AUBO SDK

0.26.0

Generated by Doxygen 1.16.1

1 AuboStudio SDK 1

2 Topic Index 13
2.1 Topics . . . 13

3 Directory Hierarchy 15
3.1 Directories . . . 15

4 Namespace Index 17
4.1 Namespace List . . . 17

5 Hierarchical Index 19
5.1 Class Hierarchy . . . 19

6 Class Index 21
6.1 Class List . . . 21

7 File Index 23
7.1 File List . . . 23

8 Topic Documentation 25
8.1 AuboApi ( ) . . . 25
8.1.1 Detailed Description . . . 26
8.1.2 Function Documentation . . . 26
8.1.2.1 getAxisInterface() . . . 26
8.1.2.2 getAxisNames() . . . 26
8.1.2.3 getGripperInterface() . . . 26
8.1.2.4 getMath() . . . 27
8.1.2.5 getRegisterControl() . . . 27
8.1.2.6 getRobotInterface() . . . 27
8.1.2.7 getRobotNames() . . . 28
8.1.2.8 getRuntimeMachine() . . . 29
8.1.2.9 getSerial() . . . 29
8.1.2.10 getSocket() . . . 29
8.1.2.11 getSyncMove() . . . 30
8.1.2.12 getSystemInfo() . . . 30
8.1.2.13 getTrace() . . . 30
8.2 AxisInterface ( ) . . . 30
8.2.1 Detailed Description . . . 32
8.2.2 Function Documentation . . . 32
8.2.2.1 clearAxisError() . . . 32
8.2.2.2 enableExtAxis() . . . 33
8.2.2.3 followAnotherAxis() . . . 33
8.2.2.4 getAxisModeType() . . . 33
8.2.2.5 getErrorCode() . . . 33
8.2.2.6 getExtAxisAcceleration() . . . 34

8.2.2.7 getExtAxisBusCurrent() . . . 34
8.2.2.8 getExtAxisBusVoltage() . . . 34
8.2.2.9 getExtAxisCurrent() . . . 34
8.2.2.10 getExtAxisMaxAcceleration() . . . 34
8.2.2.11 getExtAxisMaxAccelerationLimit() . . . 35
8.2.2.12 getExtAxisMaxPosition() . . . 35
8.2.2.13 getExtAxisMaxPositionLimit() . . . 35
8.2.2.14 getExtAxisMaxVelocity() . . . 35
8.2.2.15 getExtAxisMaxVelocityLimit() . . . 35
8.2.2.16 getExtAxisMountingPose() . . . 36
8.2.2.17 getExtAxisPose() . . . 36
8.2.2.18 getExtAxisPosition() . . . 36
8.2.2.19 getExtAxisTemperature() . . . 36
8.2.2.20 getExtAxisType() . . . 36
8.2.2.21 getExtAxisVelocity() . . . 37
8.2.2.22 getExtMinPosition() . . . 37
8.2.2.23 getExtMinPositionLimit() . . . 37
8.2.2.24 getReductionRatio() . . . 37
8.2.2.25 moveExtJoint() . . . 37
8.2.2.26 poweroffExtAxis() . . . 38
8.2.2.27 poweronExtAxis() . . . 38
8.2.2.28 setExtAxisAcceleration() . . . 38
8.2.2.29 setExtAxisMaxAccelerationLimit() . . . 38
8.2.2.30 setExtAxisMaxPositionLimit() . . . 39
8.2.2.31 setExtAxisMaxVelocityLimit() . . . 39
8.2.2.32 setExtAxisMountingPose() . . . 39
8.2.2.33 setExtAxisType() . . . 40
8.2.2.34 setExtAxisVelocity() . . . 40
8.2.2.35 setExtAxisZero() . . . 40
8.2.2.36 setExtMinPositionLimit() . . . 40
8.2.2.37 setReductionRatio() . . . 41
8.2.2.38 speedExtJoint() . . . 41
8.2.2.39 stopExtJoint() . . . 41
8.2.2.40 stopFollowAnotherAxis() . . . 42
8.3 GripperInterface ( ) . . . 42
8.3.1 Detailed Description . . . 43
8.3.2 Function Documentation . . . 43
8.3.2.1 gripperAdd() . . . 43
8.3.2.2 gripperConnect() . . . 43
8.3.2.3 gripperDelete() . . . 43
8.3.2.4 gripperDisconnect() . . . 43
8.3.2.5 gripperEnable() . . . 43
8.3.2.6 gripperGetHardwareVersion() . . . 43

