduced_mode = 0;
00096     safety_output_not_reduced_mode = 0;
00097     safety_output_safe_home = 0;
00098     safety_output_robot_not_stopping = 0;
00099     safety_output_safeguard_stop = 0;
00100
00101     tp_3pe_for_handguide = 1;
00102     allow_manual_high_speed = 0;
00103 }
00104
00105 uint32_t crc32{ 0 };
00106
00107 /// \-chinese 2 , 0 \-english At most 2
00108 /// sets of parameters can be saved, default is the 0th set
00109 struct
00110 {
00111     float power; ///< \-chinese \-english sum of joint torques times joint angular speeds
00112     float momentum; ///< \-chinese \-english robot momentum limit
00113     float stop_time; ///< \-chinese ms \-english stop time in milliseconds
00114     float stop_distance; ///< \-chinese m \-english stop distance in meters
00115     float reduced_entry_time; ///< \-chinese \-english maximum time to enter reduced mode
00116     float reduced_entry_distance; ///< \-chinese ( ) \-english maximum distance to enter reduced mode (can
be triggered by safety planes)
00117     float tcp_speed;
00118     float elbow_speed;
00119     float tcp_force;
00120     float elbow_force;
00121     std::vector<float> qmin;
00122     std::vector<float> qmax;
00123     std::vector<float> qdmax;
00124     std::vector<float> joint_torque;
00125     Vector3f tool_orientation; ///<
00126     float tool_deviation;
00127     Vector4f planes[SAFETY_PLANES_NUM]; /// x,y,z,displacement
00128     int restrict_elbow[SAFETY_PLANES_NUM];
00129 } params[SAFETY_PARAM_SELECT_NUM];
00130
00131 /// \-chinese 8 \-english 8 trigger planes
00132 struct
00133 {
00134     Vector4f plane; /// x,y,z,displacement
00135     int restrict_elbow;
00136 } trigger_planes[SAFETY_PLANES_NUM];
00137
00138 struct
00139 {
00140     Vector6f orig; ///< \-chinese (x,y,z,rx,ry,rz) \-english origin of the cubic (x,y,z,rx,ry,rz)
00141     Vector3f size; ///< \-chinese (x,y,z) \-english size of the cubic (x,y,z)
00142     int restrict_elbow;
00143 } cubic[SAFETY_CUBIC_NUM]; ///< \-chinese 10 \-english 10 safety spaces
00144
00145 /// \-chinese 3 \-english 3 tools
00146 Vector4f tools[TOOL_CONFIGURATION_NUM]; /// x,y,z,radius
00147
00148 float tool_inclination{
00149     0.
00150 };
00151 ///< \-chinese \-english inclination angle
00152 float tool_azimuth{ 0. }; ///< \-chinese \-english azimuth angle
00153 std::vector<float> safety_home;
00154
00155 /// \-chinese IO \-english Configurable IO input
00156 /// and output safety functions
00157 uint32_t safety_input_emergency_stop;
00158 uint32_t safety_input_safeguard_stop;
00159 uint32_t safety_input_safeguard_reset;
00160 uint32_t safety_input_auto_safeguard_stop;
00161 uint32_t safety_input_auto_safeguard_reset;
00162 uint32_t safety_input_three_position_switch;
00163 uint32_t safety_input_operational_mode;
00164 uint32_t safety_input_reduced_mode;
00165 uint32_t safety_input_handguide;
00166
00167 uint32_t safety_output_emergency_stop;
00168 uint32_t safety_output_not_emergency_stop;
00169 uint32_t safety_output_robot_moving;

```



```

00169     uint32_t safety_output_robot_steady;
00170     uint32_t safety_output_reduced_mode;
00171     uint32_t safety_output_not_reduced_mode;
00172     uint32_t safety_output_safe_home;
00173     uint32_t safety_output_robot_not_stopping;
00174     uint32_t safety_output_safeguard_stop;
00175
00176     int tp_3pe_for_handguide; ///< ~chinese          \~english Whether to use the three-position switch of the teach
                                pendant as a hand guiding function switch
00177     int allow_manual_high_speed; ///< ~chinese          /~english Allow high-speed operation in manual mode
00178 };
00179
00180 inline std::ostream &operator<<(std::ostream &os,
00181                                 const RobotSafetyParameterRange &vd)
00182 {
00183     // os << (int)vd;
00184     return os;
00185 }
00186
00187 struct WObjectData
00188 {
00189     ///< ~chinese          \~english Whether it is an external tool
00190     bool remote_tool{ false };
00191
00192     ///< ~chinese          \~english Coupled with the workpiece
00193     ///< coordinate system
00194     std::string attach_frame{ "" };
00195
00196     ///< ~english User coordinate system. \~chinese
00197     ///< ~chinese robhold false, uframe world \~english
00198     ///< If robhold is false, the values of uframe are based on world.
00199     ///< ~chinese uframe flange \~english Otherwise, the
00200     ///< values of uframe are based on flange.
00201     std::vector<double> user_coord{ std::vector<double>(6, 0) };
00202
00203     ///< ~chinese          uframe \~english tool coordinate system,
00204     ///< based on uframe
00205     std::vector<double> obj_coord{ std::vector<double>(6, 0) };
00206 };
00207
00208 inline std::ostream &operator<<(std::ostream &os, WObjectData p)
00209 {
00210     return os;
00211 }
00212
00213
00214 struct VibrationRecalibrationParameter{
00215     double mass; //
00216     std::vector<double> cog; //
00217     std::vector<double> inertia; //
00218     std::vector<std::vector<double>> points; //
00219     std::vector<std::vector<double>> stiff_section; //
00220     std::vector<std::vector<double>> stiff_param; //
00221 };
00222
00223 inline std::ostream &operator<<(std::ostream &os, const VibrationRecalibrationParameter p)
00224 {
00225     return os;
00226 }
00227
00228 ///< ~chinese          \~english Error codes definition
00229 ///<
00230 ///< ~chinese          0          \~english whole
00231 ///< number is warning, negative number is error, 0 is no error and no warning
00232 #define ENUM_AuboErrorCodes_DECLARES
00233     ENUM_ITEM(AUBO_OK, 0, "Success")
00234     ENUM_ITEM(AUBO_BAD_STATE, 1, "State error")
00235     ENUM_ITEM(AUBO_QUEUE_FULL, 2, "Planning queue full")
00236     ENUM_ITEM(AUBO_BUSY, 3, "The previous command is executing")
00237     ENUM_ITEM(AUBO_TIMEOUT, 4, "Timeout")
00238     ENUM_ITEM(AUBO_INVL_ARGUMENT, 5, "Invalid parameters")
00239     ENUM_ITEM(AUBO_NOT_IMPLEMENT, 6, "Interface not implemented")
00240     ENUM_ITEM(AUBO_NO_ACCESS, 7, "Cannot access")
00241     ENUM_ITEM(AUBO_CONN_REFUSED, 8, "Connection refused")
00242     ENUM_ITEM(AUBO_CONN_RESET, 9, "Connection is reset")
00243     ENUM_ITEM(AUBO_INPROGRESS, 10, "Execution in progress")
00244     ENUM_ITEM(AUBO_EIO, 11, "Input/Output error")
00245     ENUM_ITEM(AUBO_NOBUFFS, 12, "")
00246     ENUM_ITEM(AUBO_REQUEST_IGNORE, 13, "Request was ignored")
00247     ENUM_ITEM(AUBO_ALGORITHM_PLAN_FAILED, 14,
00248             "Motion planning algorithm error")
00249     ENUM_ITEM(AUBO_VERSION_INCOMPAT, 15, "Interface version unmatched")
00250     ENUM_ITEM(AUBO_DIMENSION_ERR, 16,
00251             "Input parameter dimension is incorrect")
00252     ENUM_ITEM(AUBO_SINGULAR_ERR, 17, "Input configuration may be singular")
00253     ENUM_ITEM(AUBO_POS_BOUND_ERR, 18,
00254             "Input position boundary exceeds the limit range")

```

```

00255 ENUM_ITEM(AUBO_INIT_POS_ERR, 19, "Initial position input is unreasonable") \
00256 ENUM_ITEM(AUBO_ELP_SETTING_ERR, 20, "Envelope body setting error") \
00257 ENUM_ITEM(AUBO_TRAJ_GEN_FAIL, 21, "Trajectory generation failed") \
00258 ENUM_ITEM(AUBO_TRAJ_SELF_COLLISION, 22, "Trajectory self collision") \
00259 ENUM_ITEM(
00260     AUBO_IK_NO_CONVERGE, 23,
00261     "Inverse kinematics computation did not converge; computation failed") \
00262 ENUM_ITEM(AUBO_IK_OUT_OF_RANGE, 24,
00263     "Inverse kinematics result out of robot range") \
00264 ENUM_ITEM(AUBO_IK_CONFIG_DISMATCH, 25,
00265     "Inverse kinematics input configuration contains errors") \
00266 ENUM_ITEM(AUBO_IK_JACOBIAN_FAILED, 26,
00267     "The calculation of the inverse Jacobian matrix failed") \
00268 ENUM_ITEM(AUBO_IK_NO_SOLU, 27,
00269     "The target point has solutions, but it has exceeded the joint " \
00270     "limit conditions") \
00271 ENUM_ITEM(AUBO_IK_UNKNOWN_ERROR, 28, "Inverse kinematics unkown error") \
00272 ENUM_ITEM(AUBO_MOVE_IGNORED_SERVOMODE, 29,
00273     "Robot is in servo mode where movement is disabled") \
00274 ENUM_ITEM(AUBO_MOVE_INVALID_SPEED_VALUE, 30,
00275     "Movement speed must be a valid positive value (cannot be zero or negative)") \
00276 ENUM_ITEM(AUBO_MOVE_INVALID_ACCELERATION_VALUE, 31,
00277     "Movement acceleration must be a valid positive value (cannot be zero or negative)") \
00278 ENUM_ITEM(AUBO_INST_QUEUED, 100, "Instruction pused into queue succeed") \
00279 ENUM_ITEM(AUBO_INTERNAL_ERR, 101, "Internal error caused by alg .etc.") \
00280 ENUM_ITEM(AUBO_BADSTATE_THREAD_DETACHED, 200,
00281     "Bad state: Operation not allowed on detached thread") \
00282 ENUM_ITEM(AUBO_BADSTATE_THREAD_KILLED, 201,
00283     "Bad state: Operation not allowed on killed thread") \
00284 ENUM_ITEM(AUBO_BADSTATE_TASK_NOT_FOUND, 202,
00285     "Bad state: Specified task id does not exist in the task queue") \
00286 ENUM_ITEM(AUBO_BADSTATE_RTM_NOT_STARTED, 203,
00287     "Bad state: RuntimeMachine has not been started yet") \
00288 ENUM_ITEM(AUBO_BADSTATE_RTM_NOT_STOPPED, 204,
00289     "Bad state: RuntimeMachine must be in Stopped state for this operation") \
00290 ENUM_ITEM(AUBO_BADSTATE_RTM_NOT_PAUSED, 205,
00291     "Bad state: RuntimeMachine must be in Paused state for this operation") \
00292 ENUM_ITEM(AUBO_BADSTATE_RTM_ABORTING, 206, "Bad state: RuntimeMachine is in aborting state") \
00293 ENUM_ITEM(AUBO_BADSTATE_PSTOP, 207, "Bad state: Operation blocked by protective stop") \
00294 ENUM_ITEM(AUBO_BADSTATE_ROBOT_ESTOP, 208, "Bad state: Robot emergency stop triggered") \
00295 ENUM_ITEM(AUBO_BADSTATE_SYSTEM_ESTOP, 209, "Bad state: System emergency stop triggered") \
00296 ENUM_ITEM(AUBO_BADSTATE_INVALID_ROBOT_MODE, 210,
00297     "Bad state: Robot is not in required operation mode") \
00298 ENUM_ITEM(AUBO_BADSTATE_INVALID_SAFETY_MODE, 211,
00299     "Bad state: Robot is not in required safety mode") \
00300 ENUM_ITEM(AUBO_BADSTATE_ROBOT_NOT_RUNNING, 212,
00301     "Bad state: Robot must be in Running mode for this operation") \
00302 ENUM_ITEM(AUBO_BADSTATE_ROBOT_NOT_POWERED_OFF, 213,
00303     "Bad state: Robot must be in PowerOff mode for this operation") \
00304 ENUM_ITEM(AUBO_BADSTATE_SERIAL_OPEN_FAILED, 214, "Bad state: Failed to open serial device") \
00305 ENUM_ITEM(AUBO_BADSTATE_SERIAL_NOT_A_TERMINAL, 215,
00306     "Bad state: Specified device is not a terminal") \
00307 ENUM_ITEM(AUBO_BADSTATE_SERIAL_CONFIG_FAILED, 216,
00308     "Bad state: Failed to configure serial port parameters") \
00309 ENUM_ITEM(AUBO_BADSTATE_KINEMATICS_COMPENSATE_FAILED, 217,
00310     "Bad state: Failed to set kinematics compensation parameters") \
00311 ENUM_ITEM(AUBO_BADSTATE_ROBOT_NOT_STEADY, 218, "Bad state: Robot is not in steady state") \
00312 ENUM_ITEM(AUBO_BADSTATE_FREE_DRIVE_ACTIVE, 219, "Bad state: Free-drive mode is active") \
00313 ENUM_ITEM(AUBO_BADSTATE_FORCE_CTRL_ACTIVE, 220, "Bad state: Force control mode is active") \
00314 ENUM_ITEM(AUBO_BADSTATE_SIMULATION_MODE_ACTIVE, 221,
00315     "Bad state: Operation not allowed in simulation mode") \
00316 ENUM_ITEM(AUBO_BADSTATE_MB_ERROR, 222, "Bad state: Modbus signal has error") \
00317 ENUM_ITEM(AUBO_BADSTATE_MB_TIMEOUT, 223, "Bad state: Modbus signal timeout") \
00318 ENUM_ITEM(AUBO_ERR_UNKOWN, 99999, "Unkown error occurred.")
00319
00320 /**
00321  * The RuntimeState enum
00322  *
00323  */
00324 #define ENUM_RuntimeState_DECLARES
00325     ENUM_ITEM(Running, 0, " ") \
00326     ENUM_ITEM(Retracting, 1, " ") \
00327     ENUM_ITEM(Pausing, 2, " ") \
00328     ENUM_ITEM(Paused, 3, " ") \
00329     ENUM_ITEM(Stepping, 4, " ") \
00330     ENUM_ITEM(Stopping, 5, " ( )") \
00331     ENUM_ITEM(Stopped, 6, " ") \
00332     ENUM_ITEM(Aborting, 7, " ( )")
00333
00334 /**
00335  * \chinese
00336  * @brief The RobotModeType enum
00337  *
00338  *
00339  * \endchinese
00340  *
00341  * \english

```

```

00342 * @brief The RobotModeType enum
00343 *
00344 * Hardware related
00345 * \endenglish
00346 */
00347 #define ENUM_RobotModeType_DECLARES \
00348     ENUM_ITEM(NoController, -1, " , aubo_control NoController") \
00349     ENUM_ITEM(Disconnected, 0, " ( EtherCAT )" \
00350     ENUM_ITEM(ConfirmSafety, 1, " , )" \
00351     ENUM_ITEM(Booting, 2, " , )" \
00352     ENUM_ITEM(PowerOff, 3, " , )" \
00353     ENUM_ITEM(PowerOn, 4, " , ( , )" \
00354     ENUM_ITEM(Idle, 5, " , ( , )" \
00355     ENUM_ITEM(BrakeReleasing, 6, " , )" \
00356     ENUM_ITEM(BackDrive, 7, " , )" \
00357     ENUM_ITEM(Running, 8, " , , )" \
00358     ENUM_ITEM(Maintaince, 9, " : , )" \
00359     ENUM_ITEM(Error, 10, " , )" \
00360     ENUM_ITEM(PowerOffing, 11, " , )"
00361
00362 #define ENUM_SafetyModeType_DECLARES \
00363     ENUM_ITEM(Undefined, 0, " ") \
00364     ENUM_ITEM(Normal, 1, " ") \
00365     ENUM_ITEM(ReducedMode, 2, " ") \
00366     ENUM_ITEM(Recovery, 3, " , recovery )" \
00367     ENUM_ITEM(Violation, 4, " , )" \
00368     ENUM_ITEM(ProtectiveStop, 5, " , )" \
00369     ENUM_ITEM(SafeguardStop, 6, "IO , )" \
00370     ENUM_ITEM(SystemEmergencyStop, 7, " ( , )" \
00371     ENUM_ITEM(RobotEmergencyStop, 8, " , )" \
00372     ENUM_ITEM(Fault, 9, " ")
00373 //ValidateJointId
00374
00375 /**
00376 * \chinese
00377 * ISO 10218-1:2011(E) 5.7
00378 * Automatic: In automatic mode, the robot shall execute the task programme and
00379 * the safeguarding measures shall be functioning. Automatic operation shall be
00380 * prevented if any stop condition is detected. Switching from this mode shall
00381 * result in a stop.
00382 * \endchinese
00383 * \english
00384 * Based on ISO 10218-1:2011(E) Section 5.7
00385 * Automatic: In automatic mode, the robot shall execute the task programme and
00386 * the safeguarding measures shall be functioning. Automatic operation shall be
00387 * prevented if any stop condition is detected. Switching from this mode shall
00388 * result in a stop.
00389 * \endenglish
00390 */
00391 #define ENUM_OperationalModeType_DECLARES \
00392     ENUM_ITEM(Disabled, 0, " : Operational Mode") \
00393     ENUM_ITEM(Automatic, 1, " : , (auto mode: robot normal operation, speed will not be limited)" \
00394     ENUM_ITEM(Manual, 2, " : (T1), (T2) (manual mode: robot programming teaching mode (T1), \
00395     robot running speed will be limited or robot program verification mode (T2)" \
00396
00397 /**
00398 * \-chinese , \-english Robot control mode, the final control object
00399 */
00399 #define ENUM_RobotControlModeType_DECLARES \
00400     ENUM_ITEM(Unknown, 0, " (unknown control mode)" \
00401     ENUM_ITEM(Position, 1, " movej (position control)" \
00402     ENUM_ITEM(Speed, 2, " speedj/speedl (speed control)" \
00403     ENUM_ITEM(Servo, 3, " servoj (position control)" \
00404     ENUM_ITEM(Freedrive, 4, " freedrive_mode" \
00405     ENUM_ITEM(Force, 5, " force_mode" \
00406     ENUM_ITEM(Torque, 6, " (joint torque control)" \
00407     ENUM_ITEM(Collision, 7, " (collision mode)" \
00408
00409 #define ENUM_JointServoModeType_DECLARES \
00410     ENUM_ITEM(Unknown, -1, " ") \
00411     ENUM_ITEM(Open, 0, " (open loop mode)" \
00412     ENUM_ITEM(Current, 1, " (current servo mode)" \
00413     ENUM_ITEM(Velocity, 2, " (speed servo mode)" \
00414     ENUM_ITEM(Position, 3, " (position servo mode)" \
00415     ENUM_ITEM(Torque, 4, " (torque servo mode)" \
00416
00417 #define ENUM_JointStateType_DECLARES \
00418     ENUM_ITEM(Poweroff, 0, " (node not conected to interface board or already powered off)" \
00419     ENUM_ITEM(Idle, 2, " (node idle)" \
00420     ENUM_ITEM(Fault, 3, " , (node error, node stopped servo move, brake engaged)" \
00421     ENUM_ITEM(Running, 4, " (node servo)" \
00422     ENUM_ITEM(Bootload, 5, " bootloader , (node bootloader state, pause all communication)" \
00423
00424 #define ENUM_StandardInputAction_DECLARES \
00425     ENUM_ITEM(Default, 0, " ") \
00426     ENUM_ITEM(Handguide, 1, " ") \
00427     ENUM_ITEM(GoHome, 2, " ")

```