8.3.2.7 gripperGetNames() . . . 43
8.3.2.8 gripperGetStatusCode() . . . 43
8.3.2.9 gripperGetSupportedModels() . . . 44
8.3.2.10 gripperGetWorkMode() . . . 44
8.3.2.11 gripperIsConnected() . . . 44
8.3.2.12 gripperIsEnabled() . . . 44
8.3.2.13 gripperMove() . . . 44
8.3.2.14 gripperRename() . . . 44
8.3.2.15 gripperResetSlaveId() . . . 44
8.3.2.16 gripperScanDevices() . . . 44
8.3.2.17 gripperSetMountPose() . . . 45
8.3.2.18 gripperSetPosition() . . . 45
8.3.2.19 gripperSetWorkMode() . . . 45
8.3.2.20 gripperStop() . . . 45
8.4 Math ( ) . . . 45
8.4.1 Detailed Description . . . 46
8.4.2 Function Documentation . . . 46
8.4.2.1 calculateCircleFourthPoint() . . . 46
8.4.2.2 calibrateCoordinate() . . . 47
8.4.2.3 changePoseWithXYRef() . . . 48
8.4.2.4 deltaPoseAdd() . . . 49
8.4.2.5 deltaPoseTrans() . . . 49
8.4.2.6 forceTrans() . . . 50
8.4.2.7 getDeltaPoseBySensorDistance() . . . 50
8.4.2.8 homMatrixToPose() . . . 51
8.4.2.9 interpolatePose() . . . 51
8.4.2.10 poseAdd() . . . 52
8.4.2.11 poseAngleDistance() . . . 53
8.4.2.12 poseDistance() . . . 53
8.4.2.13 poseEqual() . . . 54
8.4.2.14 poseInverse() . . . 55
8.4.2.15 poseRotation() . . . 56
8.4.2.16 poseSub() . . . 56
8.4.2.17 poseToHomMatrix() . . . 57
8.4.2.18 poseTrans() . . . 58
8.4.2.19 poseTransInv() . . . 59
8.4.2.20 quaternionToRpy() . . . 59
8.4.2.21 rpyToQuaternion() . . . 60
8.4.2.22 tcpOffsetIdentify() . . . 61
8.4.2.23 transferRefFrame() . . . 62
8.5 RegisterControl ( ) . . . 62
8.5.1 Detailed Description . . . 65
8.5.2 Function Documentation . . . 66

8.5.2.1 addFixedModbusGroup() . . . 66
8.5.2.2 addInt32RegEncoder() . . . 67
8.5.2.3 addModbusEncoder() . . . 67
8.5.2.4 clearNamedVariable() . . . 68
8.5.2.5 deleteVirtualEncoder() . . . 68
8.5.2.6 getBool() . . . 69
8.5.2.7 getBoolInput() . . . 69
8.5.2.8 getBoolOutput() . . . 70
8.5.2.9 getDouble() . . . 71
8.5.2.10 getDoubleInput() . . . 72
8.5.2.11 getDoubleOutput() . . . 73
8.5.2.12 getFloat() . . . 74
8.5.2.13 getFloatInput() . . . 74
8.5.2.14 getFloatOutput() . . . 75
8.5.2.15 getInt16Register() . . . 76
8.5.2.16 getInt16RegisterBit() . . . 77
8.5.2.17 getInt32() . . . 78
8.5.2.18 getInt32Input() . . . 79
8.5.2.19 getInt32Output() . . . 79
8.5.2.20 getModbusDeviceStatus() . . . 80
8.5.2.21 getNamedVariableType() . . . 81
8.5.2.22 getString() . . . 82
8.5.2.23 getVecChar() . . . 82
8.5.2.24 getVecDouble() . . . 83
8.5.2.25 getVecFloat() . . . 84
8.5.2.26 getVecInt32() . . . 85
8.5.2.27 getWatchDogAction() . . . 86
8.5.2.28 getWatchDogTimeout() . . . 86
8.5.2.29 hasNamedVariable() . . . 86
8.5.2.30 modbusAddSignal() . . . 87
8.5.2.31 modbusDeleteAllSignals() . . . 89
8.5.2.32 modbusDeleteSignal() . . . 89
8.5.2.33 modbusGetSignalError() . . . 90
8.5.2.34 modbusGetSignalErrors() . . . 90
8.5.2.35 modbusGetSignalIndex() . . . 90
8.5.2.36 modbusGetSignalNames() . . . 91
8.5.2.37 modbusGetSignalStatus() . . . 91
8.5.2.38 modbusGetSignalTypes() . . . 92
8.5.2.39 modbusGetSignalValues() . . . 93
8.5.2.40 modbusSendCustomCommand() . . . 93
8.5.2.41 modbusSetDigitalInputAction() . . . 94
8.5.2.42 modbusSetOutputRunstate() . . . 95
8.5.2.43 modbusSetOutputRunstate1() . . . 96