```

00428 ENUM_ITEM(StartProgram, 3, " ") \
00429 ENUM_ITEM(StopProgram, 4, " ") \
00430 ENUM_ITEM(PauseProgram, 5, " ") \
00431 ENUM_ITEM(PopupDismiss, 6, " ") \
00432 ENUM_ITEM(PowerOn, 7, " / ") \
00433 ENUM_ITEM(PowerOff, 8, " / ") \
00434 ENUM_ITEM(ResumeProgram, 9, " ") \
00435 ENUM_ITEM(SlowDown1, 10, " 1 ") \
00436 ENUM_ITEM(SlowDown2, 11, " 2 ") \
00437 ENUM_ITEM(SafeStop, 12, " ") \
00438 ENUM_ITEM(RunningGuard, 13, " ") \
00439 ENUM_ITEM(MoveToFirstPoint, 14, " ") \
00440 ENUM_ITEM(xSlowDown1, 15, " 1 ") \
00441 ENUM_ITEM(xSlowDown2, 16, " 2 ") \
00442 ENUM_ITEM(ConveyorTrack, 17, " ") \
00443 ENUM_ITEM(xConveyorTrack, 18, " ") \
00444 ENUM_ITEM(UnlockProtectiveStop, 19, " ") \
00445 ENUM_ITEM(ArbitraryResumeProgram, 20, " ") \
00446
00447 #define ENUM_StandardOutputRunState_DECLARES \
00448 ENUM_ITEM(None, 0, " ") \
00449 ENUM_ITEM(StopLow, 1, " ") \
00450 ENUM_ITEM(StopHigh, 2, " ") \
00451 ENUM_ITEM(RunningHigh, 3, " ") \
00452 ENUM_ITEM(PausedHigh, 4, " ") \
00453 ENUM_ITEM(AtHome, 5, " ") \
00454 ENUM_ITEM(Handguiding, 6, " ") \
00455 ENUM_ITEM(PowerOn, 7, " ") \
00456 ENUM_ITEM(RobotEmergencyStop, 8, " ") \
00457 ENUM_ITEM(SystemEmergencyStop, 9, " ") \
00458 ENUM_ITEM(InternalEmergencyStop, 8, " ") \
00459 ENUM_ITEM(ExternalEmergencyStop, 9, " ") \
00460 ENUM_ITEM(SystemError, 10, " ") \
00461 ENUM_ITEM(NotSystemError, 11, " ") \
00462 ENUM_ITEM(RobotOperable, 12, " ") \
00463 ENUM_ITEM(OperationalMode, 13, " ") \
00464 ENUM_ITEM(SafeguardStop, 14, " ") \
00465 ENUM_ITEM(ProtectiveStop, 15, " ") \
00466 ENUM_ITEM(ProgramAborted, 16, " ") \
00467
00468 #define ENUM_SafetyInputAction_DECLARES \
00469 ENUM_ITEM(Unassigned, 0, " ") \
00470 ENUM_ITEM(EmergencyStop, 1, " ") \
00471 ENUM_ITEM(SafeguardStop, 2, " ") \
00472 ENUM_ITEM(SafeguardReset, 3, " ") \
00473 ENUM_ITEM(ThreePositionSwitch, 4, "3 ") \
00474 ENUM_ITEM(OperationalMode, 5, " ") \
00475 ENUM_ITEM(HandGuide, 6, " ") \
00476 ENUM_ITEM(ReducedMode, 7, " 1 ( ) 0 ") \
00477 ENUM_ITEM(AutomaticModeSafeguardStop, 8, " ( ) ") \
00478 ENUM_ITEM(AutomaticModeSafeguardReset, 9, " ( ) ") \
00479
00480 #define ENUM_SafetyOutputRunState_DECLARES \
00481 ENUM_ITEM(Unassigned, 0, " ") \
00482 ENUM_ITEM(SystemEmergencyStop, 1, " ") \
00483 ENUM_ITEM(NotSystemEmergencyStop, 2, " ") \
00484 ENUM_ITEM(RobotMoving, 3, " 0.1rad/s") \
00485 ENUM_ITEM(RobotNotMoving, 4, " 0.1rad/s") \
00486 ENUM_ITEM(ReducedMode, 5, " ") \
00487 ENUM_ITEM(NotReducedMode, 6, " ") \
00488 ENUM_ITEM(SafeHome, 7, " Home ") \
00489 ENUM_ITEM(RobotNotStopping, 8, " ") \
00490
00491 #define ENUM_PayloadIdentifyMoveAxis_DECLARES \
00492 ENUM_ITEM(Joint_2_6, 0, "2 6 ") \
00493 ENUM_ITEM(Joint_3_6, 1, "3 6 ") \
00494 ENUM_ITEM(Joint_4_6, 2, "4 6 ") \
00495 ENUM_ITEM(Joint_4_5_6, 3, "4 5 6 ") \
00496
00497 #define ENUM_EnvelopingShape_DECLARES \
00498 ENUM_ITEM(Cube, 1, " ") \
00499 ENUM_ITEM(Column, 2, " ") \
00500 ENUM_ITEM(Stl, 3, " STL ") \
00501
00502 #define ENUM_TaskFrameType_DECLARES \
00503 ENUM_ITEM(NONE, 0, " ") \
00504 ENUM_ITEM(POINT_FORCE, 1, " , y TCP , x z " \
00505 " , y TCP , x z " \
00506 " TCP 10mm" \
00507 " X, X Y , X Y , " \
00508 " Z, Y-X Y-Z , ") \
00509 ENUM_ITEM(FRAME_FORCE, 2, " SIMPLE FORC") \
00510 ENUM_ITEM(MOTION_FORCE, 3, " , x TCP x-y y , x-y ") \
00511 ENUM_ITEM(TOOL_FORCE, 4, " ") \
00512
00513 #ifdef ERROR
00514 #undef ERROR

```

```

00515 #endif
00516
00517 #define ENUM_TraceLevel_DECLARES \
00518     ENUM_ITEM(FATAL, 0, " ") \
00519     ENUM_ITEM(ERROR, 1, " ") \
00520     ENUM_ITEM(WARNING, 2, " ") \
00521     ENUM_ITEM(INFO, 3, " ") \
00522     ENUM_ITEM(DEBUG, 4, " ")
00523
00524 #define ENUM_AxisModeType_DECLARES \
00525     ENUM_ITEM(NoController, -1, " ", aubo_control NoController) \
00526     ENUM_ITEM(Disconnected, 0, " ") \
00527     ENUM_ITEM(PowerOff, 1, " ") \
00528     ENUM_ITEM(BrakeReleasing, 2, " ") \
00529     ENUM_ITEM(Idle, 3, " ") \
00530     ENUM_ITEM(Running, 4, " ") \
00531     ENUM_ITEM(Fault, 5, " ")
00532
00533 #define ENUM_SafeguardedStopType_DECLARES \
00534     ENUM_ITEM(None, 0, " ") \
00535     ENUM_ITEM(SafeguardedStopIOInput, 1, " (IO )") \
00536     ENUM_ITEM(SafeguardedStop3PE, 2, " ( )") \
00537     ENUM_ITEM(SafeguardedStopOperational, 3, " ( )")
00538
00539 #define ENUM_RobotEmergencyStopType_DECLARES \
00540     ENUM_ITEM(RobotEmergencyStopNone, 0, " ") \
00541     ENUM_ITEM(RobotEmergencyStopControlBox, 1, " ( )") \
00542     ENUM_ITEM(RobotEmergencyStopTeachPendant, 2, " ( )") \
00543     ENUM_ITEM(RobotEmergencyStopHandle, 3, " ( )") \
00544     ENUM_ITEM(RobotEmergencyStopEI, 4, " (IO)")
00545
00546 #define ENUM_ITEM(c, n, ...) c = n,
00547 enum AuboErrorCodes : int
00548 {
00549     ENUM_AuboErrorCodes_DECLARES
00550 };
00551
00552 enum class RuntimeState : int
00553 {
00554     ENUM_RuntimeState_DECLARES
00555 };
00556
00557 enum class RobotModeType : int
00558 {
00559     ENUM_RobotModeType_DECLARES
00560 };
00561
00562 enum class AxisModeType : int
00563 {
00564     ENUM_AxisModeType_DECLARES
00565 };
00566
00567 /**
00568  * \~chinese : \~english Safety Mode
00569  */
00570
00571 enum class SafetyModeType : int
00572 {
00573     ENUM_SafetyModeType_DECLARES
00574 };
00575
00576 /**
00577  * \~chinese \~english Operational Mode
00578  */
00579 enum class OperationalModeType : int
00580 {
00581     ENUM_OperationalModeType_DECLARES
00582 };
00583
00584 /**
00585  * \~chinese \~english Robot Control Mode
00586  */
00587 enum class RobotControlModeType : int
00588 {
00589     ENUM_RobotControlModeType_DECLARES
00590 };
00591
00592 /**
00593  * \~chinese \~english Joint Servo Mode
00594  */
00595 enum class JointServoModeType : int
00596 {
00597     ENUM_JointServoModeType_DECLARES
00598 };
00599
00600 /**
00601  * \~chinese \~english Joint State

```

```

00602 */
00603 enum class JointStateType : int
00604 {
00605     ENUM_JointStateType_DECLARES
00606 };
00607
00608 /**
00609  * \~chinese      \~english Standard Output Run State
00610  */
00611 enum class StandardOutputRunState : int
00612 {
00613     ENUM_StandardOutputRunState_DECLARES
00614 };
00615
00616 /**
00617  * @brief The StandardInputAction enum
00618  */
00619 enum class StandardInputAction : int
00620 {
00621     ENUM_StandardInputAction_DECLARES
00622 };
00623
00624 enum class SafetyInputAction : int
00625 {
00626     ENUM_SafetyInputAction_DECLARES
00627 };
00628
00629 enum class SafetyOutputRunState : int
00630 {
00631     ENUM_SafetyOutputRunState_DECLARES
00632 };
00633
00634 enum TaskFrameType
00635 {
00636     ENUM_TaskFrameType_DECLARES
00637 };
00638
00639 enum EnvelopingShape : int
00640 {
00641     ENUM_EnvelopingShape_DECLARES
00642 };
00643
00644 enum PayloadIdentifyMoveAxis : int
00645 {
00646     ENUM_PayloadIdentifyMoveAxis_DECLARES
00647 };
00648
00649 enum TraceLevel
00650 {
00651     ENUM_TraceLevel_DECLARES
00652 };
00653
00654 enum SafeguedStopType : int
00655 {
00656     ENUM_SafeguedStopType_DECLARES
00657 };
00658
00659 enum RobotEmergencyStopType : int
00660 {
00661     ENUM_RobotEmergencyStopType_DECLARES
00662 };
00663 #undef ENUM_ITEM
00664
00665 #define DECL_TO_STRING_FUNC(ENUM)
00666     inline std::string toString(ENUM v)
00667     {
00668         using T = ENUM;
00669         std::string name = #ENUM "";
00670         ENUM_##ENUM##_DECLARES
00671
00672         return #ENUM "Unkown";
00673     }
00674     inline std::ostream &operator<<(std::ostream &os, ENUM v) \
00675     {
00676         os << toString(v);
00677         return os;
00678     }
00679
00680 #define ENUM_ITEM(c, n, ...) \
00681     if (v == T::c) { \
00682         return name + #c; \
00683     }
00684
00685 DECL_TO_STRING_FUNC(RuntimeState)
00686 DECL_TO_STRING_FUNC(RobotModeType)
00687 DECL_TO_STRING_FUNC(AxisModeType)
00688 DECL_TO_STRING_FUNC(SafetyModeType)

```



```

00689 DECL_TO_STRING_FUNC(OperationalModeType)
00690 DECL_TO_STRING_FUNC(RobotControlModeType)
00691 DECL_TO_STRING_FUNC(JointServoModeType)
00692 DECL_TO_STRING_FUNC(JointStateType)
00693 DECL_TO_STRING_FUNC(StandardInputAction)
00694 DECL_TO_STRING_FUNC(StandardOutputRunState)
00695 DECL_TO_STRING_FUNC(SafetyInputAction)
00696 DECL_TO_STRING_FUNC(SafetyOutputRunState)
00697 DECL_TO_STRING_FUNC(TaskFrameType)
00698 DECL_TO_STRING_FUNC(TraceLevel)
00699
00700 #undef ENUM_ITEM
00701
00702 enum class ForceControlState
00703 {
00704     Stopped,
00705     Starting,
00706     Stopping,
00707     Running
00708 };
00709
00710 enum class RefFrameType
00711 {
00712     None, ///<
00713     Tool, ///< \~chinese \~english Tool coordinate system
00714     Path, ///< \~chinese \~english Trajectory coordinate system
00715     Base, ///< \~chinese \~english Base coordinate system
00716 };
00717
00718 /// \~chinese \~english Circular motion parameters definition
00719 struct CircleParameters
00720 {
00721     std::vector<double> pose_via; ///< \~chinese \~english Pose of the intermediate point in circular motion
00722     std::vector<double> pose_to; ///< \~chinese \~english Pose of the end point in circular motion
00723     double a; ///< \~chinese , : m/s^2 \~english Acceleration, unit: m/s^2
00724     double v; ///< \~chinese : m/s \~english Speed, unit: m/s
00725     double blend_radius; ///< \~chinese , : m \~english Blending radius, unit: m
00726     double duration; ///< \~chinese : s \~english Running time, unit: s
00727     double helix;
00728     double spiral;
00729     double direction;
00730     int loop_times; ///< \~chinese \~english Currently not supported
00731 };
00732
00733 inline std::ostream &operator<<(std::ostream &os, CircleParameters p)
00734 {
00735     return os;
00736 }
00737
00738 struct SpiralParameters
00739 {
00740     std::vector<double> frame; ///< \~chinese \~english Reference point, the center point of the spiral and the
    reference coordinate system
00741     int plane; ///< \~chinese 0-XY 1-YZ 2-ZX \~english Reference plane selection 0-XY 1-YZ 2-ZX
00742     double angle; ///< \~chinese \~english The angle of rotation, if positive, the robot rotates counterclockwise
00743     double spiral; ///< \~chinese \~english Positive outward
00744     double helix; ///< \~chinese \~english Positive upward
00745 };
00746
00747 inline std::ostream &operator<<(std::ostream &os, SpiralParameters p)
00748 {
00749     return os;
00750 }
00751
00752 struct Enveloping
00753 {
00754     EnvelopingShape shape; ///< \~chinese \~english Enveloping shape
00755     std::vector<double> ep_args; ///< \~chinese shape None Stl ep_args ; \~english Enveloping combination, when
    shape is None or Stl, no need to assign value to ep_args.
00756     ///< \~chinese shape Cube ep_args 9 xmin,xmax,ymin,ymax,zmin,zmax,rx,ry,rz; \~english When shape is
    Cube, ep_args has 9 elements, which are xmin, xmax, ymin, ymax, zmin, zmax, rx, ry, rz;
00757     ///< \~chinese shape Column ep_args 5 radius,height,rx,ry,rz; \~english When shape is Column, ep_args has
    5 elements, which are radius, height, rx, ry, rz;
00758     std::string stl_path; ///< \~chinese stl ( ) stl , \~english Path of the stl file (absolute path), the stl file must be a
    binary file, \~chinese shape Stl \~english When shape is set to Stl, this item takes effect
00759 };
00760
00761 inline std::ostream &operator<<(std::ostream &os, Enveloping p)
00762 {
00763     return os;
00764 }
00765
00766 /// \~chinese \~english Trajectory configuration for
00767 /// payload identification
00768 struct TrajConfig
00769 {
00770     std::vector<Enveloping> envelopings; ///< \~chinese \~english Enveloping combination

```

```

00771 PayloadIdentifyMoveAxis move_axis; // \~chinese (ID), 0 \~english Axis of movement (ID), index starts from 0
00772 std::vector<double> init_joint; // \~chinese \~english Initial joint positions
00773 std::vector<double> upper_joint_bound; // \~chinese \~english Upper joint limits
00774 std::vector<double> lower_joint_bound; // \~chinese \~english Lower joint limits
00775 std::vector<double> max_velocity; // \~chinese 3.0 \~english Maximum joint velocities, default value is 3.0
00776 std::vector<double> max_acceleration; // \~chinese 5.0 \~english Maximum joint accelerations, default value
is 5.0
00777 };
00778
00779 inline std::ostream &operator<<(std::ostream &os, TrajConfig p)
00780 {
00781     return os;
00782 }
00783
00784 // result with error code
00785 using ResultWithErrno = std::tuple<std::vector<double>, int>;
00786 using ResultWithErrno1 = std::tuple<std::vector<std::vector<double>, int>;
00787 using ResultWithErrno2 = std::tuple<std::vector<std::string>, int>;
00788 using ResultWithErrno3 = std::tuple<int, int>;
00789
00790 // mass, cog, aom, inertia
00791 using Payload = std::tuple<double, std::vector<double>, std::vector<double>,
00792     std::vector<double>;
00793
00794 // force_offset, com, mass, angle
00795 using ForceSensorCalibResult =
00796     std::tuple<std::vector<double>, std::vector<double>, double,
00797     std::vector<double>;
00798
00799 // force_offset, com, mass, angle error
00800 using ForceSensorCalibResultWithError =
00801     std::tuple<std::vector<double>, std::vector<double>, double,
00802     std::vector<double>, double>;
00803
00804 // \~chinese m,d,k \~english Dynamics model m, d, k
00805 using DynamicsModel =
00806     std::tuple<std::vector<double>, std::vector<double>, std::vector<double>;
00807
00808 // double xmin;
00809 // double xmax;
00810 // double ymin;
00811 // double ymax;
00812 // double zmin;
00813 // double zmax;
00814 using Box = std::vector<double>;
00815
00816 // double xcbottom;
00817 // double ycbottom;
00818 // double zcbottom;
00819 // double height;
00820 // double radius;
00821 using Cylinder = std::vector<double>;
00822
00823 // double xc;
00824 // double yc;
00825 // double radius;
00826 using Sphere = std::vector<double>;
00827
00828 struct RobotMsg
00829 {
00830     uint64_t timestamp; ///< \~chinese \~english Timestamp,
00831         ///< i.e., system time
00832     TraceLevel level; ///< \~chinese \~english Log level
00833     int code; ///< \~chinese \~english Error code
00834     std::string
00835         source; ///< \~chinese alias \~english Alias of the
00836         ///< robot sending the message
00837         ///< \~chinese /root/arcs_ws/config/aubo_control.conf
00838         ///< \~english Can be found in
00839         ///< /root/arcs_ws/config/aubo_control.conf
00840         ///< \~chinese alias \~english The robot's
00841         ///< alias can be found in the configuration file
00842         ///< /root/arcs_ws/config/aubo_control.conf
00843     std::vector<std::string> args; ///< \~chinese \~english Robot parameters
00844 };
00845 using RobotMsgVector = std::vector<RobotMsg>;
00846
00847 struct GripperStatus
00848 {
00849     std::string name;
00850     bool is_connected;
00851     bool is_enabled;
00852     double position;
00853     double velocity;
00854     double force;
00855     double angle;
00856     double r_velocity;

```