8.5.2.44 modbusSetOutputSignal() . . . 97
8.5.2.45 modbusSetOutputSignal1() . . . 97
8.5.2.46 modbusSetOutputSignalPulse() . . . 98
8.5.2.47 modbusSetSignalUpdateFrequency() . . . 99
8.5.2.48 setBool() . . . 100
8.5.2.49 setBoolInput() . . . 101
8.5.2.50 setBoolOutput() . . . 102
8.5.2.51 setDouble() . . . 103
8.5.2.52 setDoubleInput() . . . 103
8.5.2.53 setDoubleOutput() . . . 104
8.5.2.54 setFloat() . . . 105
8.5.2.55 setFloatInput() . . . 106
8.5.2.56 setFloatOutput() . . . 107
8.5.2.57 setInt16Register() . . . 108
8.5.2.58 setInt16RegisterBit() . . . 108
8.5.2.59 setInt32() . . . 109
8.5.2.60 setInt32Input() . . . 110
8.5.2.61 setInt32Output() . . . 111
8.5.2.62 setString() . . . 112
8.5.2.63 setVecChar() . . . 112
8.5.2.64 setVecDouble() . . . 113
8.5.2.65 setVecFloat() . . . 114
8.5.2.66 setVecInt32() . . . 115
8.5.2.67 setWatchDog() . . . 116
8.5.2.68 variableUpdated() . . . 116
8.6 RobotInterface( ) . . . 117
8.6.1 Detailed Description . . . 118
8.6.2 Function Documentation . . . 118
8.6.2.1 getForceControl() . . . 118
8.6.2.2 getIoControl() . . . 119
8.6.2.3 getMotionControl() . . . 119
8.6.2.4 getRobotAlgorithm() . . . 119
8.6.2.5 getRobotConfig() . . . 120
8.6.2.6 getRobotManage() . . . 120
8.6.2.7 getRobotState() . . . 120
8.6.2.8 getSyncMove() . . . 121
8.6.2.9 getTrace() . . . 121
8.6.3 ForceControl( ) . . . 121
8.6.3.1 Detailed Description . . . 124
8.6.3.2 Function Documentation . . . 124
8.6.4 IoControl (IO) . . . 158
8.6.4.1 Detailed Description . . . 163
8.6.4.2 Function Documentation . . . 163

8.6.5 MotionControl ( ) . . . 246
8.6.5.1 Detailed Description . . . 252
8.6.5.2 Function Documentation . . . 252
8.6.6 RobotAlgorithm ( ) . . . 349
8.6.6.1 Detailed Description . . . 351
8.6.6.2 Function Documentation . . . 352
8.6.7 RobotConfig ( ) . . . 381
8.6.7.1 Detailed Description . . . 384
8.6.7.2 Function Documentation . . . 384
8.6.8 RobotManage ( ) . . . 439
8.6.8.1 Detailed Description . . . 441
8.6.8.2 Function Documentation . . . 441
8.6.9 RobotState ( ) . . . 467
8.6.9.1 Detailed Description . . . 470
8.6.9.2 Function Documentation . . . 470
8.7 RuntimeMachine ( ) . . . 521
8.7.1 Detailed Description . . . 523
8.7.2 Function Documentation . . . 523
8.7.2.1 abort() . . . 523
8.7.2.2 arbitraryResume() . . . 524
8.7.2.3 clearBreakPoints() . . . 524
8.7.2.4 clearPreloadPrograms() . . . 525
8.7.2.5 deleteTask() . . . 525
8.7.2.6 detachTask() . . . 525
8.7.2.7 enterCritical() . . . 525
8.7.2.8 exitCritical() . . . 526
8.7.2.9 getAdvancePlanContext() . . . 527
8.7.2.10 getAdvancePtr() . . . 527
8.7.2.11 getExecutionStatus() . . . 528
8.7.2.12 getInterpPtr() . . . 528
8.7.2.13 getMainPtr() . . . 528
8.7.2.14 getPlanContext() . . . 529
8.7.2.15 getPreloadProgram() . . . 529
8.7.2.16 getStatus() . . . 530
8.7.2.17 getTaskQueueSize() . . . 530
8.7.2.18 getTimer() . . . 531
8.7.2.19 getTriggInterrupts() . . . 532
8.7.2.20 gotoLine() . . . 532
8.7.2.21 isTaskAlive() . . . 532
8.7.2.22 loadProgram() . . . 533
8.7.2.23 newTask() . . . 534
8.7.2.24 nop() . . . 534
8.7.2.25 pause() . . . 534