```

00857     double torque;
00858     bool object_detection;
00859     bool motion_state;
00860     double voltage;
00861     double temperature;
00862     int status_code;
00863 };
00864 using GripperStatusVector = std::vector<GripperStatus>;
00865
00866 /// \~chinese RTDE \~english RTDE menu
00867 struct RtdeRecipe
00868 {
00869     bool to_server; ///< \~chinese / \~english Input/Output
00870     int chanel; ///< \~chinese \~english Channel
00871     double frequency; ///< \~chinese \~english Update frequency
00872     int trigger; ///< \~chinese ( ): 0 - ; 1 -
00873     ///< \~english Trigger method (this feature is not yet
00874     ///< implemented): 0 - Periodic; 1 - Change
00875     std::vector<std::string> segments; ///< \~chinese \~english segment list
00876 };
00877
00878 /// \~chinese \~english Error type
00879 enum error_type
00880 {
00881     parse_error = -32700, ///< \~chinese \~english Parse error
00882     invalid_request = -32600, ///< \~chinese \~english Invalid request
00883     method_not_found = -32601, ///< \~chinese \~english Method not found
00884     invalid_params = -32602, ///< \~chinese \~english Invalid parameters
00885     internal_error = -32603, ///< \~chinese \~english Internal error
00886     server_error = -32500, ///< \~chinese \~english Server error
00887     invalid = -32400 ///< \~chinese \~english Invalid
00888 };
00889
00890 /// \~chinese \~english Exception code
00891 enum ExceptionCode
00892 {
00893     EC_DISCONNECTED = -1, ///< \~chinese \~english Disconnected
00894     EC_NOT_LOGINED = -2, ///< \~chinese \~english Not logged in
00895     EC_INVALID_SOCKET = -3, ///< \~chinese \~english Invalid socket
00896     EC_REQUEST_BUSY = -4, ///< \~chinese \~english Request busy
00897     EC_SEND_FAILED = -5, ///< \~chinese \~english Send failed
00898     EC_RECV_TIMEOUT = -6, ///< \~chinese \~english Receive timeout
00899     EC_RECV_ERROR = -7, ///< \~chinese \~english Receive error
00900     EC_PARSE_ERROR = -8, ///< \~chinese \~english Parse error
00901     EC_INVALID_REQUEST = -9, ///< \~chinese \~english Invalid request
00902     EC_METHOD_NOT_FOUND = -10, ///< \~chinese \~english Method not found
00903     EC_INVALID_PARAMS = -11, ///< \~chinese \~english Invalid parameters
00904     EC_INTERNAL_ERROR = -12, ///< \~chinese \~english Internal error
00905     EC_SERVER_ERROR = -13, ///< \~chinese \~english Server error
00906     EC_INVALID = -14 ///< \~chinese \~english Invalid
00907 };
00908
00909 /// \~chinese AuboException \~english Custom exception class
00910 /// AuboException
00911 class AuboException : public std::exception
00912 {
00913 public:
00914     AuboException(int code, const std::string &prefix,
00915                  const std::string &message) noexcept
00916         : code_(code), message_(prefix + "\n" + message)
00917     {
00918     }
00919
00920     AuboException(int code, const std::string &message) noexcept
00921         : code_(code), message_(message)
00922     {
00923     }
00924
00925     error_type type() const
00926     {
00927         if (code_ >= -32603 && code_ <= -32600) {
00928             return static_cast<error_type>(code_);
00929         } else if (code_ >= -32099 && code_ <= -32000) {
00930             return server_error;
00931         } else if (code_ == -32700) {
00932             return parse_error;
00933         }
00934         return invalid;
00935     }
00936
00937     int code() const { return code_; }
00938     const char *what() const noexcept override { return message_.c_str(); }
00939
00940 private:
00941     int code_; ///< \~chinese \~english Exception code
00942     std::string message_; ///< \~chinese \~chinese Exception message
00943 };

```

```
00944
00945 inline const char *returnValue2Str(int retval)
00946 {
00947     static const char *retval_str[] = {
00948 #define ENUM_ITEM(n, v, s) s,
00949     ENUM_AuboErrorCodes_DECLARES
00950 #undef ENUM_ITEM
00951     };
00952
00953     enum arcs_index
00954     {
00955 #define ENUM_ITEM(n, v, s) n##_INDEX,
00956     ENUM_AuboErrorCodes_DECLARES
00957 #undef ENUM_ITEM
00958     };
00959
00960     int index = -1;
00961
00962 #define ENUM_ITEM(n, v, s) \
00963     if (abs(retval) == v) \
00964         index = n##_INDEX;
00965     ENUM_AuboErrorCodes_DECLARES
00966 #undef ENUM_ITEM
00967
00968     if (index == -1)
00969     {
00970         index = AUBO_ERR_UNKOWN_INDEX;
00971     }
00972
00973     return retval_str[(unsigned)index];
00974 }
00975
00976 } // namespace common_interface
00977 } // namespace arcs
00978 #endif
00979
00980 // clang-format on
00981
00982 #if defined ENABLE_JSON_TYPES
00983 #include "bindings/jsonrpc/json_types.h"
00984 #endif
```

Index

- `_D`
 - `error_stack.h`, [670](#), [671](#)
 - `_PH1_`
 - `error_stack.h`, [671](#)
 - `_PH2_`
 - `error_stack.h`, [671](#)
 - `_PH3_`
 - `error_stack.h`, [671](#)
 - `_PH4_`
 - `error_stack.h`, [672](#)
 - `~AuboApi`
 - `arcs::common_interface::AuboApi`, [570](#)
 - `~AxisInterface`
 - `arcs::common_interface::AxisInterface`, [575](#)
 - `~ForceControl`
 - `arcs::common_interface::ForceControl`, [582](#)
 - `~GripperInterface`
 - `arcs::common_interface::GripperInterface`, [584](#)
 - `~IoControl`
 - `arcs::common_interface::IoControl`, [593](#)
 - `~Math`
 - `arcs::common_interface::Math`, [595](#)
 - `~MotionControl`
 - `arcs::common_interface::MotionControl`, [601](#)
 - `~RegisterControl`
 - `arcs::common_interface::RegisterControl`, [605](#)
 - `~RobotAlgorithm`
 - `arcs::common_interface::RobotAlgorithm`, [609](#)
 - `~RobotConfig`
 - `arcs::common_interface::RobotConfig`, [614](#)
 - `~RobotInterface`
 - `arcs::common_interface::RobotInterface`, [616](#)
 - `~RobotManage`
 - `arcs::common_interface::RobotManage`, [618](#)
 - `~RobotState`
 - `arcs::common_interface::RobotState`, [634](#)
 - `~RuntimeMachine`
 - `arcs::common_interface::RuntimeMachine`, [640](#)
 - `~Serial`
 - `arcs::common_interface::Serial`, [641](#)
 - `~Socket`
 - `arcs::common_interface::Socket`, [643](#)
 - `~SyncMove`
 - `arcs::common_interface::SyncMove`, [647](#)
 - `~SystemInfo`
 - `arcs::common_interface::SystemInfo`, [648](#)
 - `~Trace`
 - `arcs::common_interface::Trace`, [649](#)
- a
 - `arcs::common_interface::CircleParameters`, [577](#)
 - abort
 - `RuntimeMachine ()`, [482](#)
 - addInt32RegEncoder
 - `RegisterControl ()`, [54](#)
 - addModbusEncoder
 - `RegisterControl ()`, [54](#)
 - alarm
 - `Trace ()`, [541](#)
 - allow_manual_high_speed
 - `arcs::common_interface::RobotSafetyParameterRange`, [623](#)
 - angle
 - `arcs::common_interface::GripperStatus`, [588](#)
 - `arcs::common_interface::SpiralParameters`, [644](#)
 - arbitraryResume
 - `RuntimeMachine ()`, [482](#)
 - arcs, [549](#)
 - `arcs::common_interface`, [549](#)
 - `AuboApiPtr`, [552](#)
 - `AuboErrorCodes`, [557](#)
 - `AxisInterfacePtr`, [552](#)
 - `AxisModeType`, [557](#)
 - `Base`, [561](#)
 - `Box`, [552](#)
 - `Cylinder`, [552](#)
 - `DynamicsModel`, [552](#)
 - `EC_DISCONNECTED`, [558](#)
 - `EC_INTERNAL_ERROR`, [559](#)
 - `EC_INVALID_SOCKET`, [558](#)
 - `EC_INVALID`, [559](#)
 - `EC_INVALID_PARAMS`, [559](#)
 - `EC_INVALID_REQUEST`, [559](#)
 - `EC_METHOD_NOT_FOUND`, [559](#)
 - `EC_NOT_LOGINED`, [558](#)
 - `EC_PARSE_ERROR`, [559](#)
 - `EC_RECV_ERROR`, [558](#)
 - `EC_RECV_TIMEOUT`, [558](#)
 - `EC_REQUEST_BUSY`, [558](#)
 - `EC_SEND_FAILED`, [558](#)
 - `EC_SERVER_ERROR`, [559](#)
 - `ENUM_AuboErrorCodes_DECLARES`, [557](#)
 - `ENUM_AxisModeType_DECLARES`, [557](#)
 - `ENUM_EnvelopingShape_DECLARES`, [558](#)
 - `ENUM_JointServoModeType_DECLARES`, [559](#)

- ENUM_JointStateType_DECLARES, [560](#)
- ENUM_OperationalModeType_DECLARES, [560](#)
- ENUM_PayloadIdentifyMoveAxis_DECLARES, [561](#)
- ENUM_RobotControlModeType_DECLARES, [561](#)
- ENUM_RobotEmergencyStopType_DECLARES, [562](#)
- ENUM_RobotModeType_DECLARES, [562](#)
- ENUM_RuntimeState_DECLARES, [562](#)
- ENUM_SafeguardedStopType_DECLARES, [562](#)
- ENUM_SafetyInputAction_DECLARES, [563](#)
- ENUM_SafetyModeType_DECLARES, [563](#)
- ENUM_SafetyOutputRunState_DECLARES, [563](#)
- ENUM_StandardInputAction_DECLARES, [563](#)
- ENUM_StandardOutputRunState_DECLARES, [564](#)
- ENUM_TaskFrameType_DECLARES, [564](#)
- ENUM_TraceLevel_DECLARES, [564](#)
- EnvelopingShape, [557](#)
- error_type, [558](#)
- ExceptionCode, [558](#)
- ForceControlPtr, [552](#)
- ForceControlState, [559](#)
- ForceSensorCalibResult, [553](#)
- ForceSensorCalibResultWithError, [553](#)
- GripperInterfacePtr, [553](#)
- GripperStatusVector, [553](#)
- internal_error, [558](#)
- invalid, [558](#)
- invalid_params, [558](#)
- invalid_request, [558](#)
- IoControlPtr, [553](#)
- JointServoModeType, [559](#)
- JointStateType, [559](#)
- MathPtr, [553](#)
- method_not_found, [558](#)
- MotionControlPtr, [553](#)
- None, [561](#)
- OperationalModeType, [560](#)
- operator<<, [564](#), [565](#)
- parse_error, [558](#)
- Path, [561](#)
- PathBuffer_CubicSpline, [560](#)
- PathBuffer_JointBSpline, [560](#)
- PathBuffer_JointBSplineC, [560](#)
- PathBuffer_JointSpline, [560](#)
- PathBuffer_JointSplineC, [560](#)
- PathBuffer_TOPPPRA, [560](#)
- PathBufferType, [560](#)
- Payload, [554](#)
- PayloadIdentifyMoveAxis, [560](#)
- RefFrameType, [561](#)
- RegisterControlPtr, [554](#)
- ResultWithErrno, [554](#)
- ResultWithErrno1, [554](#)
- ResultWithErrno2, [554](#)
- ResultWithErrno3, [554](#)
- returnValue2Str, [565](#)
- RobotAlgorithmPtr, [554](#)
- RobotConfigPtr, [555](#)
- RobotControlModeType, [561](#)
- RobotEmergencyStopType, [561](#)
- RobotInterfacePtr, [555](#)
- RobotManagePtr, [555](#)
- RobotModeType, [562](#)
- RobotMsgVector, [555](#)
- RobotStatePtr, [555](#)
- Running, [559](#)
- RuntimeMachinePtr, [555](#)
- RuntimeState, [562](#)
- SafeguardedStopType, [562](#)
- SafetyInputAction, [562](#)
- SafetyModeType, [563](#)
- SafetyOutputRunState, [563](#)
- SerialPtr, [555](#)
- server_error, [558](#)
- SocketPtr, [555](#)
- Sphere, [556](#)
- StandardInputAction, [563](#)
- StandardOutputRunState, [563](#)
- Starting, [559](#)
- Stopped, [559](#)
- Stopping, [559](#)
- SyncMovePtr, [556](#)
- SystemInfoPtr, [556](#)
- TaskFrameType, [564](#)
- TaskSet, [556](#)
- Tool, [561](#)
- TraceLevel, [564](#)
- TracePtr, [556](#)
- Vector3d, [556](#)
- Vector3f, [556](#)
- Vector4d, [556](#)
- Vector4f, [557](#)
- Vector6f, [557](#)
- arcs::common_interface::AuboApi, [569](#)
 - ~AuboApi, [570](#)
 - AuboApi, [570](#)
 - d_, [570](#)
- arcs::common_interface::AuboException, [571](#)
 - AuboException, [572](#)
 - code, [573](#)
 - code_, [573](#)
 - message_, [573](#)
 - type, [573](#)
 - what, [573](#)
- arcs::common_interface::AxisInterface, [574](#)
 - ~AxisInterface, [575](#)
 - AxisInterface, [575](#)
 - d_, [576](#)
- arcs::common_interface::CircleParameters, [576](#)
 - a, [577](#)

- blend_radius, 577
- direction, 577
- duration, 577
- helix, 577
- loop_times, 577
- pose_to, 578
- pose_via, 578
- spiral, 578
- v, 578
- arcs::common_interface::Enveloping, 579
 - ep_args, 579
 - shape, 579
 - stl_path, 579
- arcs::common_interface::ForceControl, 580
 - ~ForceControl, 582
 - d_, 583
 - ForceControl, 582
- arcs::common_interface::GripperInterface, 583
 - ~GripperInterface, 584
 - d_, 586
 - gripperGetAngle, 584
 - gripperGetForce, 584
 - gripperGetMotionState, 584
 - gripperGetMountPose, 584
 - gripperGetObjectDetection, 585
 - gripperGetPosition, 585
 - gripperGetRVelocity, 585
 - gripperGetSoftwareVersion, 585
 - gripperGetTemperature, 585
 - gripperGetTorque, 585
 - gripperGetVelocity, 585
 - gripperGetVoltage, 585
 - GripperInterface, 584
 - gripperSetAngle, 586
 - gripperSetForce, 586
 - gripperSetRVelocity, 586
 - gripperSetTorque, 586
 - gripperSetVelocity, 586
- arcs::common_interface::GripperStatus, 587
 - angle, 588
 - force, 588
 - is_connected, 588
 - is_enabled, 588
 - motion_state, 588
 - name, 588
 - object_detection, 588
 - position, 588
 - r_velocity, 589
 - status_code, 589
 - temperature, 589
 - torque, 589
 - velocity, 589
 - voltage, 589
- arcs::common_interface::IoControl, 590
 - ~IoControl, 593
 - d_, 593
 - IoControl, 593
- arcs::common_interface::Math, 594
 - ~Math, 595
 - d_, 595
 - Math, 595
- arcs::common_interface::MotionControl, 595
 - ~MotionControl, 601
 - d_, 602
 - getConveyorTrackNextItem, 601
 - MotionControl, 601
 - servoCartesianWithAxisGroup, 601
 - servoJointWithAxisGroup, 602
- arcs::common_interface::RegisterControl, 602
 - ~RegisterControl, 605
 - d_, 607
 - modbusSetOutputSignalWithTimeout, 606
 - RegisterControl, 605
- arcs::common_interface::RobotAlgorithm, 607
 - ~RobotAlgorithm, 609
 - d_, 610
 - pathMovej, 609
 - RobotAlgorithm, 609
- arcs::common_interface::RobotConfig, 610
 - ~RobotConfig, 614
 - d_, 615
 - disableAxisGroup, 614
 - enableAxisGroup, 614
 - getRobotBaseParent, 614
 - RobotConfig, 614
- arcs::common_interface::RobotInterface, 615
 - ~RobotInterface, 616
 - d_, 616
 - RobotInterface, 616
- arcs::common_interface::RobotManage, 616
 - ~RobotManage, 618
 - d_, 618
 - RobotManage, 618
- arcs::common_interface::RobotMsg, 619
 - args, 620
 - code, 620
 - level, 620
 - source, 620
 - timestamp, 620
- arcs::common_interface::RobotSafetyParameterRange, 621
 - allow_manual_high_speed, 623
 - crc32, 623
 - cubic, 623
 - elbow_force, 623
 - elbow_speed, 624
 - joint_torque, 624
 - momentum, 624
 - orig, 624
 - params, 624
 - plane, 624
 - planes, 624
 - power, 625
 - qddmax, 625
 - qmax, 625
 - qmin, 625

- reduced_entry_distance, 625
- reduced_entry_time, 625
- restrict_elbow, 626
- RobotSafetyParameterRange, 623
- safety_home, 626
- safety_input_auto_safeguard_reset, 626
- safety_input_auto_safeguard_stop, 626
- safety_input_emergency_stop, 626
- safety_input_handguide, 626
- safety_input_operational_mode, 627
- safety_input_reduced_mode, 627
- safety_input_safeguard_reset, 627
- safety_input_safeguard_stop, 627
- safety_input_three_position_switch, 627
- safety_output_emergency_stop, 627
- safety_output_not_emergency_stop, 628
- safety_output_reduced_mode, 628
- safety_output_robot_moving, 628
- safety_output_robot_not_stopping, 628
- safety_output_robot_steady, 628
- safety_output_safe_home, 629
- safety_output_safeguard_stop, 629
- size, 629
- stop_distance, 629
- stop_time, 629
- tcp_force, 629
- tcp_speed, 630
- tool_azimuth, 630
- tool_deviation, 630
- tool_inclination, 630
- tool_orientation, 630
- tools, 630
- tp_3pe_for_handguide, 631
- trigger_planes, 631
- arcs::common_interface::RobotState, 631
 - ~RobotState, 634
 - d_, 635
 - getBaseForceSensor, 635
 - RobotState, 634
- arcs::common_interface::RtdeRecipe, 636
 - channel, 637
 - frequency, 637
 - segments, 637
 - to_server, 637
 - trigger, 637
- arcs::common_interface::RuntimeMachine, 637
 - ~RuntimeMachine, 640
 - d_, 640
 - getExecutionStatus1, 640
 - getRuntimeState, 640
 - RuntimeMachine, 640
- arcs::common_interface::Serial, 640
 - ~Serial, 641
 - d_, 642
 - Serial, 641
- arcs::common_interface::Socket, 642
 - ~Socket, 643
- d_, 643
- Socket, 643
- arcs::common_interface::SpiralParameters, 643
 - angle, 644
 - frame, 644
 - helix, 644
 - plane, 644
 - spiral, 644
- arcs::common_interface::SyncMove, 645
 - ~SyncMove, 647
 - d_, 647
 - SyncMove, 647
- arcs::common_interface::SystemInfo, 647
 - ~SystemInfo, 648
 - d_, 648
 - SystemInfo, 648
- arcs::common_interface::Trace, 648
 - ~Trace, 649
 - d_, 649
 - notify, 649
 - Trace, 649
- arcs::common_interface::TrajConfig, 650
 - envelopings, 651
 - init_joint, 651
 - lower_joint_bound, 651
 - max_acceleration, 651
 - max_velocity, 651
 - move_axis, 651
 - upper_joint_bound, 651
- arcs::common_interface::VibrationRecalibrationParameter, 652
 - cog, 653
 - inertia, 653
 - mass, 653
 - points, 653
 - stiff_param, 653
 - stiff_section, 653
- arcs::common_interface::WObjectData, 654
 - attach_frame, 654
 - obj_coord, 654
 - remote_tool, 655
 - user_coord, 655
- arcs::error_stack, 566
 - ARCS_ERROR_CODES, 566
 - codeCompose, 566
 - dump, 566
 - errorCode2Str, 566
 - ErrorCodes, 566
 - mod, 567
 - str2ErrorCode, 567
- ARCS_ERROR_CODES
 - arcs::error_stack, 566
 - error_stack.h, 672
- args
 - arcs::common_interface::RobotMsg, 620
- attach_frame
 - arcs::common_interface::WObjectData, 654
- attachRobotBaseTo

- RobotConfig (), 350
- attachWeldingGun
 - RobotConfig (), 350
- Aubo SDK, 1
- AUBO_SDK_HAL_ERROR_H
 - error_stack.h, 672
- AUBO_SDK_RTM_ERROR_H
 - error_stack.h, 672
- AUBO_SDK_SYSTEM_ERROR_H
 - error_stack.h, 672
- AuboApi
 - arcs::common_interface::AuboApi, 570
- AuboApi (), 17
 - getAxisInterface, 18
 - getAxisNames, 18
 - getGripperInterface, 18
 - getMath, 18
 - getRegisterControl, 19
 - getRobotInterface, 19
 - getRobotNames, 20
 - getRuntimeMachine, 20
 - getSerial, 21
 - getSocket, 21
 - getSyncMove, 21
 - getSystemInfo, 22
 - getTrace, 22
- AuboApiPtr
 - arcs::common_interface, 552
- AuboErrorCodes
 - arcs::common_interface, 557
- AuboException
 - arcs::common_interface::AuboException, 572
- axisGroupAdd
 - SyncMove (), 523
- axisGroupAddAxis
 - SyncMove (), 523
- axisGroupDelete
 - SyncMove (), 524
- axisGroupGetActualPositions
 - SyncMove (), 524
- axisGroupGetAxisIndex
 - SyncMove (), 525
- axisGroupGetAxisName
 - SyncMove (), 525
- axisGroupGetTargetPositions
 - SyncMove (), 525
- axisGroupMoveJoint
 - SyncMove (), 526
- axisGroupOffsetPositions
 - SyncMove (), 526
- axisGroupSpeedJoint
 - SyncMove (), 527
- axisGroupUpdateAxis
 - SyncMove (), 527
- AxisInterface
 - arcs::common_interface::AxisInterface, 575
- AxisInterface (), 22
 - clearAxisError, 24
 - enableExtAxis, 24
 - followAnotherAxis, 24
 - getAxisModeType, 25
 - getErrorCode, 25
 - getExtAxisAcceleration, 25
 - getExtAxisBusCurrent, 25
 - getExtAxisBusVoltage, 25
 - getExtAxisCurrent, 26
 - getExtAxisMaxAcceleration, 26
 - getExtAxisMaxPosition, 26
 - getExtAxisMaxVelocity, 26
 - getExtAxisMountingPose, 26
 - getExtAxisPose, 27
 - getExtAxisPosition, 27
 - getExtAxisTemperature, 27
 - getExtAxisType, 27
 - getExtAxisVelocity, 27
 - getExtMinPosition, 28
 - moveExtJoint, 28
 - poweroffExtAxis, 28
 - poweronExtAxis, 29
 - setExtAxisMountingPose, 29
 - speedExtJoint, 29
 - stopExtJoint, 30
 - stopFollowAnotherAxis, 30
- AxisInterfacePtr
 - arcs::common_interface, 552
- AxisModeType
 - arcs::common_interface, 557
- backdrive
 - RobotManage (), 406
- Base
 - arcs::common_interface, 561
- blend_radius
 - arcs::common_interface::CircleParameters, 577
- Box
 - arcs::common_interface, 552
- calcJacobian
 - RobotAlgorithm (), 321
- calcSafetyParametersChecksum
 - RobotConfig (), 350
- calculateCircleFourthPoint
 - Math (), 35
- calibrateCoordinate
 - Math (), 36
- calibrateTcpForceSensor
 - RobotAlgorithm (), 322
- calibrateTcpForceSensor2
 - RobotAlgorithm (), 323
- calibrateTcpForceSensor3
 - RobotAlgorithm (), 324
- calibVibrationParams
 - RobotAlgorithm (), 324
- calibVibrationParams1
 - RobotAlgorithm (), 324
- calibWorkpieceCoordinatePara

- RobotAlgorithm (), 325
- CARTESIAN_DOF
 - type_def.h, 1111
- chanel
 - arcs::common_interface::RtdeRecipe, 637
- changePoseWithXYRef
 - Math (), 37
- clearAxisError
 - AxisInterface (), 24
- clearBreakPoints
 - RuntimeMachine (), 482
- clearNamedVariable
 - RegisterControl (), 55
- clearPath
 - MotionControl (), 225
- clearPreloadPrograms
 - RuntimeMachine (), 483
- code
 - arcs::common_interface::AuboException, 573
 - arcs::common_interface::RobotMsg, 620
- code_
 - arcs::common_interface::AuboException, 573
- codeCompose
 - arcs::error_stack, 566
- cog
 - arcs::common_interface::VibrationRecalibrationParameters, 653
- confirmSafetyParameters
 - RobotConfig (), 350
- conveyorTrackCircle
 - MotionControl (), 225
- conveyorTrackClearItems
 - MotionControl (), 226
- conveyorTrackCreatItem
 - MotionControl (), 226
- conveyorTrackLine
 - MotionControl (), 227
- conveyorTrackStop
 - MotionControl (), 228
- conveyorTrackSwitch
 - MotionControl (), 229
- crc32
 - arcs::common_interface::RobotSafetyParameterRange, 623
- cubic
 - arcs::common_interface::RobotSafetyParameterRange, 623
- Cylinder
 - arcs::common_interface, 552
- d_
 - arcs::common_interface::AuboApi, 570
 - arcs::common_interface::AxisInterface, 576
 - arcs::common_interface::ForceControl, 583
 - arcs::common_interface::GripperInterface, 586
 - arcs::common_interface::IoControl, 593
 - arcs::common_interface::Math, 595
 - arcs::common_interface::MotionControl, 602
 - arcs::common_interface::RegisterControl, 607
 - arcs::common_interface::RobotAlgorithm, 610
 - arcs::common_interface::RobotConfig, 615
 - arcs::common_interface::RobotInterface, 616
 - arcs::common_interface::RobotManage, 618
 - arcs::common_interface::RobotState, 635
 - arcs::common_interface::RuntimeMachine, 640
 - arcs::common_interface::Serial, 642
 - arcs::common_interface::Socket, 643
 - arcs::common_interface::SyncMove, 647
 - arcs::common_interface::SystemInfo, 648
 - arcs::common_interface::Trace, 649
- DECL_TO_STRING_FUNC
 - type_def.h, 1111
- deleteTask
 - RuntimeMachine (), 483
- deleteVirtualEncoder
 - RegisterControl (), 55
- deltaPoseAdd
 - Math (), 37
- deltaPoseTrans
 - Math (), 38
- detachTask
 - RuntimeMachine (), 483
- direction
 - arcs::common_interface::CircleParameters, 577
- disableAxisGroup
 - arcs::common_interface::RobotConfig, 614
- disableJointSoftServo
 - MotionControl (), 230
- disbaleVibrationSuppress
 - MotionControl (), 231
- doc Directory Reference, 546
- doc/SDK_overview.md, 657
- dump
 - arcs::error_stack, 566
- duration