8.7.2.26 preloadProgram() . . . 535
8.7.2.27 removeBreakPoint() . . . 535
8.7.2.28 resume() . . . 536
8.7.2.29 runProgram() . . . 536
8.7.2.30 setBreakPoint() . . . 536
8.7.2.31 setLabel() . . . 537
8.7.2.32 setPlanContext() . . . 538
8.7.2.33 setResumeWait() . . . 538
8.7.2.34 start() . . . 539
8.7.2.35 step() . . . 540
8.7.2.36 stop() . . . 540
8.7.2.37 switchTask() . . . 540
8.7.2.38 timerDelete() . . . 541
8.7.2.39 timerReset() . . . 541
8.7.2.40 timerStart() . . . 542
8.7.2.41 timerStop() . . . 543
8.7.2.42 triggBegin() . . . 543
8.7.2.43 triggEnd() . . . 544
8.7.2.44 triggInterrupt() . . . 544
8.8 Serial ( ) . . . 545
8.8.1 Detailed Description . . . 546
8.8.2 Function Documentation . . . 546
8.8.2.1 serialClose() . . . 546
8.8.2.2 serialOpen() . . . 546
8.8.2.3 serialReadByte() . . . 547
8.8.2.4 serialReadByteList() . . . 548
8.8.2.5 serialReadString() . . . 548
8.8.2.6 serialSendAllString() . . . 549
8.8.2.7 serialSendByte() . . . 550
8.8.2.8 serialSendInt() . . . 550
8.8.2.9 serialSendLine() . . . 551
8.8.2.10 serialSendString() . . . 551
8.9 Socket (socket) . . . 552
8.9.1 Detailed Description . . . 553
8.9.2 Function Documentation . . . 553
8.9.2.1 socketClose() . . . 553
8.9.2.2 socketHasConnected() . . . 554
8.9.2.3 socketOpen() . . . 554
8.9.2.4 socketReadAllString() . . . 555
8.9.2.5 socketReadAsciiFloat() . . . 556
8.9.2.6 socketReadBinaryInteger() . . . 557
8.9.2.7 socketReadByteList() . . . 557
8.9.2.8 socketReadString() . . . 558

8.9.2.9 socketSendAllString() . . . 559
8.9.2.10 socketSendByte() . . . 560
8.9.2.11 socketSendInt() . . . 561
8.9.2.12 socketSendLine() . . . 561
8.9.2.13 socketSendString() . . . 562
8.10 SyncMove ( ) . . . 563
8.10.1 Detailed Description . . . 565
8.10.2 Function Documentation . . . 565
8.10.2.1 axisGroupAdd() . . . 565
8.10.2.2 axisGroupAddAxis() . . . 565
8.10.2.3 axisGroupDelete() . . . 566
8.10.2.4 axisGroupGetActualPositions() . . . 566
8.10.2.5 axisGroupGetAxisIndex() . . . 567
8.10.2.6 axisGroupGetAxisName() . . . 567
8.10.2.7 axisGroupGetTargetPositions() . . . 567
8.10.2.8 axisGroupMoveJoint() . . . 568
8.10.2.9 axisGroupOffsetPositions() . . . 568
8.10.2.10 axisGroupSpeedJoint() . . . 569
8.10.2.11 axisGroupUpdateAxis() . . . 569
8.10.2.12 frameAdd() . . . 570
8.10.2.13 frameAttach() . . . 570
8.10.2.14 frameConvertPose() . . . 571
8.10.2.15 frameDelete() . . . 571
8.10.2.16 frameDeleteAll() . . . 572
8.10.2.17 frameExist() . . . 572
8.10.2.18 frameGetChildren() . . . 572
8.10.2.19 frameGetParent() . . . 573
8.10.2.20 frameGetPose() . . . 573
8.10.2.21 frameMove() . . . 574
8.10.2.22 isSyncMoveOn() . . . 575
8.10.2.23 syncMoveOff() . . . 575
8.10.2.24 syncMoveOn() . . . 575
8.10.2.25 syncMoveResume() . . . 576
8.10.2.26 syncMoveSegment() . . . 576
8.10.2.27 syncMoveSuspend() . . . 577
8.10.2.28 syncMoveUndo() . . . 577
8.10.2.29 waitSyncTasks() . . . 578
8.11 SystemInfo ( ) . . . 578
8.11.1 Detailed Description . . . 579
8.11.2 Function Documentation . . . 579
8.11.2.1 getControlSoftwareBuildDate() . . . 579
8.11.2.2 getControlSoftwareFullVersion() . . . 580
8.11.2.3 getControlSoftwareVersionCode() . . . 580

8.11.2.4 getControlSoftwareVersionHash() . . . 581
8.11.2.5 getControlSystemTime() . . . 581
8.11.2.6 getInterfaceVersionCode() . . . 582
8.12 Trace ( ) . . . 582
8.12.1 Detailed Description . . . 583
8.12.2 Function Documentation . . . 583
8.12.2.1 alarm() . . . 583
8.12.2.2 peek() . . . 583
8.12.2.3 popup() . . . 584
8.12.2.4 textmsg() . . . 585

9 Directory Documentation 587
9.1 include/aubo Directory Reference . . . 587
9.2 doc Directory Reference . . . 588
9.3 include/aubo/error_stack Directory Reference . . . 588
9.4 include Directory Reference . . . 589
9.5 include/aubo/robot Directory Reference . . . 589

10 Namespace Documentation 591
10.1 arcs Namespace Reference . . . 591
10.2 arcs::common_interface Namespace Reference . . . 591
10.2.1 Typedef Documentation . . . 594
10.2.1.1 AuboApiPtr . . . 594
10.2.1.2 AxisInterfacePtr . . . 594
10.2.1.3 Box . . . 594
10.2.1.4 Cylinder . . . 594
10.2.1.5 DynamicsModel . . . 594
10.2.1.6 ForceControlPtr . . . 595
10.2.1.7 ForceSensorCalibResult . . . 595
10.2.1.8 ForceSensorCalibResultWithError . . . 595
10.2.1.9 GripperInterfacePtr . . . 595
10.2.1.10 GripperStatusVector . . . 595
10.2.1.11 IoControlPtr . . . 595
10.2.1.12 MathPtr . . . 595
10.2.1.13 MotionControlPtr . . . 596
10.2.1.14 Payload . . . 596
10.2.1.15 RegisterControlPtr . . . 596
10.2.1.16 ResultWithErrno . . . 596
10.2.1.17 ResultWithErrno1 . . . 596
10.2.1.18 ResultWithErrno2 . . . 596
10.2.1.19 ResultWithErrno3 . . . 596
10.2.1.20 RobotAlgorithmPtr . . . 597
10.2.1.21 RobotConfigPtr . . . 597
10.2.1.22 RobotInterfacePtr . . . 597

10.2.1.23 RobotManagePtr . . . 597
10.2.1.24 RobotMsgVector . . . 597
10.2.1.25 RobotStatePtr . . . 597
10.2.1.26 RuntimeMachinePtr . . . 597
10.2.1.27 SerialPtr . . . 597
10.2.1.28 SocketPtr . . . 598
10.2.1.29 Sphere . . . 598
10.2.1.30 SyncMovePtr . . . 598
10.2.1.31 SystemInfoPtr . . . 598
10.2.1.32 TaskSet . . . 598
10.2.1.33 TracePtr . . . 598
10.2.1.34 Vector3d . . . 598
10.2.1.35 Vector3f . . . 598
10.2.1.36 Vector4d . . . 599
10.2.1.37 Vector4f . . . 599
10.2.1.38 Vector6f . . . 599
10.2.2 Enumeration Type Documentation . . . 599
10.2.2.1 AuboErrorCodes . . . 599
10.2.2.2 AxisModeType . . . 599
10.2.2.3 EnvelopingShape . . . 599
10.2.2.4 error_type . . . 600
10.2.2.5 ExceptionCode . . . 600
10.2.2.6 ForceControlState . . . 601
10.2.2.7 HandleModeType . . . 601
10.2.2.8 HandleStateType . . . 601
10.2.2.9 JointServoModeType . . . 602
10.2.2.10 JointStateType . . . 602
10.2.2.11 OperationalModeType . . . 602
10.2.2.12 PathBufferType . . . 603
10.2.2.13 PayloadIdentifyMoveAxis . . . 603
10.2.2.14 RefFrameType . . . 603
10.2.2.15 RobotControlModeType . . . 604
10.2.2.16 RobotEmergencyStopType . . . 604
10.2.2.17 RobotModeType . . . 604
10.2.2.18 RuntimeState . . . 605
10.2.2.19 SafeguedStopType . . . 605
10.2.2.20 SafetyInputAction . . . 605
10.2.2.21 SafetyModeType . . . 605
10.2.2.22 SafetyOutputRunState . . . 606
10.2.2.23 StandardInputAction . . . 606
10.2.2.24 StandardOutputRunState . . . 606
10.2.2.25 TaskFrameType . . . 606
10.2.2.26 TraceLevel . . . 607

10.2.3 Function Documentation . . . 607
10.2.3.1 operator<<() [1/7] . . . 607
10.2.3.2 operator<<() [2/7] . . . 607
10.2.3.3 operator<<() [3/7] . . . 607
10.2.3.4 operator<<() [4/7] . . . 608
10.2.3.5 operator<<() [5/7] . . . 608
10.2.3.6 operator<<() [6/7] . . . 608
10.2.3.7 operator<<() [7/7] . . . 608
10.2.3.8 returnValue2Str() . . . 608
10.3 arcs::error_stack Namespace Reference . . . 608
10.3.1 Enumeration Type Documentation . . . 608
10.3.1.1 ErrorCodes . . . 608
10.3.2 Function Documentation . . . 609
10.3.2.1 codeCompose() . . . 609
10.3.2.2 dump() . . . 609
10.3.2.3 errorCode2Str() . . . 609
10.3.2.4 mod() . . . 609
10.3.2.5 str2ErrorCode() . . . 609