 - arcs::common_interface::CircleParameters, 577
- DynamicsModel
 - arcs::common_interface, 552
- EC_DISCONNECTED
 - arcs::common_interface, 558
- EC_INTERNAL_ERROR
 - arcs::common_interface, 559
- EC_INVALID_SOCKET
 - arcs::common_interface, 558
- EC_INVALID
 - arcs::common_interface, 559
- EC_INVALID_PARAMS
 - arcs::common_interface, 559
- EC_INVALID_REQUEST
 - arcs::common_interface, 559
- EC_METHOD_NOT_FOUND
 - arcs::common_interface, 559
- EC_NOT_LOGINED
 - arcs::common_interface, 558

- EC_PARSE_ERROR
 - arcs::common_interface, [559](#)
- EC_RECV_ERROR
 - arcs::common_interface, [558](#)
- EC_RECV_TIMEOUT
 - arcs::common_interface, [558](#)
- EC_REQUEST_BUSY
 - arcs::common_interface, [558](#)
- EC_SEND_FAILED
 - arcs::common_interface, [558](#)
- EC_SERVER_ERROR
 - arcs::common_interface, [559](#)
- elbow_force
 - arcs::common_interface::RobotSafetyParameterRange, [623](#)
- elbow_speed
 - arcs::common_interface::RobotSafetyParameterRange, [624](#)
- enableAxisGroup
 - arcs::common_interface::RobotConfig, [614](#)
- enableEndCollisionCheck
 - RobotConfig (), [351](#)
- enableExtAxis
 - AxisInterface (), [24](#)
- enableJointSoftServo
 - MotionControl (), [231](#)
- enableVibrationSuppress
 - MotionControl (), [232](#)
- enterCritical
 - RuntimeMachine (), [484](#)
- ENUM_AuboErrorCodes_DECLARES
 - arcs::common_interface, [557](#)
 - type_def.h, [1111](#)
- ENUM_AxisModeType_DECLARES
 - arcs::common_interface, [557](#)
 - type_def.h, [1111](#)
- ENUM_EnvelopingShape_DECLARES
 - arcs::common_interface, [558](#)
 - type_def.h, [1111](#)
- ENUM_ITEM
 - type_def.h, [1112](#), [1113](#)
- ENUM_JointServoModeType_DECLARES
 - arcs::common_interface, [559](#)
 - type_def.h, [1113](#)
- ENUM_JointStateType_DECLARES
 - arcs::common_interface, [560](#)
 - type_def.h, [1113](#)
- ENUM_OperationalModeType_DECLARES
 - arcs::common_interface, [560](#)
 - type_def.h, [1113](#)
- ENUM_PayloadIdentifyMoveAxis_DECLARES
 - arcs::common_interface, [561](#)
 - type_def.h, [1114](#)
- ENUM_RobotControlModeType_DECLARES
 - arcs::common_interface, [561](#)
 - type_def.h, [1114](#)
- ENUM_RobotEmergencyStopType_DECLARES
 - arcs::common_interface, [562](#)
- ENUM_RobotModeType_DECLARES
 - arcs::common_interface, [562](#)
 - type_def.h, [1114](#)
- ENUM_RuntimeState_DECLARES
 - arcs::common_interface, [562](#)
 - type_def.h, [1115](#)
- ENUM_SafeguardedStopType_DECLARES
 - arcs::common_interface, [562](#)
 - type_def.h, [1115](#)
- ENUM_SafetyInputAction_DECLARES
 - arcs::common_interface, [563](#)
 - type_def.h, [1115](#)
- ENUM_SafetyModeType_DECLARES
 - arcs::common_interface, [563](#)
 - type_def.h, [1116](#)
- ENUM_SafetyOutputRunState_DECLARES
 - arcs::common_interface, [563](#)
 - type_def.h, [1116](#)
- ENUM_StandardInputAction_DECLARES
 - arcs::common_interface, [563](#)
 - type_def.h, [1116](#)
- ENUM_StandardOutputRunState_DECLARES
 - arcs::common_interface, [564](#)
 - type_def.h, [1117](#)
- ENUM_TaskFrameType_DECLARES
 - arcs::common_interface, [564](#)
 - type_def.h, [1117](#)
- ENUM_TraceLevel_DECLARES
 - arcs::common_interface, [564](#)
 - type_def.h, [1117](#)
- envelopings
 - arcs::common_interface::TrajConfig, [651](#)
- EnvelopingShape
 - arcs::common_interface, [557](#)
- ep_args
 - arcs::common_interface::Enveloping, [579](#)
- error_stack.h
 - _D, [670](#), [671](#)
 - _PH1_, [671](#)
 - _PH2_, [671](#)
 - _PH3_, [671](#)
 - _PH4_, [672](#)
 - ARCS_ERROR_CODES, [672](#)
 - AUBO_SDK_HAL_ERROR_H, [672](#)
 - AUBO_SDK_RTM_ERROR_H, [672](#)
 - AUBO_SDK_SYSTEM_ERROR_H, [672](#)
 - HAL_ERRORS, [672](#)
 - HARDWARE_INTERFACE_ERRORS, [673](#)
 - JOINT_ERRORS, [673](#)
 - PEDSTRAL_ERRORS, [673](#)
 - RTM_ERRORS, [673](#)
 - SAFETY_INTERFACE_BOARD_ERRORS, [673](#)
 - SYSTEM_ERRORS, [673](#)
 - TOOL_ERRORS, [674](#)
- error_type
 - arcs::common_interface, [558](#)

- errorCode2Str
 - arcs::error_stack, [566](#)
- ErrorCodes
 - arcs::error_stack, [566](#)
- ExceptionCode
 - arcs::common_interface, [558](#)
- exitCritical
 - RuntimeMachine (), [484](#)
- exitHandguideMode
 - RobotManage (), [407](#)
- fcCalDynamicModel
 - ForceControl(), [110](#)
- fcDisable
 - ForceControl(), [111](#)
- fcEnable
 - ForceControl(), [111](#)
- fcSetSensorLimits
 - ForceControl(), [112](#)
- fcSetSensorThresholds
 - ForceControl(), [113](#)
- firmwareUpdate
 - RobotConfig (), [351](#)
- followAnotherAxis
 - AxisInterface (), [24](#)
- followJoint
 - MotionControl (), [232](#)
- followLine
 - MotionControl (), [232](#)
- force
 - arcs::common_interface::GripperStatus, [588](#)
- ForceControl
 - arcs::common_interface::ForceControl, [582](#)
- ForceControl(), [107](#)
 - fcCalDynamicModel, [110](#)
 - fcDisable, [111](#)
 - fcEnable, [111](#)
 - fcSetSensorLimits, [112](#)
 - fcSetSensorThresholds, [113](#)
 - getActualJointPositionsHistory, [114](#)
 - getDynamicModel, [114](#)
 - getFcSensorLimits, [115](#)
 - getFcSensorThresholds, [115](#)
 - isCondFullfiled, [116](#)
 - isFcEnabled, [117](#)
 - isSoftFloatEnabled, [117](#)
 - resetDamping, [118](#)
 - resetLpFilter, [118](#)
 - setCondActive, [119](#)
 - setCondAdvanced, [120](#)
 - setCondCylinder, [121](#)
 - setCondDistance, [122](#)
 - setCondForce, [122](#)
 - setCondOrient, [123](#)
 - setCondPlane, [124](#)
 - setCondSphere, [125](#)
 - setCondTcpSpeed, [126](#)
 - setDamping, [127](#)
 - setDynamicModel, [128](#)
 - setDynamicModel1, [129](#)
 - setDynamicModelContact, [130](#)
 - setDynamicModelInsert, [131](#)
 - setDynamicModelSearch, [132](#)
 - setLpFilter, [132](#)
 - setSoftFloatParams, [133](#)
 - setSupvForce, [134](#)
 - setSupvOrient, [135](#)
 - setSupvPosBox, [136](#)
 - setSupvPosCylinder, [136](#)
 - setSupvPosSphere, [137](#)
 - setSupvReoriSpeed, [138](#)
 - setSupvTcpSpeed, [139](#)
 - setTargetForce, [140](#)
 - softFloatDisable, [141](#)
 - softFloatEnable, [141](#)
 - speedChangeDisable, [142](#)
 - speedChangeEnable, [142](#)
 - speedChangeTune, [143](#)
 - toolContact, [144](#)
- ForceControlPtr
 - arcs::common_interface, [552](#)
- ForceControlState
 - arcs::common_interface, [559](#)
- ForceSensorCalibResult
 - arcs::common_interface, [553](#)
- ForceSensorCalibResultWithError
 - arcs::common_interface, [553](#)
- forceTrans
 - Math (), [38](#)
- forwardDynamics
 - RobotAlgorithm (), [325](#)
- forwardDynamics1
 - RobotAlgorithm (), [326](#)
- forwardKinematics
 - RobotAlgorithm (), [327](#)
- forwardKinematics1
 - RobotAlgorithm (), [328](#)
- forwardKinematicsAll
 - RobotAlgorithm (), [329](#)
- forwardToolKinematics
 - RobotAlgorithm (), [330](#)
- frame
 - arcs::common_interface::SpiralParameters, [644](#)
- frameAdd
 - SyncMove (), [528](#)
- frameAttach
 - SyncMove (), [528](#)
- frameConvertPose
 - SyncMove (), [529](#)
- frameDelete
 - SyncMove (), [529](#)
- frameDeleteAll
 - SyncMove (), [530](#)
- frameExist
 - SyncMove (), [530](#)
- frameGetChildren

- SyncMove (), 530
- frameGetParent
 - SyncMove (), 531
- frameGetPose
 - SyncMove (), 531
- frameMove
 - SyncMove (), 532
- freedrive
 - RobotManage (), 408
- frequency
 - arcs::common_interface::RtdeRecipe, 637
- frictionModelIdentify
 - RobotAlgorithm (), 331
- generateDiagnoseFile
 - RobotManage (), 408
- generatePayloadIdentifyTraj
 - RobotAlgorithm (), 332
- getActualJointPositionsHistory
 - ForceControl(), 114
- getActualTcpOffset
 - RobotState (), 435
- getAdvancePlanContext
 - RuntimeMachine (), 485
- getAdvancePtr
 - RuntimeMachine (), 485
- getAxisInterface
 - AuboApi (), 18
- getAxisModeType
 - AxisInterface (), 25
- getAxisNames
 - AuboApi (), 18
- getBaseForce
 - RobotState (), 435
- getBaseForceOffset
 - RobotConfig (), 352
- getBaseForceSensor
 - arcs::common_interface::RobotState, 635
- getBaseForceSensorNames
 - RobotConfig (), 353
- getBool
 - RegisterControl (), 56
- getBoolInput
 - RegisterControl (), 56
- getBoolOutput
 - RegisterControl (), 57
- getCollisionLevel
 - RobotConfig (), 354
- getCollisionStopType
 - RobotConfig (), 354
- getCollisionThreshold
 - RobotConfig (), 355
- getConfigurableDigitalInput
 - IoControl (IO), 149
- getConfigurableDigitalInputAction
 - IoControl (IO), 149
- getConfigurableDigitalInputNum
 - IoControl (IO), 150
- getConfigurableDigitalInputs
 - IoControl (IO), 151
- getConfigurableDigitalOutput
 - IoControl (IO), 152
- getConfigurableDigitalOutputNum
 - IoControl (IO), 153
- getConfigurableDigitalOutputRunstate
 - IoControl (IO), 153
- getConfigurableDigitalOutputs
 - IoControl (IO), 154
- getControlBoxHumidity
 - RobotState (), 435
- getControlBoxTemperature
 - RobotState (), 436
- getControlBoxType
 - RobotConfig (), 356
- getControlSoftwareBuildDate
 - SystemInfo (), 537
- getControlSoftwareFullVersion
 - SystemInfo (), 537
- getControlSoftwareVersionCode
 - SystemInfo (), 538
- getControlSoftwareVersionHash
 - SystemInfo (), 538
- getControlSystemTime
 - SystemInfo (), 539
- getConveyorTrackNextItem
 - arcs::common_interface::MotionControl, 601
- getConveyorTrackQueue
 - MotionControl (), 233
- getCycletime
 - RobotConfig (), 357
- getDefaultJointAcc
 - RobotConfig (), 357
- getDefaultJointSpeed
 - RobotConfig (), 358
- getDefaultToolAcc
 - RobotConfig (), 359
- getDefaultToolSpeed
 - RobotConfig (), 360
- getDeltaPoseBySensorDistance
 - Math (), 39
- getDof
 - RobotConfig (), 360
- getDouble
 - RegisterControl (), 58
- getDoubleInput
 - RegisterControl (), 59
- getDoubleOutput
 - RegisterControl (), 60
- getDuration
 - MotionControl (), 234
- getDynamicModel
 - ForceControl(), 114
- getElbowPosistion
 - RobotState (), 437
- getElbowVelocity
 - RobotState (), 438
- getEncDecoderType

- IoControl (IO), 155
- getEncTickCount
 - IoControl (IO), 156
- getEqradius
 - MotionControl (), 235
- getErrorCode
 - AxisInterface (), 25
- getExecId
 - MotionControl (), 235
- getExecutionStatus
 - RuntimeMachine (), 486
- getExecutionStatus1
 - arcs::common_interface::RuntimeMachine, 640
- getExtAxisAcceleration
 - AxisInterface (), 25
- getExtAxisBusCurrent
 - AxisInterface (), 25
- getExtAxisBusVoltage
 - AxisInterface (), 25
- getExtAxisCurrent
 - AxisInterface (), 26
- getExtAxisMaxAcceleration
 - AxisInterface (), 26
- getExtAxisMaxPosition
 - AxisInterface (), 26
- getExtAxisMaxVelocity
 - AxisInterface (), 26
- getExtAxisMountingPose
 - AxisInterface (), 26
- getExtAxisPose
 - AxisInterface (), 27
- getExtAxisPosition
 - AxisInterface (), 27
- getExtAxisTemperature
 - AxisInterface (), 27
- getExtAxisType
 - AxisInterface (), 27
- getExtAxisVelocity
 - AxisInterface (), 27
- getExtMinPosition
 - AxisInterface (), 28
- getFcSensorLimits
 - ForceControl(), 115
- getFcSensorThresholds
 - ForceControl(), 115
- getFirmwareUpdateProcess
 - RobotConfig (), 361
- getFloat
 - RegisterControl (), 60
- getFloatInput
 - RegisterControl (), 61
- getFloatOutput
 - RegisterControl (), 62
- getForceControl
 - RobotInterface(), 104
- getFreedriveDamp
 - RobotConfig (), 362
- getFuturePathPointsJoint
 - MotionControl (), 236
- getGravity
 - RobotConfig (), 363
- getGripperInterface
 - AuboApi (), 18
- getHandguidDamp
 - RobotConfig (), 363
- getHandguideFeature
 - RobotManage (), 409
- getHandguideFreeAxes
 - RobotManage (), 410
- getHandguideStatus
 - RobotManage (), 410
- getHandguideTrigger
 - RobotManage (), 410
- getHandleIoStatus
 - IoControl (IO), 156
- getHandleType
 - IoControl (IO), 157
- getHardwareCustomParameters
 - RobotConfig (), 364
- getHomePosition
 - RobotConfig (), 365
- getInt16Register
 - RegisterControl (), 63
- getInt16RegisterBit
 - RegisterControl (), 64
- getInt32
 - RegisterControl (), 65
- getInt32Input
 - RegisterControl (), 65
- getInt32Output
 - RegisterControl (), 66
- getInterfaceVersionCode
 - SystemInfo (), 539
- getInterpPtr
 - RuntimeMachine (), 486
- getIoControl
 - RobotInterface(), 104
- getJointAccelerations
 - RobotState (), 438
- getJointContactTorques
 - RobotState (), 439
- getJointCurrents
 - RobotState (), 440
- getJointFirmwareVersions
 - RobotState (), 441
- getJointGravityTorques
 - RobotState (), 441
- getJointHardwareVersions
 - RobotState (), 442
- getJointMaxAccelerations
 - RobotConfig (), 365
- getJointMaxPositions
 - RobotConfig (), 366
- getJointMaxSpeeds
 - RobotConfig (), 367

- getJointMinPositions
 - RobotConfig (), [367](#)
- getJointPositions
 - RobotState (), [443](#)
- getJointPositionsHistory
 - RobotState (), [444](#)
- getJointServoMode
 - RobotState (), [444](#)
- getJointSpeeds
 - RobotState (), [445](#)
- getJointState
 - RobotState (), [446](#)
- getJointTargetAccelerations
 - RobotState (), [446](#)
- getJointTargetCurrents
 - RobotState (), [447](#)
- getJointTargetPositions
 - RobotState (), [448](#)
- getJointTargetSpeeds
 - RobotState (), [449](#)
- getJointTargetTorques
 - RobotState (), [449](#)
- getJointTemperatures
 - RobotState (), [450](#)
- getJointTorqueSensors
 - RobotState (), [451](#)
- getJointUniqueIds
 - RobotState (), [452](#)
- getJointVoltages
 - RobotState (), [452](#)
- getKinematicsCompensate
 - RobotConfig (), [368](#)
- getKinematicsParam
 - RobotConfig (), [369](#)
- getLimitJointMaxAccelerations
 - RobotConfig (), [370](#)
- getLimitJointMaxPositions
 - RobotConfig (), [371](#)
- getLimitJointMaxSpeeds
 - RobotConfig (), [372](#)
- getLimitJointMinPositions
 - RobotConfig (), [372](#)
- getLimitTcpMaxSpeed
 - RobotConfig (), [373](#)
- getLookAheadSize
 - MotionControl (), [237](#)
- getMainCurrent
 - RobotState (), [453](#)
- getMainPtr
 - RuntimeMachine (), [487](#)
- getMainVoltage
 - RobotState (), [454](#)
- getMasterBoardFirmwareVersion
 - RobotState (), [455](#)
- getMasterBoardHardwareVersion
 - RobotState (), [455](#)
- getMasterBoardUniqueId
 - RobotState (), [456](#)
- getMath
 - AuboApi (), [18](#)
- getModbusDeviceStatus
 - RegisterControl (), [67](#)
- getMotionControl
 - RobotInterface(), [105](#)
- getMotionLeftTime
 - MotionControl (), [237](#)
- getMountingPose
 - RobotConfig (), [374](#)
- getName
 - RobotConfig (), [375](#)
- getNamedVariableType
 - RegisterControl (), [68](#)
- getOperationalMode
 - RobotManage (), [411](#)
- getPauseJointPositions
 - MotionControl (), [238](#)
- getPayload
 - RobotConfig (), [375](#)
- getPayloadIdentifyResult
 - RobotAlgorithm (), [333](#)
- getPedestalFirmwareVersion
 - RobotState (), [457](#)
- getPedestalHardwareVersion
 - RobotState (), [458](#)
- getPedestalUniqueId
 - RobotState (), [458](#)
- getPlanContext
 - RuntimeMachine (), [487](#)
- getPreloadProgram
 - RuntimeMachine (), [488](#)
- getProgress
 - MotionControl (), [239](#)
- getQueueSize
 - MotionControl (), [240](#)
- getRecordCache
 - RobotManage (), [411](#)
- getRegisterControl
 - AuboApi (), [19](#)
- getResumeMode
 - MotionControl (), [240](#)
- getRobotAlgorithm
 - RobotInterface(), [105](#)
- getRobotBaseParent
 - arcs::common_interface::RobotConfig, [614](#)
- getRobotConfig
 - RobotInterface(), [105](#)
- getRobotConfiguration
 - RobotAlgorithm (), [333](#)
- getRobotControlMode
 - RobotManage (), [412](#)
- getRobotCurrent
 - RobotState (), [459](#)
- getRobotEmergencyStopSource
 - RobotConfig (), [376](#)
- getRobotInterface
 - AuboApi (), [19](#)

getRobotManage
 RobotInterface(), 106
 getRobotModeType
 RobotState (), 460
 getRobotNames
 AuboApi (), 20
 getRobotState
 RobotInterface(), 106
 getRobotSubType
 RobotConfig (), 376
 getRobotType
 RobotConfig (), 377
 getRobotVoltage
 RobotState (), 461
 getRuntimeMachine
 AuboApi (), 20
 getRuntimeState
 arcs::common_interface::RuntimeMachine,
 640
 getSafeguardStopSource
 RobotConfig (), 378
 getSafeguardStopType
 RobotConfig (), 378
 getSafetyModeType
 RobotState (), 461
 getSafetyParametersChecksum
 RobotConfig (), 379
 getSelectedTcpForceSensorName
 RobotConfig (), 379
 getSerial
 AuboApi (), 21
 getServoModeSelect
 MotionControl (), 241
 getSlaveBoardFirmwareVersion
 RobotState (), 462
 getSlaveBoardHardwareVersion
 RobotState (), 463
 getSlaveBoardUniqueId
 RobotState (), 464
 getSlowDownFraction
 RobotConfig (), 380
 getSlowDownLevel
 RobotState (), 464
 getSocket
 AuboApi (), 21
 getSpeedFraction
 MotionControl (), 241
 getStandardAnalogInput
 IoControl (IO), 158
 getStandardAnalogInputDomain
 IoControl (IO), 158
 getStandardAnalogInputNum
 IoControl (IO), 159
 getStandardAnalogOutput
 IoControl (IO), 160
 getStandardAnalogOutputDomain
 IoControl (IO), 161
 getStandardAnalogOutputNum
 IoControl (IO), 162
 getStandardAnalogOutputRunstate
 IoControl (IO), 163
 getStandardDigitalInput
 IoControl (IO), 164
 getStandardDigitalInputAction
 IoControl (IO), 165
 getStandardDigitalInputNum
 IoControl (IO), 165
 getStandardDigitalInputs
 IoControl (IO), 166
 getStandardDigitalOutput
 IoControl (IO), 167
 getStandardDigitalOutputNum
 IoControl (IO), 168
 getStandardDigitalOutputRunstate
 IoControl (IO), 169
 getStandardDigitalOutputs
 IoControl (IO), 170
 getStaticLinkInputNum
 IoControl (IO), 170
 getStaticLinkInputs
 IoControl (IO), 171
 getStaticLinkOutputNum
 IoControl (IO), 172
 getStaticLinkOutputs
 IoControl (IO), 173
 getStatus
 RuntimeMachine (), 488
 getString
 RegisterControl (), 68
 getSyncMove
 AuboApi (), 21
 RobotInterface(), 106
 getSystemInfo
 AuboApi (), 22
 getTargetTcpPose
 RobotState (), 465
 getTaskQueueSize
 RuntimeMachine (), 489
 getTcpForce
 RobotState (), 466
 getTcpForceOffset
 RobotConfig (), 381
 getTcpForceSensorNames
 RobotConfig (), 381
 getTcpForceSensorPose
 RobotConfig (), 382
 getTcpForceSensors
 RobotState (), 466
 getTcpForceSensorStatus
 RobotState (), 467
 getTcpMaxAccelerations
 RobotConfig (), 382
 getTcpMaxLinearVelocity
 MotionControl (), 242
 getTcpMaxSpeeds
 RobotConfig (), 383

- getTcpOffset
 - RobotConfig (), 384
- getTcpPose
 - RobotState (), 468
- getTcpSpeed
 - RobotState (), 468
- getTcpTargetForce
 - RobotState (), 469
- getTcpTargetPose
 - RobotState (), 470