11 Class Documentation 611
11.1 arcs::common_interface::AuboApi Class Reference . . . 611
11.1.1 Detailed Description . . . 612
11.1.2 Constructor & Destructor Documentation . . . 612
11.1.2.1 AuboApi() . . . 612
11.1.2.2 ~AuboApi() . . . 612
11.1.3 Member Data Documentation . . . 612
11.1.3.1 d_ . . . 612
11.2 arcs::common_interface::AuboException Class Reference . . . 613
11.2.1 Detailed Description . . . 614
11.2.2 Constructor & Destructor Documentation . . . 614
11.2.2.1 AuboException() [1/2] . . . 614
11.2.2.2 AuboException() [2/2] . . . 614
11.2.3 Member Function Documentation . . . 615
11.2.3.1 code() . . . 615
11.2.3.2 type() . . . 615
11.2.3.3 what() . . . 615
11.2.4 Member Data Documentation . . . 615
11.2.4.1 code_ . . . 615
11.2.4.2 message_ . . . 616
11.3 arcs::common_interface::AxisInterface Class Reference . . . 616
11.3.1 Detailed Description . . . 618
11.3.2 Constructor & Destructor Documentation . . . 618
11.3.2.1 AxisInterface() . . . 618

11.3.2.2 ~AxisInterface() . . . 618
11.3.3 Member Data Documentation . . . 618
11.3.3.1 d_ . . . 618
11.4 arcs::common_interface::CircleParameters Struct Reference . . . 618
11.4.1 Detailed Description . . . 619
11.4.2 Member Data Documentation . . . 619
11.4.2.1 a . . . 619
11.4.2.2 blend_radius . . . 619
11.4.2.3 direction . . . 619
11.4.2.4 duration . . . 620
11.4.2.5 helix . . . 620
11.4.2.6 loop_times . . . 620
11.4.2.7 pose_to . . . 620
11.4.2.8 pose_via . . . 620
11.4.2.9 spiral . . . 620
11.4.2.10 v . . . 621
11.5 arcs::common_interface::Enveloping Struct Reference . . . 621
11.5.1 Detailed Description . . . 621
11.5.2 Member Data Documentation . . . 622
11.5.2.1 ep_args . . . 622
11.5.2.2 shape . . . 622
11.5.2.3 stl_path . . . 622
11.6 arcs::common_interface::ForceControl Class Reference . . . 622
11.6.1 Detailed Description . . . 624
11.6.2 Constructor & Destructor Documentation . . . 624
11.6.2.1 ForceControl() . . . 624
11.6.2.2 ~ForceControl() . . . 625
11.6.3 Member Data Documentation . . . 625
11.6.3.1 d_ . . . 625
11.7 arcs::common_interface::GripperInterface Class Reference . . . 625
11.7.1 Detailed Description . . . 626
11.7.2 Constructor & Destructor Documentation . . . 626
11.7.2.1 GripperInterface() . . . 626
11.7.2.2 ~GripperInterface() . . . 626
11.7.3 Member Function Documentation . . . 626
11.7.3.1 gripperGetAngle() . . . 626
11.7.3.2 gripperGetForce() . . . 626
11.7.3.3 gripperGetMotionState() . . . 626
11.7.3.4 gripperGetMountPose() . . . 626
11.7.3.5 gripperGetObjectDetection() . . . 627
11.7.3.6 gripperGetPosition() . . . 627
11.7.3.7 gripperGetRVelocity() . . . 627
11.7.3.8 gripperGetSoftwareVersion() . . . 627

11.7.3.9 gripperGetTemperature() . . . 627
11.7.3.10 gripperGetTorque() . . . 627
11.7.3.11 gripperGetVelocity() . . . 627
11.7.3.12 gripperGetVoltage() . . . 628
11.7.3.13 gripperSetAngle() . . . 628
11.7.3.14 gripperSetForce() . . . 628
11.7.3.15 gripperSetRVelocity() . . . 628
11.7.3.16 gripperSetTorque() . . . 628
11.7.3.17 gripperSetVelocity() . . . 628
11.7.4 Member Data Documentation . . . 628
11.7.4.1 d_ . . . 628
11.8 arcs::common_interface::GripperStatus Struct Reference . . . 629
11.8.1 Detailed Description . . . 629
11.8.2 Member Data Documentation . . . 630
11.8.2.1 angle . . . 630
11.8.2.2 force . . . 630
11.8.2.3 is_connected . . . 630
11.8.2.4 is_enabled . . . 630
11.8.2.5 motion_state . . . 630
11.8.2.6 name . . . 630
11.8.2.7 object_detection . . . 630
11.8.2.8 position . . . 631
11.8.2.9 r_velocity . . . 631
11.8.2.10 status_code . . . 631
11.8.2.11 temperature . . . 631
11.8.2.12 torque . . . 631
11.8.2.13 velocity . . . 631
11.8.2.14 voltage . . . 631
11.9 arcs::common_interface::IoControl Class Reference . . . 632
11.9.1 Detailed Description . . . 636
11.9.2 Constructor & Destructor Documentation . . . 636
11.9.2.1 IoControl() . . . 636
11.9.2.2 ~IoControl() . . . 636
11.9.3 Member Data Documentation . . . 636
11.9.3.1 d_ . . . 636
11.10 arcs::common_interface::Math Class Reference . . . 636
11.10.1 Detailed Description . . . 638
11.10.2 Constructor & Destructor Documentation . . . 638
11.10.2.1 Math() . . . 638
11.10.2.2 ~Math() . . . 638
11.10.3 Member Data Documentation . . . 638
11.10.3.1 d_ . . . 638
11.11 arcs::common_interface::MotionControl Class Reference . . . 638

11.11.1 Detailed Description . . . 644
11.11.2 Constructor & Destructor Documentation . . . 644
11.11.2.1 MotionControl() . . . 644
11.11.2.2 ~MotionControl() . . . 644
11.11.3 Member Function Documentation . . . 644
11.11.3.1 getConveyorTrackNextItem() . . . 644
11.11.3.2 isConveyorTrackExceed() . . . 645
11.11.3.3 servoCartesianWithAxisGroup() . . . 646
11.11.3.4 servoJointWithAxisGroup() . . . 646
11.11.4 Member Data Documentation . . . 646
11.11.4.1 d_ . . . 646
11.12 arcs::common_interface::RegisterControl Class Reference . . . 647
11.12.1 Detailed Description . . . 650
11.12.2 Constructor & Destructor Documentation . . . 650
11.12.2.1 RegisterControl() . . . 650
11.12.2.2 ~RegisterControl() . . . 650
11.12.3 Member Function Documentation . . . 650
11.12.3.1 modbusSetOutputSignalWithTimeout() . . . 650
11.12.4 Member Data Documentation . . . 651
11.12.4.1 d_ . . . 651
11.13 arcs::common_interface::RobotAlgorithm Class Reference . . . 651
11.13.1 Detailed Description . . . 653
11.13.2 Constructor & Destructor Documentation . . . 653
11.13.2.1 RobotAlgorithm() . . . 653
11.13.2.2 ~RobotAlgorithm() . . . 654
11.13.3 Member Function Documentation . . . 654
11.13.3.1 pathMovej() . . . 654
11.13.4 Member Data Documentation . . . 655
11.13.4.1 d_ . . . 655
11.14 arcs::common_interface::RobotConfig Class Reference . . . 655
11.14.1 Detailed Description . . . 658
11.14.2 Constructor & Destructor Documentation . . . 658
11.14.2.1 RobotConfig() . . . 658
11.14.2.2 ~RobotConfig() . . . 659
11.14.3 Member Function Documentation . . . 659
11.14.3.1 disableAxisGroup() . . . 659
11.14.3.2 enableAxisGroup() . . . 659
11.14.3.3 getRobotBaseParent() . . . 659
11.14.4 Member Data Documentation . . . 659
11.14.4.1 d_ . . . 659
11.15 arcs::common_interface::RobotInterface Class Reference . . . 659
11.15.1 Detailed Description . . . 660
11.15.2 Constructor & Destructor Documentation . . . 660