- getTcpTargetSpeed
 - RobotState (), 471
- getTimer
 - RuntimeMachine (), 489
- getToolAnalogInput
 - IoControl (IO), 173
- getToolAnalogInputDomain
 - IoControl (IO), 174
- getToolAnalogInputNum
 - IoControl (IO), 175
- getToolAnalogOutput
 - IoControl (IO), 176
- getToolAnalogOutputDomain
 - IoControl (IO), 177
- getToolAnalogOutputNum
 - IoControl (IO), 178
- getToolAnalogOutputRunstate
 - IoControl (IO), 179
- getToolButtonStatus
 - IoControl (IO), 179
- getToolCommMode
 - RobotState (), 471
- getToolDigitalInput
 - IoControl (IO), 180
- getToolDigitalInputAction
 - IoControl (IO), 181
- getToolDigitalInputNum
 - IoControl (IO), 182
- getToolDigitalInputs
 - IoControl (IO), 183
- getToolDigitalOutput
 - IoControl (IO), 183
- getToolDigitalOutputNum
 - IoControl (IO), 184
- getToolDigitalOutputRunstate
 - IoControl (IO), 185
- getToolDigitalOutputs
 - IoControl (IO), 186
- getToolFirmwareVersion
 - RobotState (), 472
- getToolHardwareVersion
 - RobotState (), 473
- getToolPose
 - RobotState (), 474
- getToolUniqueId
 - RobotState (), 474
- getToolVoltageOutputDomain
 - IoControl (IO), 187
- getTrace
 - AuboApi (), 22
 - RobotInterface(), 107
- getTrajectoryQueueSize
 - MotionControl (), 243
- getTriggInterrupts
 - RuntimeMachine (), 490
- getVecChar
 - RegisterControl (), 69
- getVecDouble
 - RegisterControl (), 70
- getVecFloat
 - RegisterControl (), 71
- getVecInt32
 - RegisterControl (), 72
- getWatchDogAction
 - RegisterControl (), 72
- getWatchDogTimeout
 - RegisterControl (), 73
- getWorkObjectHold
 - MotionControl (), 244
- gotoLine
 - RuntimeMachine (), 490
- gripperAdd
 - GripperInterface (), 31
- gripperConnect
 - GripperInterface (), 31
- gripperDelete
 - GripperInterface (), 31
- gripperDisconnect
 - GripperInterface (), 32
- gripperEnable
 - GripperInterface (), 32
- gripperGetAngle
 - arcs::common_interface::GripperInterface, 584
- gripperGetForce
 - arcs::common_interface::GripperInterface, 584
- gripperGetHardwareVersion
 - GripperInterface (), 32
- gripperGetMotionState
 - arcs::common_interface::GripperInterface, 584
- gripperGetMountPose
 - arcs::common_interface::GripperInterface, 584
- gripperGetNames
 - GripperInterface (), 32
- gripperGetObjectDetection
 - arcs::common_interface::GripperInterface, 585
- gripperGetPosition
 - arcs::common_interface::GripperInterface, 585
- gripperGetRVelocity
 - arcs::common_interface::GripperInterface, 585
- gripperGetSoftwareVersion
 - arcs::common_interface::GripperInterface, 585
- gripperGetStatusCode
 - GripperInterface (), 32
- gripperGetSupportedModels
 - GripperInterface (), 32
- gripperGetTemperature

- arcs::common_interface::GripperInterface, 585
- gripperGetTorque
 - arcs::common_interface::GripperInterface, 585
- gripperGetVelocity
 - arcs::common_interface::GripperInterface, 585
- gripperGetVoltage
 - arcs::common_interface::GripperInterface, 585
- gripperGetWorkMode
 - GripperInterface (), 32
- GripperInterface
 - arcs::common_interface::GripperInterface, 584
- GripperInterface (), 31
 - gripperAdd, 31
 - gripperConnect, 31
 - gripperDelete, 31
 - gripperDisconnect, 32
 - gripperEnable, 32
 - gripperGetHardwareVersion, 32
 - gripperGetNames, 32
 - gripperGetStatusCode, 32
 - gripperGetSupportedModels, 32
 - gripperGetWorkMode, 32
 - gripperIsConnected, 32
 - gripperIsEnabled, 32
 - gripperMove, 33
 - gripperRename, 33
 - gripperResetSlaveId, 33
 - gripperScanDevices, 33
 - gripperSetMountPose, 33
 - gripperSetPosition, 33
 - gripperSetWorkMode, 33
 - gripperStop, 33
- GripperInterfacePtr
 - arcs::common_interface, 553
- gripperIsConnected
 - GripperInterface (), 32
- gripperIsEnabled
 - GripperInterface (), 32
- gripperMove
 - GripperInterface (), 33
- gripperRename
 - GripperInterface (), 33
- gripperResetSlaveId
 - GripperInterface (), 33
- gripperScanDevices
 - GripperInterface (), 33
- gripperSetAngle
 - arcs::common_interface::GripperInterface, 586
- gripperSetForce
 - arcs::common_interface::GripperInterface, 586
- gripperSetMountPose
 - GripperInterface (), 33
- gripperSetPosition
 - GripperInterface (), 33
- gripperSetRVelocity
 - arcs::common_interface::GripperInterface, 586
- gripperSetTorque
 - arcs::common_interface::GripperInterface, 586
- gripperSetVelocity
 - arcs::common_interface::GripperInterface, 586
- gripperSetWorkMode
 - GripperInterface (), 33
- GripperStatusVector
 - arcs::common_interface, 553
- gripperStop
 - GripperInterface (), 33
- hal_error.h
 - HAL_ERRORS, 676
 - HARDWARE_INTERFACE_ERRORS, 676
 - JOINT_ERRORS, 676
 - PEDSTRAL_ERRORS, 677
 - SAFETY_INTERFACE_BOARD_ERRORS, 677
 - TOOL_ERRORS, 677
- HAL_ERRORS
 - error_stack.h, 672
 - hal_error.h, 676
- handguideMode
 - RobotManage (), 413
- HARDWARE_INTERFACE_ERRORS
 - error_stack.h, 673
 - hal_error.h, 676
- hasBaseForceSensor
 - RobotConfig (), 384
- hasEncoderSensor
 - IoControl (IO), 187
- hasItemOnConveyorToTrack
 - MotionControl (), 244
- hasNamedVariable
 - RegisterControl (), 73
- hasTcpForceSensor
 - RobotConfig (), 385
- helix
 - arcs::common_interface::CircleParameters, 577
 - arcs::common_interface::SpiralParameters, 644
- homMatrixToPose
 - Math (), 39
- include Directory Reference, 547
- include/aubo Directory Reference, 545
- include/aubo/aubo_api.h, 657, 658
- include/aubo/axis_interface.h, 662, 663
- include/aubo/error_stack Directory Reference, 546
- include/aubo/error_stack/error_stack.h, 668, 674
- include/aubo/error_stack/hal_error.h, 675, 678
- include/aubo/error_stack/rtm_error.h, 681, 682
- include/aubo/error_stack/system_error.h, 685, 686
- include/aubo/gripper_interface.h, 687, 688
- include/aubo/math.h, 692, 693
- include/aubo/register_control.h, 706, 708
- include/aubo/robot Directory Reference, 547
- include/aubo/robot/force_control.h, 747, 748
- include/aubo/robot/io_control.h, 779, 780

- include/aubo/robot/motion_control.h, 837, 839
- include/aubo/robot/robot_algorithm.h, 915, 917
- include/aubo/robot/robot_config.h, 938, 939
- include/aubo/robot/robot_manage.h, 983, 985
- include/aubo/robot/robot_state.h, 1007, 1008
- include/aubo/robot_interface.h, 1044, 1045
- include/aubo/runtime_machine.h, 1049, 1051
- include/aubo/serial.h, 1069, 1071
- include/aubo/socket.h, 1075, 1077
- include/aubo/sync_move.h, 1084, 1086
- include/aubo/system_info.h, 1097, 1099
- include/aubo/trace.h, 1103, 1104
- include/aubo/type_def.h, 1107, 1119
- inertia
 - arcs::common_interface::VibrationRecalibrationParameters, 653
- init_joint
 - arcs::common_interface::TrajConfig, 651
- internal_error
 - arcs::common_interface, 558
- interpolatePose
 - Math (), 40
- invalid
 - arcs::common_interface, 558
- invalid_params
 - arcs::common_interface, 558
- invalid_request
 - arcs::common_interface, 558
- inverseKinematics
 - RobotAlgorithm (), 335
- inverseKinematics1
 - RobotAlgorithm (), 336
- inverseKinematicsAll
 - RobotAlgorithm (), 337
- inverseKinematicsAll1
 - RobotAlgorithm (), 338
- inverseToolKinematics
 - RobotAlgorithm (), 339
- inverseToolKinematicsAll
 - RobotAlgorithm (), 340
- IoControl
 - arcs::common_interface::IoControl, 593
- IoControl (IO), 144
 - getConfigurableDigitalInput, 149
 - getConfigurableDigitalInputAction, 149
 - getConfigurableDigitalInputNum, 150
 - getConfigurableDigitalInputs, 151
 - getConfigurableDigitalOutput, 152
 - getConfigurableDigitalOutputNum, 153
 - getConfigurableDigitalOutputRunstate, 153
 - getConfigurableDigitalOutputs, 154
 - getEncDecoderType, 155
 - getEncTickCount, 156
 - getHandleIoStatus, 156
 - getHandleType, 157
 - getStandardAnalogInput, 158
 - getStandardAnalogInputDomain, 158
 - getStandardAnalogInputNum, 159
 - getStandardAnalogOutput, 160
 - getStandardAnalogOutputDomain, 161
 - getStandardAnalogOutputNum, 162
 - getStandardAnalogOutputRunstate, 163
 - getStandardDigitalInput, 164
 - getStandardDigitalInputAction, 165
 - getStandardDigitalInputNum, 165
 - getStandardDigitalInputs, 166
 - getStandardDigitalOutput, 167
 - getStandardDigitalOutputNum, 168
 - getStandardDigitalOutputRunstate, 169
 - getStandardDigitalOutputs, 170
 - getStaticLinkInputNum, 170
 - getStaticLinkInputs, 171
 - getStaticLinkOutputNum, 172
 - getStaticLinkOutputs, 173
 - getToolAnalogInput, 173
 - getToolAnalogInputDomain, 174
 - getToolAnalogInputNum, 175
 - getToolAnalogOutput, 176
 - getToolAnalogOutputDomain, 177
 - getToolAnalogOutputNum, 178
 - getToolAnalogOutputRunstate, 179
 - getToolButtonStatus, 179
 - getToolDigitalInput, 180
 - getToolDigitalInputAction, 181
 - getToolDigitalInputNum, 182
 - getToolDigitalInputs, 183
 - getToolDigitalOutput, 183
 - getToolDigitalOutputNum, 184
 - getToolDigitalOutputRunstate, 185
 - getToolDigitalOutputs, 186
 - getToolVoltageOutputDomain, 187
 - hasEncoderSensor, 187
 - isToolIoInput, 188
 - setAnalogOutputAfterEStopDefault, 189
 - setConfigurableDigitalInputAction, 190
 - setConfigurableDigitalOutput, 191
 - setConfigurableDigitalOutputAfterEStop, 192
 - setConfigurableDigitalOutputPulse, 193
 - setConfigurableDigitalOutputRunstate, 194
 - setDigitalInputActionDefault, 195
 - setDigitalOutputAfterEStopDefault, 196
 - setDigitalOutputRunstateDefault, 197
 - setEncDecoderType, 198
 - setEncTickCount, 198
 - setStandardAnalogInputDomain, 199
 - setStandardAnalogOutput, 200
 - setStandardAnalogOutputAfterEStop, 200
 - setStandardAnalogOutputDomain, 201
 - setStandardAnalogOutputRunstate, 202
 - setStandardDigitalInputAction, 203
 - setStandardDigitalOutput, 204
 - setStandardDigitalOutputAfterEStop, 205
 - setStandardDigitalOutputPulse, 206
 - setStandardDigitalOutputRunstate, 207
 - setToolAnalogInputDomain, 208
 - setToolAnalogOutput, 209

- setToolAnalogOutputDomain, [210](#)
- setToolAnalogOutputRunstate, [211](#)
- setToolDigitalInputAction, [212](#)
- setToolDigitalOutput, [213](#)
- setToolDigitalOutputPulse, [214](#)
- setToolDigitalOutputRunstate, [215](#)
- setToolIoInput, [216](#)
- setToolVoltageOutputDomain, [217](#)
- unwindEncDeltaTickCount, [218](#)
- IoControlPtr
 - arcs::common_interface, [553](#)
- is_connected
 - arcs::common_interface::GripperStatus, [588](#)
- is_enabled
 - arcs::common_interface::GripperStatus, [588](#)
- isBackdriveEnabled
 - RobotManage (), [414](#)
- isBlending
 - MotionControl (), [245](#)
- isCollisionOccurred
 - RobotState (), [475](#)
- isCondFullfiled
 - ForceControl(), [116](#)
- isConveyorTrackExceed
 - MotionControl (), [246](#)
- isConveyorTrackSync
 - MotionControl (), [247](#)
- isEndCollisionCheckEnabled
 - RobotConfig (), [386](#)
- isFcEnabled
 - ForceControl(), [117](#)
- isFreedriveEnabled
 - RobotManage (), [414](#)
- isHandguideEnabled
 - RobotManage (), [415](#)
- isJointSoftServoEnabled
 - MotionControl (), [247](#)
- isLinkModeEnabled
 - RobotManage (), [416](#)
- isPowerOn
 - RobotState (), [476](#)
- isServoModeEnabled
 - MotionControl (), [248](#)
- isSimulationEnabled
 - RobotManage (), [417](#)
- isSoftFloatEnabled
 - ForceControl(), [117](#)
- isSpeedFractionCritical
 - MotionControl (), [249](#)
- isSteady
 - RobotState (), [476](#)
- isSupportedTimeOptimal
 - MotionControl (), [249](#)
- isSyncMoveOn
 - SyncMove (), [532](#)
- isTaskAlive
 - RuntimeMachine (), [491](#)
- isTeachPendantEnabled
 - RobotState (), [477](#)
- isTimeOptimalEnabled
 - MotionControl (), [250](#)
- isToolFlangeEnabled
 - RobotState (), [478](#)
- isToolIoInput
 - IoControl (IO), [188](#)
- isWithinSafetyLimits
 - RobotState (), [479](#)
- JOINT_ERRORS
 - error_stack.h, [673](#)
 - hal_error.h, [676](#)
- joint_torque
 - arcs::common_interface::RobotSafetyParameterRange, [624](#)
- jointOffsetDisable
 - MotionControl (), [251](#)
- jointOffsetEnable
 - MotionControl (), [252](#)
- jointOffsetSet
 - MotionControl (), [252](#)
- JointServoModeType
 - arcs::common_interface, [559](#)
- JointStateType
 - arcs::common_interface, [559](#)
- level
 - arcs::common_interface::RobotMsg, [620](#)
- loadProgram
 - RuntimeMachine (), [491](#)
- lockRobotBrake
 - RobotManage (), [418](#)
- loop_times
 - arcs::common_interface::CircleParameters, [577](#)
- lower_joint_bound
 - arcs::common_interface::TrajConfig, [651](#)
- mass
 - arcs::common_interface::VibrationRecalibrationParameter, [653](#)
- Math
 - arcs::common_interface::Math, [595](#)
- Math (), [34](#)
 - calculateCircleFourthPoint, [35](#)
 - calibrateCoordinate, [36](#)
 - changePoseWithXYRef, [37](#)
 - deltaPoseAdd, [37](#)
 - deltaPoseTrans, [38](#)
 - forceTrans, [38](#)
 - getDeltaPoseBySensorDistance, [39](#)
 - homMatrixToPose, [39](#)
 - interpolatePose, [40](#)
 - poseAdd, [40](#)
 - poseAngleDistance, [41](#)
 - poseDistance, [42](#)
 - poseEqual, [42](#)
 - poseInverse, [43](#)

- poseRotation, [44](#)
- poseSub, [44](#)
- poseToHomMatrix, [45](#)
- poseTrans, [46](#)
- poseTransInv, [47](#)
- quaternionToRpy, [47](#)
- rpyToQuaternion, [48](#)
- tcpOffsetIdentify, [49](#)
- transferRefFrame, [50](#)
- MathPtr
 - arcs::common_interface, [553](#)
- max_acceleration
 - arcs::common_interface::TrajConfig, [651](#)
- max_velocity
 - arcs::common_interface::TrajConfig, [651](#)
- MB_ERR_ACKNOWLEDGE
 - register_control.h, [707](#)
- MB_ERR_DISCONNECTED
 - register_control.h, [707](#)
- MB_ERR_ILLEGAL_DATA_ADDRESS
 - register_control.h, [707](#)
- MB_ERR_ILLEGAL_DATA_VALUE
 - register_control.h, [707](#)
- MB_ERR_ILLEGAL_FUNCTION
 - register_control.h, [707](#)
- MB_ERR_NOT_INIT
 - register_control.h, [707](#)
- MB_ERR_SLAVE_DEVICE_BUSY
 - register_control.h, [707](#)
- MB_ERR_SLAVE_DEVICE_FAILURE
 - register_control.h, [707](#)
- message_
 - arcs::common_interface::AuboException, [573](#)
- method_not_found
 - arcs::common_interface, [558](#)
- mod
 - arcs::error_stack, [567](#)
- modbusAddSignal
 - RegisterControl (), [74](#)
- modbusDeleteAllSignals
 - RegisterControl (), [75](#)
- modbusDeleteSignal
 - RegisterControl (), [76](#)
- ModbusErrorNum
 - register_control.h, [707](#)
- modbusGetSignalError
 - RegisterControl (), [76](#)
- modbusGetSignalErrors
 - RegisterControl (), [77](#)
- modbusGetSignalIndex
 - RegisterControl (), [77](#)
- modbusGetSignalNames
 - RegisterControl (), [78](#)
- modbusGetSignalStatus
 - RegisterControl (), [78](#)
- modbusGetSignalTypes
 - RegisterControl (), [79](#)
- modbusGetSignalValues
 - RegisterControl (), [79](#)
- modbusSendCustomCommand
 - RegisterControl (), [80](#)
- modbusSetDigitalInputAction
 - RegisterControl (), [81](#)
- modbusSetOutputRunstate
 - RegisterControl (), [82](#)
- modbusSetOutputSignal
 - RegisterControl (), [83](#)
- modbusSetOutputSignal1
 - RegisterControl (), [84](#)
- modbusSetOutputSignalPulse
 - RegisterControl (), [84](#)
- modbusSetOutputSignalWithTimeout
 - arcs::common_interface::RegisterControl, [606](#)
- modbusSetSignalUpdateFrequency
 - RegisterControl (), [85](#)
- momentum
 - arcs::common_interface::RobotSafetyParameterRange, [624](#)
- motion_state
 - arcs::common_interface::GripperStatus, [588](#)
- MotionControl
 - arcs::common_interface::MotionControl, [601](#)
- MotionControl (), [219](#)
 - clearPath, [225](#)
 - conveyorTrackCircle, [225](#)
 - conveyorTrackClearItems, [226](#)
 - conveyorTrackCreatItem, [226](#)
 - conveyorTrackLine, [227](#)
 - conveyorTrackStop, [228](#)
 - conveyorTrackSwitch, [229](#)
 - disableJointSoftServo, [230](#)
 - disbaleVibrationSuppress, [231](#)
 - enableJointSoftServo, [231](#)
 - enableVibrationSuppress, [232](#)
 - followJoint, [232](#)
 - followLine, [232](#)
 - getConveyorTrackQueue, [233](#)
 - getDuration, [234](#)
 - getEqradius, [235](#)
 - getExecId, [235](#)
 - getFuturePathPointsJoint, [236](#)
 - getLookAheadSize, [237](#)
 - getMotionLeftTime, [237](#)
 - getPauseJointPositions, [238](#)
 - getProgress, [239](#)
 - getQueueSize, [240](#)
 - getResumeMode, [240](#)
 - getServoModeSelect, [241](#)
 - getSpeedFraction, [241](#)
 - getTcpMaxLinearVelocity, [242](#)
 - getTrajectoryQueueSize, [243](#)
 - getWorkObjectHold, [244](#)
 - hasItemOnConveyorToTrack, [244](#)
 - isBlending, [245](#)
 - isConveyorTrackExceed, [246](#)
 - isConveyorTrackSync, [247](#)

- isJointSoftServoEnabled, 247
- isServoModeEnabled, 248
- isSpeedFractionCritical, 249
- isSupportedTimeOptimal, 249
- isTimeOptimalEnabled, 250
- jointOffsetDisable, 251
- jointOffsetEnable, 252
- jointOffsetSet, 252
- moveCircle, 253
- moveCircle2, 254
- moveCircleWithAxisGroup, 255
- moveIntersection, 256
- moveJoint, 257
- moveJointWithAxisGroup, 259
- moveLine, 259
- moveLineWithAxisGroup, 261
- movePathBuffer, 261
- moveProcess, 262
- moveSpiral, 263
- moveSpline, 264
- pathBufferAlloc, 265
- pathBufferAppend, 266
- pathBufferEval, 267
- pathBufferFilter, 268
- pathBufferFree, 269
- pathBufferList, 270
- pathBufferValid, 271
- pathOffsetCoordinate, 272
- pathOffsetDisable, 273
- pathOffsetEnable, 274
- pathOffsetLimits, 275
- pathOffsetSet, 275
- pathOffsetSupv, 276
- resetTcpMaxLinearVelocity, 277
- restoPath, 278
- resumeMoveJoint, 279
- resumeMoveLine, 280
- resumeSpeedJoint, 281
- resumeSpeedLine, 282
- resumeStopJoint, 283
- resumeStopLine, 284
- servoCartesian, 285
- servoCartesianWithAxes, 287
- servoJoint, 288
- servoJointWithAxes, 289
- setCirclePathMode, 290
- setConveyorTrackCompensate, 291
- setConveyorTrackEncoder, 292
- setConveyorTrackLimit, 293
- setConveyorTrackSensorOffset, 294
- setConveyorTrackStartWindow, 295
- setConveyorTrackSyncSeparation, 296
- setEndPath, 297
- setEqradius, 298
- setFuturePointSamplePeriod, 299
- setLookAheadSize, 300
- setResumeStartPoint, 300
- setServoMode, 301
- setServoModeSelect, 302
- setSpeedFraction, 303
- setTcpMaxLinearVelocity, 304
- setTimeOptimalEnable, 305
- setWorkObjectHold, 305
- speedFractionCritical, 306
- speedJoint, 307
- speedLine, 308
- startMove, 310
- stopJoint, 310
- stopLine, 311
- stopMove, 312
- storePath, 313
- trackCartesian, 314
- trackJoint, 314
- weaveEnd, 315
- weaveStart, 316
- weaveUpdateParameters, 318
- MotionControlPtr
 - arcs::common_interface, 553
- move_axis
 - arcs::common_interface::TrajConfig, 651
- moveCircle
 - MotionControl (), 253
- moveCircle2
 - MotionControl (), 254
- moveCircleWithAxisGroup
 - MotionControl (), 255
- moveExtJoint
 - AxisInterface (), 28
- moveIntersection
 - MotionControl (), 256
- moveJoint
 - MotionControl (), 257
- moveJointWithAxisGroup
 - MotionControl (), 259
- moveLine
 - MotionControl (), 259
- moveLineWithAxisGroup
 - MotionControl (), 261
- movePathBuffer
 - MotionControl (), 261
- moveProcess
 - MotionControl (), 262
- moveSpiral
 - MotionControl (), 263
- moveSpline
 - MotionControl (), 264
- name
 - arcs::common_interface::GripperStatus, 588
- needVibrationRecalib
 - RobotAlgorithm (), 340
- newTask
 - RuntimeMachine (), 492
- None
 - arcs::common_interface, 561
- nop
 - RuntimeMachine (), 492

- notify
 - arcs::common_interface::Trace, [649](#)
- obj_coord
 - arcs::common_interface::WObjectData, [654](#)
- object_detection
 - arcs::common_interface::GripperStatus, [588](#)
- OperationalModeType
 - arcs::common_interface, [560](#)
- operator<<
 - arcs::common_interface, [564](#), [565](#)
- orig
 - arcs::common_interface::RobotSafetyParameterRange, [624](#)
- params
 - arcs::common_interface::RobotSafetyParameterRange, [624](#)
- parse_error
 - arcs::common_interface, [558](#)
- Path
 - arcs::common_interface, [561](#)
- pathBlend3Points
 - RobotAlgorithm (), [341](#)
- PathBuffer_CubicSpline
 - arcs::common_interface, [560](#)
- PathBuffer_JointBSpline
 - arcs::common_interface, [560](#)
- PathBuffer_JointBSplineC
 - arcs::common_interface, [560](#)
- PathBuffer_JointSpline
 - arcs::common_interface, [560](#)
- PathBuffer_JointSplineC
 - arcs::common_interface, [560](#)
- PathBuffer_TOPPRA
 - arcs::common_interface, [560](#)
- pathBufferAlloc
 - MotionControl (), [265](#)
- pathBufferAppend
 - MotionControl (), [266](#)
- pathBufferEval
 - MotionControl (), [267](#)
- pathBufferFilter
 - MotionControl (), [268](#)
- pathBufferFree
 - MotionControl (), [269](#)
- pathBufferList
 - MotionControl (), [270](#)
- PathBufferType
 - arcs::common_interface, [560](#)
- pathBufferValid
 - MotionControl (), [271](#)
- pathMovej
 - arcs::common_interface::RobotAlgorithm, [609](#)
- pathMoveS
 - RobotAlgorithm (), [341](#)
- pathOffsetCoordinate
 - MotionControl (), [272](#)
- pathOffsetDisable
 - MotionControl (), [273](#)
- pathOffsetEnable
 - MotionControl (), [274](#)
- pathOffsetLimits
 - MotionControl (), [275](#)
- pathOffsetSet
 - MotionControl (), [275](#)
- pathOffsetSupv
 - MotionControl (), [276](#)
- pause
 - RuntimeMachine (), [492](#)
- pauseRecord
 - RobotManage (), [418](#)
- pauseRecordCache
 - RobotManage (), [419](#)
- Payload
 - arcs::common_interface, [554](#)
- payloadCalculateFinished
 - RobotAlgorithm (), [342](#)
- payloadIdentify
 - RobotAlgorithm (), [342](#)
- payloadIdentify1
 - RobotAlgorithm (), [343](#)
- PayloadIdentifyMoveAxis
 - arcs::common_interface, [560](#)
- payloadIdentifyTrajGenFinished
 - RobotAlgorithm (), [344](#)
- PEDSTRAL_ERRORS
 - error_stack.h, [673](#)
 - hal_error.h, [677](#)
- peek
 - Trace (), [541](#)
- plane
 - arcs::common_interface::RobotSafetyParameterRange, [624](#)
 - arcs::common_interface::SpiralParameters, [644](#)
- planes
 - arcs::common_interface::RobotSafetyParameterRange, [624](#)
- points
 - arcs::common_interface::VibrationRecalibrationParameter, [653](#)
- popup
 - Trace (), [542](#)
- pose_to
 - arcs::common_interface::CircleParameters, [578](#)
- pose_via
 - arcs::common_interface::CircleParameters, [578](#)
- poseAdd
 - Math (), [40](#)
- poseAngleDistance
 - Math (), [41](#)
- poseDistance
 - Math (), [42](#)
- poseEqual

- Math (), 42
- poseInverse
 - Math (), 43
- poseRotation
 - Math (), 44
- poseSub
 - Math (), 44
- poseToHomMatrix
 - Math (), 45
- poseTrans
 - Math (), 46
- poseTransInv
 - Math (), 47
- position
 - arcs::common_interface::GripperStatus, 588
- power
 - arcs::common_interface::RobotSafetyParameterRange, 625
- poweroff
 - RobotManage (), 420
- poweroffExtAxis
 - AxisInterface (), 28
- poweron
 - RobotManage (), 420
- poweronExtAxis
 - AxisInterface (), 29
- preloadProgram
 - RuntimeMachine (), 493
- qdma
 - arcs::common_interface::RobotSafetyParameterRange, 625
- qmax
 - arcs::common_interface::RobotSafetyParameterRange, 625
- qmin
 - arcs::common_interface::RobotSafetyParameterRange, 625
- quaternionToRpy
 - Math (), 47
- r_velocity
 - arcs::common_interface::GripperStatus, 589
- recordCacheFree
 - RobotManage (), 421
- reduced_entry_distance
 - arcs::common_interface::RobotSafetyParameterRange, 625
- reduced_entry_time
 - arcs::common_interface::RobotSafetyParameterRange, 625
- RefFrameType
 - arcs::common_interface, 561
- register_control.h
 - MB_ERR_ACKNOWLEDGE, 707
 - MB_ERR_DISCONNECTED, 707
 - MB_ERR_ILLEGAL_DATA_ADDRESS, 707
 - MB_ERR_ILLEGAL_DATA_VALUE, 707
 - MB_ERR_ILLEGAL_FUNCTION, 707
 - MB_ERR_NOT_INIT, 707
 - MB_ERR_SLAVE_DEVICE_BUSY, 707
 - MB_ERR_SLAVE_DEVICE_FAILURE, 707
 - ModbusErrorNum, 707
- RegisterControl
 - arcs::common_interface::RegisterControl, 605
- RegisterControl (), 51
 - addInt32RegEncoder, 54
 - addModbusEncoder, 54
 - clearNamedVariable, 55
 - deleteVirtualEncoder, 55
 - getBool, 56
 - getBoolInput, 56
 - getBoolOutput, 57
 - getDouble, 58
 - getDoubleInput, 59
 - getDoubleOutput, 60
 - getFloat, 60
 - getFloatInput, 61
 - getFloatOutput, 62
 - getInt16Register, 63
 - getInt16RegisterBit, 64
 - getInt32, 65
 - getInt32Input, 65
 - getInt32Output, 66
 - getModbusDeviceStatus, 67
 - getNamedVariableType, 68
 - getString, 68
 - getVecChar, 69
 - getVecDouble, 70
 - getVecFloat, 71
 - getVecInt32, 72
 - getWatchDogAction, 72
 - getWatchDogTimeout, 73
 - hasNamedVariable, 73
 - modbusAddSignal, 74
 - modbusDeleteAllSignals, 75
 - modbusDeleteSignal, 76
 - modbusGetSignalError, 76
 - modbusGetSignalErrors, 77
 - modbusGetSignalIndex, 77
 - modbusGetSignalNames, 78
 - modbusGetSignalStatus, 78
 - modbusGetSignalTypes, 79
 - modbusGetSignalValues, 79
 - modbusSendCustomCommand, 80
 - modbusSetDigitalInputAction, 81
 - modbusSetOutputRunstate, 82
 - modbusSetOutputSignal, 83
 - modbusSetOutputSignal1, 84
 - modbusSetOutputSignalPulse, 84
 - modbusSetSignalUpdateFrequency, 85
 - setBool, 86
 - setBoolInput, 87
 - setBoolOutput, 88
 - setDouble, 88

- setDoubleInput, [89](#)
- setDoubleOutput, [90](#)
- setFloat, [91](#)
- setFloatInput, [92](#)
- setFloatOutput, [93](#)
- setInt16Register, [93](#)
- setInt16RegisterBit, [94](#)
- setInt32, [95](#)
- setInt32Input, [96](#)
- setInt32Output, [97](#)
- setString, [97](#)
- setVecChar, [98](#)
- setVecDouble, [99](#)
- setVecFloat, [100](#)
- setVecInt32, [101](#)
- setWatchDog, [101](#)
- variableUpdated, [102](#)
- RegisterControlPtr
 - arcs::common_interface, [554](#)
- releaseRobotBrake
 - RobotManage (), [422](#)
- remote_tool
 - arcs::common_interface::WObjectData, [655](#)
- removeBreakPoint
 - RuntimeMachine (), [493](#)
- resetDamping
 - ForceControl(), [118](#)
- resetLpFilter
 - ForceControl(), [118](#)
- resetTcpMaxLinearVelocity
 - MotionControl (), [277](#)
- restartInterfaceBoard
 - RobotManage (), [423](#)
- restoPath
 - MotionControl (), [278](#)
- restrict_elbow
 - arcs::common_interface::RobotSafetyParameterRange, [626](#)
- ResultWithErrno
 - arcs::common_interface, [554](#)
- ResultWithErrno1
 - arcs::common_interface, [554](#)
- ResultWithErrno2
 - arcs::common_interface, [554](#)
- ResultWithErrno3
 - arcs::common_interface, [554](#)
- resume
 - RuntimeMachine (), [494](#)
- resumeMoveJoint
 - MotionControl (), [279](#)
- resumeMoveLine
 - MotionControl (), [280](#)
- resumeSpeedJoint
 - MotionControl (), [281](#)
- resumeSpeedLine
 - MotionControl (), [282](#)
- resumeStopJoint
 - MotionControl (), [283](#)
- resumeStopLine
 - MotionControl (), [284](#)
- returnValue2Str
 - arcs::common_interface, [565](#)
- RobotAlgorithm
 - arcs::common_interface::RobotAlgorithm, [609](#)
- RobotAlgorithm (), [319](#)
 - calcJacobian, [321](#)
 - calibrateTcpForceSensor, [322](#)
 - calibrateTcpForceSensor2, [323](#)
 - calibrateTcpForceSensor3, [324](#)
 - calibVibrationParams, [324](#)
 - calibVibrationParams1, [324](#)
 - calibWorkpieceCoordinatePara, [325](#)
 - forwardDynamics, [325](#)
 - forwardDynamics1, [326](#)
 - forwardKinematics, [327](#)
 - forwardKinematics1, [328](#)
 - forwardKinematicsAll, [329](#)
 - forwardToolKinematics, [330](#)
 - frictionModelIdentify, [331](#)
 - generatePayloadIdentifyTraj, [332](#)
 - getPayloadIdentifyResult, [333](#)
 - getRobotConfiguration, [333](#)
 - inverseKinematics, [335](#)
 - inverseKinematics1, [336](#)
 - inverseKinematicsAll, [337](#)
 - inverseKinematicsAll1, [338](#)
 - inverseToolKinematics, [339](#)
 - inverseToolKinematicsAll, [340](#)
 - needVibrationRecalib, [340](#)
 - pathBlend3Points, [341](#)
 - pathMoveS, [341](#)
 - payloadCalculateFinished, [342](#)
 - payloadIdentify, [342](#)
 - payloadIdentify1, [343](#)
 - payloadIdentifyTrajGenFinished, [344](#)
 - validatePath, [344](#)
- RobotAlgorithmPtr
 - arcs::common_interface, [554](#)
- RobotConfig
 - arcs::common_interface::RobotConfig, [614](#)
- RobotConfig (), [346](#)
 - attachRobotBaseTo, [350](#)
 - attachWeldingGun, [350](#)
 - calcSafetyParametersChecksum, [350](#)
 - confirmSafetyParameters, [350](#)
 - enableEndCollisionCheck, [351](#)
 - firmwareUpdate, [351](#)
 - getBaseForceOffset, [352](#)
 - getBaseForceSensorNames, [353](#)
 - getCollisionLevel, [354](#)
 - getCollisionStopType, [354](#)
 - getCollisionThreshold, [355](#)
 - getControlBoxType, [356](#)
 - getCycletime, [357](#)
 - getDefaultJointAcc, [357](#)
 - getDefaultJointSpeed, [358](#)

- getDefaultToolAcc, 359
- getDefaultToolSpeed, 360
- getDof, 360
- getFirmwareUpdateProcess, 361
- getFreedriveDamp, 362
- getGravity, 363
- getHandguidDamp, 363
- getHardwareCustomParameters, 364
- getHomePosition, 365
- getJointMaxAccelerations, 365
- getJointMaxPositions, 366
- getJointMaxSpeeds, 367
- getJointMinPositions, 367
- getKinematicsCompensate, 368
- getKinematicsParam, 369
- getLimitJointMaxAccelerations, 370
- getLimitJointMaxPositions, 371
- getLimitJointMaxSpeeds, 372
- getLimitJointMinPositions, 372
- getLimitTcpMaxSpeed, 373
- getMountingPose, 374
- getName, 375
- getPayload, 375
- getRobotEmergencyStopSource, 376
- getRobotSubType, 376
- getRobotType, 377
- getSafeguardStopSource, 378
- getSafeguardStopType, 378
- getSafetyParametersChecksum, 379
- getSelectedTcpForceSensorName, 379
- getSlowDownFraction, 380
- getTcpForceOffset, 381
- getTcpForceSensorNames, 381
- getTcpForceSensorPose, 382
- getTcpMaxAccelerations, 382
- getTcpMaxSpeeds, 383
- getTcpOffset, 384
- hasBaseForceSensor, 384
- hasTcpForceSensor, 385
- isEndCollisionCheckEnabled, 386
- selectBaseForceSensor, 387
- selectTcpForceSensor, 387
- setBaseForceOffset, 388
- setCollisionLevel, 389
- setCollisionStopType, 390
- setCollisionThreshold, 391
- setFreedriveDamp, 391
- setGravity, 392
- setHandguidDamp, 393
- setHardwareCustomParameters, 394
- setHomePosition, 395
- setKinematicsCompensate, 395
- setMountingPose, 396
- setPayload, 397
- setPersistentParameters, 398
- setRobotZero, 398
- setSlowDownFraction, 399
- setTcpForceOffset, 400
- setTcpForceSensorPose, 401
- setTcpOffset, 401
- setToolInertial, 402
- setWorkObjectData, 403
- toolSpaceInRange, 403
- RobotConfigPtr
 - arcs::common_interface, 555
- RobotControlModeType
 - arcs::common_interface, 561
- RobotEmergencyStopType
 - arcs::common_interface, 561
- RobotInterface
 - arcs::common_interface::RobotInterface, 616
- RobotInterface(), 103
 - getForceControl, 104
 - getIoControl, 104
 - getMotionControl, 105
 - getRobotAlgorithm, 105
 - getRobotConfig, 105
 - getRobotManage, 106
 - getRobotState, 106
 - getSyncMove, 106
 - getTrace, 107
- RobotInterfacePtr
 - arcs::common_interface, 555
- RobotManage
 - arcs::common_interface::RobotManage, 618
- RobotManage (), 404
 - backdrive, 406
 - exitHandguideMode, 407
 - freedrive, 408
 - generateDiagnoseFile, 408
 - getHandguideFeature, 409
 - getHandguideFreeAxes, 410
 - getHandguideStatus, 410
 - getHandguideTrigger, 410
 - getOperationalMode, 411
 - getRecordCache, 411
 - getRobotControlMode, 412
 - handguideMode, 413
 - isBackdriveEnabled, 414
 - isFreedriveEnabled, 414
 - isHandguideEnabled, 415
 - isLinkModeEnabled, 416
 - isSimulationEnabled, 417
 - lockRobotBrake, 418
 - pauseRecord, 418
 - pauseRecordCache, 419
 - poweroff, 420
 - poweron, 420
 - recordCacheFree, 421
 - releaseRobotBrake, 422
 - restartInterfaceBoard, 423
 - setHandguideParams, 424
 - setLinkModeEnable, 424
 - setOperationalMode, 425
 - setSim, 426
 - setUnlockProtectiveStop, 427

- startRecord, [428](#)
- startRecordCache, [429](#)
- startup, [429](#)
- stopRecord, [430](#)
- stopRecordCache, [431](#)
- RobotManagePtr
 - arcs::common_interface, [555](#)
- RobotModeType
 - arcs::common_interface, [562](#)
- RobotMsgVector
 - arcs::common_interface, [555](#)
- RobotSafetyParameterRange
 - arcs::common_interface::RobotSafetyParameterRange, [623](#)
- RobotState
 - arcs::common_interface::RobotState, [634](#)
- RobotState (), [432](#)
 - getActualTcpOffset, [435](#)
 - getBaseForce, [435](#)
 - getControlBoxHumidity, [435](#)
 - getControlBoxTemperature, [436](#)
 - getElbowPosistion, [437](#)
 - getElbowVelocity, [438](#)
 - getJointAccelerations, [438](#)
 - getJointContactTorques, [439](#)
 - getJointCurrents, [440](#)
 - getJointFirmwareVersions, [441](#)
 - getJointGravityTorques, [441](#)
 - getJointHardwareVersions, [442](#)
 - getJointPositions, [443](#)
 - getJointPositionsHistory, [444](#)
 - getJointServoMode, [444](#)
 - getJointSpeeds, [445](#)
 - getJointState, [446](#)
 - getJointTargetAccelerations, [446](#)
 - getJointTargetCurrents, [447](#)
 - getJointTargetPositions, [448](#)
 - getJointTargetSpeeds, [449](#)
 - getJointTargetTorques, [449](#)
 - getJointTemperatures, [450](#)
 - getJointTorqueSensors, [451](#)
 - getJointUniqueIds, [452](#)
 - getJointVoltages, [452](#)
 - getMainCurrent, [453](#)
 - getMainVoltage, [454](#)
 - getMasterBoardFirmwareVersion, [455](#)
 - getMasterBoardHardwareVersion, [455](#)
 - getMasterBoardUniqueId, [456](#)
 - getPedestalFirmwareVersion, [457](#)
 - getPedestalHardwareVersion, [458](#)
 - getPedestalUniqueId, [458](#)
 - getRobotCurrent, [459](#)
 - getRobotModeType, [460](#)
 - getRobotVoltage, [461](#)
 - getSafetyModeType, [461](#)
 - getSlaveBoardFirmwareVersion, [462](#)
 - getSlaveBoardHardwareVersion, [463](#)
 - getSlaveBoardUniqueId, [464](#)
 - getSlowDownLevel, [464](#)
 - getTargetTcpPose, [465](#)
 - getTcpForce, [466](#)
 - getTcpForceSensors, [466](#)
 - getTcpForceSensorStatus, [467](#)
 - getTcpPose, [468](#)
 - getTcpSpeed, [468](#)
 - getTcpTargetForce, [469](#)
 - getTcpTargetPose, [470](#)
 - getTcpTargetSpeed, [471](#)
 - getToolCommMode, [471](#)
 - getToolFirmwareVersion, [472](#)
 - getToolHardwareVersion, [473](#)
 - getToolPose, [474](#)
 - getToolUniqueId, [474](#)
 - isCollisionOccurred, [475](#)
 - isPowerOn, [476](#)
 - isSteady, [476](#)
 - isTeachPendantEnabled, [477](#)
 - isToolFlangeEnabled, [478](#)
 - isWithinSafetyLimits, [479](#)
- RobotStatePtr
 - arcs::common_interface, [555](#)
- rpyToQuaternion
 - Math (), [48](#)
- rtm_error.h
 - RTM_ERRORS, [681](#)
- RTM_ERRORS
 - error_stack.h, [673](#)
 - rtm_error.h, [681](#)
- Running
 - arcs::common_interface, [559](#)
- runProgram
 - RuntimeMachine (), [494](#)
- RuntimeMachine
 - arcs::common_interface::RuntimeMachine, [640](#)
- RuntimeMachine (), [480](#)
 - abort, [482](#)
 - arbitraryResume, [482](#)
 - clearBreakPoints, [482](#)
 - clearPreloadPrograms, [483](#)
 - deleteTask, [483](#)
 - detachTask, [483](#)
 - enterCritical, [484](#)
 - exitCritical, [484](#)
 - getAdvancePlanContext, [485](#)
 - getAdvancePtr, [485](#)
 - getExecutionStatus, [486](#)
 - getInterpPtr, [486](#)
 - getMainPtr, [487](#)
 - getPlanContext, [487](#)
 - getPreloadProgram, [488](#)
 - getStatus, [488](#)
 - getTaskQueueSize, [489](#)
 - getTimer, [489](#)
 - getTriggInterrupts, [490](#)
 - gotoLine, [490](#)

- isTaskAlive, [491](#)
- loadProgram, [491](#)
- newTask, [492](#)
- nop, [492](#)
- pause, [492](#)
- preloadProgram, [493](#)
- removeBreakPoint, [493](#)
- resume, [494](#)
- runProgram, [494](#)
- setBreakPoint, [495](#)
- setLabel, [495](#)
- setPlanContext, [496](#)
- setResumeWait, [496](#)
- start, [497](#)
- step, [497](#)
- stop, [498](#)
- switchTask, [498](#)
- timerDelete, [499](#)
- timerReset, [499](#)
- timerStart, [500](#)
- timerStop, [500](#)
- triggBegin, [501](#)
- triggEnd, [502](#)
- triggInterrupt, [502](#)
- RuntimeMachinePtr
 - arcs::common_interface, [555](#)
- RuntimeState
 - arcs::common_interface, [562](#)
- SafeguardedStopType
 - arcs::common_interface, [562](#)
- SAFETY_CUBIC_NUM
 - type_def.h, [1118](#)
- safety_home
 - arcs::common_interface::RobotSafetyParameterRange, [629](#)
- safety_input_auto_safeguard_reset
 - arcs::common_interface::RobotSafetyParameterRange, [626](#)
- safety_input_auto_safeguard_stop
 - arcs::common_interface::RobotSafetyParameterRange, [626](#)
- safety_input_emergency_stop
 - arcs::common_interface::RobotSafetyParameterRange, [626](#)
- safety_input_handguide
 - arcs::common_interface::RobotSafetyParameterRange, [626](#)
- safety_input_operational_mode
 - arcs::common_interface::RobotSafetyParameterRange, [627](#)
- safety_input_reduced_mode
 - arcs::common_interface::RobotSafetyParameterRange, [627](#)
- safety_input_safeguard_reset
 - arcs::common_interface::RobotSafetyParameterRange, [627](#)
- safety_input_safeguard_stop
 - arcs::common_interface::RobotSafetyParameterRange, [627](#)
- safety_input_three_position_switch
 - arcs::common_interface::RobotSafetyParameterRange, [627](#)
- SAFETY_INTERFACE_BOARD_ERRORS
 - error_stack.h, [673](#)
 - hal_error.h, [677](#)
- safety_output_emergency_stop
 - arcs::common_interface::RobotSafetyParameterRange, [627](#)
- safety_output_not_emergency_stop
 - arcs::common_interface::RobotSafetyParameterRange, [628](#)
- safety_output_not_reduced_mode
 - arcs::common_interface::RobotSafetyParameterRange, [628](#)
- safety_output_reduced_mode
 - arcs::common_interface::RobotSafetyParameterRange, [628](#)
- safety_output_robot_moving
 - arcs::common_interface::RobotSafetyParameterRange, [628](#)
- safety_output_robot_not_stopping
 - arcs::common_interface::RobotSafetyParameterRange, [628](#)
- safety_output_robot_steady
 - arcs::common_interface::RobotSafetyParameterRange, [628](#)
- safety_output_safe_home
 - arcs::common_interface::RobotSafetyParameterRange, [629](#)
- safety_output_safetyguard_stop
 - arcs::common_interface::RobotSafetyParameterRange, [629](#)
- SAFETY_PARAM_SELECT_NUM
 - type_def.h, [1118](#)
- SAFETY_PLANES_NUM
 - type_def.h, [1118](#)
- SafetyInputAction
 - arcs::common_interface, [562](#)
- SafetyModeType
 - arcs::common_interface, [563](#)
- SafetyOutputRunState
 - arcs::common_interface, [563](#)
- segments
 - arcs::common_interface::RtdeRecipe, [637](#)
- selectBaseForceSensor
 - RobotConfig (), [387](#)
- selectTcpForceSensor
 - RobotConfig (), [387](#)
- Serial
 - arcs::common_interface::Serial, [641](#)
 - Serial (), [503](#)
 - serialClose, [504](#)
 - serialOpen, [504](#)
 - serialReadByte, [505](#)
 - serialReadByteList, [505](#)

- serialReadString, [506](#)
- serialSendAllString, [507](#)
- serialSendByte, [507](#)
- serialSendInt, [508](#)
- serialSendLine, [509](#)
- serialSendString, [509](#)
- serialClose
 - Serial (), [504](#)
- serialOpen
 - Serial (), [504](#)
- SerialPtr
 - arcs::common_interface, [555](#)
- serialReadByte
 - Serial (), [505](#)
- serialReadByteList
 - Serial (), [505](#)
- serialReadString
 - Serial (), [506](#)
- serialSendAllString
 - Serial (), [507](#)
- serialSendByte
 - Serial (), [507](#)
- serialSendInt
 - Serial (), [508](#)
- serialSendLine
 - Serial (), [509](#)
- serialSendString
 - Serial (), [509](#)
- server_error
 - arcs::common_interface, [558](#)
- servoCartesian
 - MotionControl (), [285](#)
- servoCartesianWithAxes
 - MotionControl (), [287](#)
- servoCartesianWithAxisGroup
 - arcs::common_interface::MotionControl, [601](#)
- servoJoint
 - MotionControl (), [288](#)
- servoJointWithAxes
 - MotionControl (), [289](#)
- servoJointWithAxisGroup
 - arcs::common_interface::MotionControl, [602](#)
- setAnalogOutputAfterEStopDefault
 - IoControl (IO), [189](#)
- setBaseForceOffset
 - RobotConfig (), [388](#)
- setBool
 - RegisterControl (), [86](#)
- setBoolInput
 - RegisterControl (), [87](#)
- setBoolOutput
 - RegisterControl (), [88](#)
- setBreakPoint
 - RuntimeMachine (), [495](#)
- setCirclePathMode
 - MotionControl (), [290](#)
- setCollisionLevel
 - RobotConfig (), [389](#)
- setCollisionStopType
 - RobotConfig (), [390](#)
- setCollisionThreshold
 - RobotConfig (), [391](#)
- setCondActive
 - ForceControl(), [119](#)
- setCondAdvanced
 - ForceControl(), [120](#)
- setCondCylinder
 - ForceControl(), [121](#)
- setCondDistance
 - ForceControl(), [122](#)
- setCondForce
 - ForceControl(), [122](#)
- setCondOrient
 - ForceControl(), [123](#)
- setCondPlane
 - ForceControl(), [124](#)
- setCondSphere
 - ForceControl(), [125](#)
- setCondTcpSpeed
 - ForceControl(), [126](#)
- setConfigurableDigitalInputAction
 - IoControl (IO), [190](#)
- setConfigurableDigitalOutput
 - IoControl (IO), [191](#)
- setConfigurableDigitalOutputAfterEStop
 - IoControl (IO), [192](#)
- setConfigurableDigitalOutputPulse
 - IoControl (IO), [193](#)
- setConfigurableDigitalOutputRunstate
 - IoControl (IO), [194](#)
- setConveyorTrackCompensate
 - MotionControl (), [291](#)
- setConveyorTrackEncoder
 - MotionControl (), [292](#)
- setConveyorTrackLimit
 - MotionControl (), [293](#)
- setConveyorTrackSensorOffset
 - MotionControl (), [294](#)
- setConveyorTrackStartWindow
 - MotionControl (), [295](#)
- setConveyorTrackSyncSeparation
 - MotionControl (), [296](#)
- setDamping
 - ForceControl(), [127](#)
- setDigitalInputActionDefault
 - IoControl (IO), [195](#)
- setDigitalOutputAfterEStopDefault
 - IoControl (IO), [196](#)
- setDigitalOutputRunstateDefault
 - IoControl (IO), [197](#)
- setDouble
 - RegisterControl (), [88](#)
- setDoubleInput
 - RegisterControl (), [89](#)
- setDoubleOutput
 - RegisterControl (), [90](#)

- setDynamicModel
 - ForceControl(), [128](#)
- setDynamicModel1
 - ForceControl(), [129](#)
- setDynamicModelContact
 - ForceControl(), [130](#)
- setDynamicModelInsert
 - ForceControl(), [131](#)
- setDynamicModelSearch
 - ForceControl(), [132](#)
- setEncDecoderType
 - IoControl (IO), [198](#)
- setEncTickCount
 - IoControl (IO), [198](#)
- setEndPath
 - MotionControl (), [297](#)
- setEqradius
 - MotionControl (), [298](#)
- setExtAxisMountingPose
 - AxisInterface (), [29](#)
- setFloat
 - RegisterControl (), [91](#)
- setFloatInput
 - RegisterControl (), [92](#)
- setFloatOutput
 - RegisterControl (), [93](#)
- setFreedriveDamp
 - RobotConfig (), [391](#)
- setFuturePointSamplePeriod
 - MotionControl (), [299](#)
- setGravity
 - RobotConfig (), [392](#)
- setHandguidDamp
 - RobotConfig (), [393](#)
- setHandguideParams
 - RobotManage (), [424](#)
- setHardwareCustomParameters
 - RobotConfig (), [394](#)
- setHomePosition
 - RobotConfig (), [395](#)
- setInt16Register
 - RegisterControl (), [93](#)
- setInt16RegisterBit
 - RegisterControl (), [94](#)
- setInt32
 - RegisterControl (), [95](#)
- setInt32Input
 - RegisterControl (), [96](#)
- setInt32Output
 - RegisterControl (), [97](#)
- setKinematicsCompensate
 - RobotConfig (), [395](#)
- setLabel
 - RuntimeMachine (), [495](#)
- setLinkModeEnable
 - RobotManage (), [424](#)
- setLookAheadSize
 - MotionControl (), [300](#)
- setLpFilter
 - ForceControl(), [132](#)
- setMountingPose
 - RobotConfig (), [396](#)
- setOperationalMode
 - RobotManage (), [425](#)
- setPayload
 - RobotConfig (), [397](#)
- setPersistentParameters
 - RobotConfig (), [398](#)
- setPlanContext
 - RuntimeMachine (), [496](#)
- setResumeStartPoint
 - MotionControl (), [300](#)
- setResumeWait
 - RuntimeMachine (), [496](#)
- setRobotZero
 - RobotConfig (), [398](#)
- setServoMode
 - MotionControl (), [301](#)
- setServoModeSelect
 - MotionControl (), [302](#)
- setSim
 - RobotManage (), [426](#)
- setSlowDownFraction
 - RobotConfig (), [399](#)
- setSoftFloatParams
 - ForceControl(), [133](#)
- setSpeedFraction
 - MotionControl (), [303](#)
- setStandardAnalogInputDomain
 - IoControl (IO), [199](#)
- setStandardAnalogOutput
 - IoControl (IO), [200](#)
- setStandardAnalogOutputAfterEStop
 - IoControl (IO), [200](#)
- setStandardAnalogOutputDomain
 - IoControl (IO), [201](#)
- setStandardAnalogOutputRunstate
 - IoControl (IO), [202](#)
- setStandardDigitalInputAction
 - IoControl (IO), [203](#)
- setStandardDigitalOutput
 - IoControl (IO), [204](#)
- setStandardDigitalOutputAfterEStop
 - IoControl (IO), [205](#)
- setStandardDigitalOutputPulse
 - IoControl (IO), [206](#)
- setStandardDigitalOutputRunstate
 - IoControl (IO), [207](#)
- setString
 - RegisterControl (), [97](#)
- setSupvForce
 - ForceControl(), [134](#)
- setSupvOrient
 - ForceControl(), [135](#)
- setSupvPosBox
 - ForceControl(), [136](#)

- setSupvPosCylinder
 - ForceControl(), [136](#)
- setSupvPosSphere
 - ForceControl(), [137](#)
- setSupvReoriSpeed
 - ForceControl(), [138](#)
- setSupvTcpSpeed
 - ForceControl(), [139](#)
- setTargetForce
 - ForceControl(), [140](#)
- setTcpForceOffset
 - RobotConfig (), [400](#)
- setTcpForceSensorPose
 - RobotConfig (), [401](#)
- setTcpMaxLinearVelocity
 - MotionControl (), [304](#)
- setTcpOffset
 - RobotConfig (), [401](#)
- setTimeOptimalEnable
 - MotionControl (), [305](#)
- setToolAnalogInputDomain
 - IoControl (IO), [208](#)
- setToolAnalogOutput
 - IoControl (IO), [209](#)
- setToolAnalogOutputDomain
 - IoControl (IO), [210](#)
- setToolAnalogOutputRunstate
 - IoControl (IO), [211](#)
- setToolDigitalInputAction
 - IoControl (IO), [212](#)
- setToolDigitalOutput
 - IoControl (IO), [213](#)
- setToolDigitalOutputPulse
 - IoControl (IO), [214](#)
- setToolDigitalOutputRunstate
 - IoControl (IO), [215](#)
- setToolInertial
 - RobotConfig (), [402](#)
- setToolIoInput
 - IoControl (IO), [216](#)
- setToolVoltageOutputDomain
 - IoControl (IO), [217](#)
- setUnlockProtectiveStop
 - RobotManage (), [427](#)
- setVecChar
 - RegisterControl (), [98](#)
- setVecDouble
 - RegisterControl (), [99](#)
- setVecFloat
 - RegisterControl (), [100](#)
- setVecInt32
 - RegisterControl (), [101](#)
- setWatchDog
 - RegisterControl (), [101](#)
- setWorkObjectData
 - RobotConfig (), [403](#)
- setWorkObjectHold
 - MotionControl (), [305](#)
- shape
 - arcs::common_interface::Enveloping, [579](#)
- size
 - arcs::common_interface::RobotSafetyParameterRange, [629](#)
- Socket
 - arcs::common_interface::Socket, [643](#)
- Socket (socket), [510](#)
 - socketClose, [511](#)
 - socketHasConnected, [512](#)
 - socketOpen, [512](#)
 - socketReadAllString, [513](#)
 - socketReadAsciiFloat, [514](#)
 - socketReadBinaryInteger, [514](#)
 - socketReadByteList, [515](#)
 - socketReadString, [516](#)
 - socketSendAllString, [517](#)
 - socketSendByte, [518](#)
 - socketSendInt, [518](#)
 - socketSendLine, [519](#)
 - socketSendString, [520](#)
- socketClose
 - Socket (socket), [511](#)
- socketHasConnected
 - Socket (socket), [512](#)
- socketOpen
 - Socket (socket), [512](#)
- SocketPtr
 - arcs::common_interface, [555](#)
- socketReadAllString
 - Socket (socket), [513](#)
- socketReadAsciiFloat
 - Socket (socket), [514](#)
- socketReadBinaryInteger
 - Socket (socket), [514](#)
- socketReadByteList
 - Socket (socket), [515](#)
- socketReadString
 - Socket (socket), [516](#)
- socketSendAllString
 - Socket (socket), [517](#)
- socketSendByte
 - Socket (socket), [518](#)
- socketSendInt
 - Socket (socket), [518](#)
- socketSendLine
 - Socket (socket), [519](#)
- socketSendString
 - Socket (socket), [520](#)
- softFloatDisable
 - ForceControl(), [141](#)
- softFloatEnable
 - ForceControl(), [141](#)
- source
 - arcs::common_interface::RobotMsg, [620](#)
- speedChangeDisable
 - ForceControl(), [142](#)
- speedChangeEnable

- ForceControl(), 142
- speedChangeTune
 - ForceControl(), 143
- speedExtJoint
 - AxisInterface (), 29
- speedFractionCritical
 - MotionControl (), 306
- speedJoint
 - MotionControl (), 307
- speedLine
 - MotionControl (), 308
- Sphere
 - arcs::common_interface, 556
- spiral
 - arcs::common_interface::CircleParameters, 578
 - arcs::common_interface::SpiralParameters, 644
- StandardInputAction
 - arcs::common_interface, 563
- StandardOutputRunState
 - arcs::common_interface, 563
- start
 - RuntimeMachine (), 497
- Starting
 - arcs::common_interface, 559
- startMove
 - MotionControl (), 310
- startRecord
 - RobotManage (), 428
- startRecordCache
 - RobotManage (), 429
- startup
 - RobotManage (), 429
- status_code
 - arcs::common_interface::GripperStatus, 589
- step
 - RuntimeMachine (), 497
- stiff_param
 - arcs::common_interface::VibrationRecalibrationParameters, 653
- stiff_section
 - arcs::common_interface::VibrationRecalibrationParameters, 653
- stl_path
 - arcs::common_interface::Enveloping, 579
- stop
 - RuntimeMachine (), 498
- stop_distance
 - arcs::common_interface::RobotSafetyParameterRange, 629
- stop_time
 - arcs::common_interface::RobotSafetyParameterRange, 629
- stopExtJoint
 - AxisInterface (), 30
- stopFollowAnotherAxis
 - AxisInterface (), 30
- stopJoint
 - MotionControl (), 310
- stopLine
 - MotionControl (), 311
- stopMove
 - MotionControl (), 312
- Stopped
 - arcs::common_interface, 559
- stopRecord
 - RobotManage (), 430
- stopRecordCache
 - RobotManage (), 431
- storePath
 - MotionControl (), 313
- str2ErrorCode
 - arcs::error_stack, 567
- Stropping
 - arcs::common_interface, 559
- switchTask
 - RuntimeMachine (), 498
- SyncMove
 - arcs::common_interface::SyncMove, 647
- SyncMove (), 521
 - axisGroupAdd, 523
 - axisGroupAddAxis, 523
 - axisGroupDelete, 524
 - axisGroupGetActualPositions, 524
 - axisGroupGetAxisIndex, 525
 - axisGroupGetAxisName, 525
 - axisGroupGetTargetPositions, 525
 - axisGroupMoveJoint, 526
 - axisGroupOffsetPositions, 526
 - axisGroupSpeedJoint, 527
 - axisGroupUpdateAxis, 527
 - frameAdd, 528
 - frameAttach, 528
 - frameConvertPose, 529
 - frameDelete, 529
 - frameDeleteAll, 530
 - frameExist, 530
 - frameGetChildren, 530
 - frameGetParent, 531
 - frameGetPose, 531
 - frameMove, 532
 - isSyncMoveOn, 532
 - syncMoveOff, 533
 - syncMoveOn, 533
 - syncMoveResume, 534
 - syncMoveSegment, 534
 - syncMoveSuspend, 535
 - syncMoveUndo, 535
 - waitSyncTasks, 535
- SyncMoveOff
 - SyncMove (), 533
- syncMoveOn
 - SyncMove (), 533
- SyncMovePtr
 - arcs::common_interface, 556

- syncMoveResume
 - SyncMove (), 534
- syncMoveSegment
 - SyncMove (), 534
- syncMoveSuspend
 - SyncMove (), 535
- syncMoveUndo
 - SyncMove (), 535
- system_error.h
 - SYSTEM_ERRORS, 686
- SYSTEM_ERRORS
 - error_stack.h, 673
 - system_error.h, 686
- SystemInfo
 - arcs::common_interface::SystemInfo, 648
- SystemInfo (), 536
 - getControlSoftwareBuildDate, 537
 - getControlSoftwareFullVersion, 537
 - getControlSoftwareVersionCode, 538
 - getControlSoftwareVersionHash, 538
 - getControlSystemTime, 539
 - getInterfaceVersionCode, 539
- SystemInfoPtr
 - arcs::common_interface, 556
- TaskFrameType
 - arcs::common_interface, 564
- TaskSet
 - arcs::common_interface, 556
- tcp_force
 - arcs::common_interface::RobotSafetyParameterRange, 629
- tcp_speed
 - arcs::common_interface::RobotSafetyParameterRange, 630
- tcpOffsetIdentify
 - Math (), 49
- temperature
 - arcs::common_interface::GripperStatus, 589
- textmsg
 - Trace (), 543
- timerDelete
 - RuntimeMachine (), 499
- timerReset
 - RuntimeMachine (), 499
- timerStart
 - RuntimeMachine (), 500
- timerStop
 - RuntimeMachine (), 500
- timestamp
 - arcs::common_interface::RobotMsg, 620
- to_server
 - arcs::common_interface::RtdeRecipe, 637
- Tool
 - arcs::common_interface, 561
- tool_azimuth
 - arcs::common_interface::RobotSafetyParameterRange, 630
- TOOL_CONFIGURATION_NUM
 - type_def.h, 1118
- tool_deviation
 - arcs::common_interface::RobotSafetyParameterRange, 630
- TOOL_ERRORS
 - error_stack.h, 674
 - hal_error.h, 677
- tool_inclination
 - arcs::common_interface::RobotSafetyParameterRange, 630
- tool_orientation
 - arcs::common_interface::RobotSafetyParameterRange, 630
- toolContact
 - ForceControl(), 144
- tools
 - arcs::common_interface::RobotSafetyParameterRange, 630
- toolSpaceInRange
 - RobotConfig (), 403
- torque
 - arcs::common_interface::GripperStatus, 589
- tp_3pe_for_handguide
 - arcs::common_interface::RobotSafetyParameterRange, 631
- Trace
 - arcs::common_interface::Trace, 649
- Trace (), 540
 - alarm, 541
 - peek, 541
 - popup, 542
 - textmsg, 543
- TraceLevel
 - arcs::common_interface, 564
- TracePtr
 - arcs::common_interface, 556
- trackCartesian
 - MotionControl (), 314
- trackJoint
 - MotionControl (), 314
- transferRefFrame
 - Math (), 50
- triggBegin
 - RuntimeMachine (), 501
- triggEnd
 - RuntimeMachine (), 502
- trigger
 - arcs::common_interface::RtdeRecipe, 637
- trigger_planes
 - arcs::common_interface::RobotSafetyParameterRange, 631
- triggInterrupt
 - RuntimeMachine (), 502
- type
 - arcs::common_interface::AuboException, 573
- type_def.h
 - CARTESIAN_DOF, 1111
 - DECL_TO_STRING_FUNC, 1111

- ENUM__AuboErrorCodes__DECLARES, [1111](#)
- ENUM__AxisModeType__DECLARES, [1111](#)
- ENUM__EnvelopingShape__DECLARES, [1111](#)
- ENUM__ITEM, [1112](#), [1113](#)
- ENUM__JointServoModeType__DECLARES, [1113](#)
- ENUM__JointStateType__DECLARES, [1113](#)
- ENUM__OperationalModeType__DECLARES, [1113](#)
- ENUM__PayloadIdentifyMoveAxis__DECLARES, [1114](#)
- ENUM__RobotControlModeType__DECLARES, [1114](#)
- ENUM__RobotEmergencyStopType__DECLARES, [1114](#)
- ENUM__RobotModeType__DECLARES, [1114](#)
- ENUM__RuntimeState__DECLARES, [1115](#)
- ENUM__SafeguedStopType__DECLARES, [1115](#)
- ENUM__SafetyInputAction__DECLARES, [1115](#)
- ENUM__SafetyModeType__DECLARES, [1116](#)
- ENUM__SafetyOutputRunState__DECLARES, [1116](#)
- ENUM__StandardInputAction__DECLARES, [1116](#)
- ENUM__StandardOutputRunState__DECLARES, [1117](#)
- ENUM__TaskFrameType__DECLARES, [1117](#)
- ENUM__TraceLevel__DECLARES, [1117](#)
- SAFETY__CUBIC__NUM, [1118](#)
- SAFETY__PARAM__SELECT__NUM, [1118](#)
- SAFETY__PLANES__NUM, [1118](#)
- TOOL__CONFIGURATION__NUM, [1118](#)
- unwindEncDeltaTickCount
 - IoControl (IO), [218](#)
- upper_joint_bound
 - arcs::common_interface::TrajConfig, [651](#)
- user_coord
 - arcs::common_interface::WObjectData, [655](#)
- v
 - arcs::common_interface::CircleParameters, [578](#)
- validatePath
 - RobotAlgorithm (), [344](#)
- variableUpdated
 - RegisterControl (), [102](#)
- Vector3d
 - arcs::common_interface, [556](#)
- Vector3f
 - arcs::common_interface, [556](#)
- Vector4d
 - arcs::common_interface, [556](#)
- Vector4f
 - arcs::common_interface, [557](#)
- Vector6f
 - arcs::common_interface, [557](#)
- velocity
 - arcs::common_interface::GripperStatus, [589](#)
- voltage
 - arcs::common_interface::GripperStatus, [589](#)
- waitSyncTasks
 - SyncMove (), [535](#)
- weaveEnd
 - MotionControl (), [315](#)
- weaveStart
 - MotionControl (), [316](#)
- weaveUpdateParameters
 - MotionControl (), [318](#)
- what
 - arcs::common_interface::AuboException, [573](#)