11.15.2.1 RobotInterface() . . . 660
11.15.2.2 ~RobotInterface() . . . 660
11.15.3 Member Data Documentation . . . 660
11.15.3.1 d_ . . . 660
11.16 arcs::common_interface::RobotManage Class Reference . . . 661
11.16.1 Detailed Description . . . 662
11.16.2 Constructor & Destructor Documentation . . . 662
11.16.2.1 RobotManage() . . . 662
11.16.2.2 ~RobotManage() . . . 663
11.16.3 Member Data Documentation . . . 663
11.16.3.1 d_ . . . 663
11.17 arcs::common_interface::RobotMsg Struct Reference . . . 663
11.17.1 Detailed Description . . . 664
11.17.2 Member Data Documentation . . . 664
11.17.2.1 args . . . 664
11.17.2.2 code . . . 664
11.17.2.3 level . . . 664
11.17.2.4 source . . . 664
11.17.2.5 timestamp . . . 665
11.18 arcs::common_interface::RobotSafetyParameterRange Struct Reference . . . 665
11.18.1 Detailed Description . . . 667
11.18.2 Constructor & Destructor Documentation . . . 667
11.18.2.1 RobotSafetyParameterRange() . . . 667
11.18.3 Member Data Documentation . . . 667
11.18.3.1 allow_manual_high_speed . . . 667
11.18.3.2 crc32 . . . 668
11.18.3.3 [struct] . . . 668
11.18.3.4 elbow_force . . . 668
11.18.3.5 elbow_speed . . . 668
11.18.3.6 joint_torque . . . 668
11.18.3.7 momentum . . . 668
11.18.3.8 orig . . . 668
11.18.3.9 [struct] . . . 669
11.18.3.10 plane . . . 669
11.18.3.11 planes . . . 669
11.18.3.12 power . . . 669
11.18.3.13 qdmax . . . 669
11.18.3.14 qmax . . . 669
11.18.3.15 qmin . . . 669
11.18.3.16 reduced_entry_distance . . . 670
11.18.3.17 reduced_entry_time . . . 670
11.18.3.18 restrict_elbow . . . 670
11.18.3.19 safety_home . . . 670

11.18.3.20 safety_input_auto_safeguard_reset . . . 670
11.18.3.21 safety_input_auto_safeguard_stop . . . 670
11.18.3.22 safety_input_emergency_stop . . . 671
11.18.3.23 safety_input_handguide . . . 671
11.18.3.24 safety_input_operational_mode . . . 671
11.18.3.25 safety_input_reduced_mode . . . 671
11.18.3.26 safety_input_safeguard_reset . . . 671
11.18.3.27 safety_input_safeguard_stop . . . 671
11.18.3.28 safety_input_three_position_switch . . . 672
11.18.3.29 safety_output_emergency_stop . . . 672
11.18.3.30 safety_output_not_emergency_stop . . . 672
11.18.3.31 safety_output_not_reduced_mode . . . 672
11.18.3.32 safety_output_reduced_mode . . . 672
11.18.3.33 safety_output_robot_moving . . . 672
11.18.3.34 safety_output_robot_not_stopping . . . 673
11.18.3.35 safety_output_robot_steady . . . 673
11.18.3.36 safety_output_safe_home . . . 673
11.18.3.37 safety_output_safetyguard_stop . . . 673
11.18.3.38 size . . . 673
11.18.3.39 stop_distance . . . 673
11.18.3.40 stop_time . . . 674
11.18.3.41 tcp_force . . . 674
11.18.3.42 tcp_speed . . . 674
11.18.3.43 tool_azimuth . . . 674
11.18.3.44 tool_deviation . . . 674
11.18.3.45 tool_inclination . . . 674
11.18.3.46 tool_orientation . . . 675
11.18.3.47 tools . . . 675
11.18.3.48 tp_3pe_for_handguide . . . 675
11.18.3.49 [struct] . . . 675
11.19 arcs::common_interface::RobotState Class Reference . . . 675
11.19.1 Detailed Description . . . 679
11.19.2 Constructor & Destructor Documentation . . . 679
11.19.2.1 RobotState() . . . 679
11.19.2.2 ~RobotState() . . . 679
11.19.3 Member Function Documentation . . . 679
11.19.3.1 getBaseForceSensor() . . . 679
11.19.4 Member Data Documentation . . . 680
11.19.4.1 d_ . . . 680
11.20 arcs::common_interface::RtdeRecipe Struct Reference . . . 680
11.20.1 Detailed Description . . . 681
11.20.2 Member Data Documentation . . . 681
11.20.2.1 chanel . . . 681

11.20.2.2 frequency . . . 681
11.20.2.3 segments . . . 681
11.20.2.4 to_server . . . 682
11.20.2.5 trigger . . . 682
11.21 arcs::common_interface::RuntimeMachine Class Reference . . . 682
11.21.1 Detailed Description . . . 684
11.21.2 Constructor & Destructor Documentation . . . 684
11.21.2.1 RuntimeMachine() . . . 684
11.21.2.2 ~RuntimeMachine() . . . 684
11.21.3 Member Function Documentation . . . 684
11.21.3.1 getExecutionStatus1() . . . 684
11.21.3.2 getRuntimeState() . . . 684
11.21.4 Member Data Documentation . . . 685
11.21.4.1 d_ . . . 685
11.22 arcs::common_interface::Serial Class Reference . . . 685
11.22.1 Detailed Description . . . 686
11.22.2 Constructor & Destructor Documentation . . . 686
11.22.2.1 Serial() . . . 686
11.22.2.2 ~Serial() . . . 686
11.22.3 Member Data Documentation . . . 686
11.22.3.1 d_ . . . 686
11.23 arcs::common_interface::Socket Class Reference . . . 686
11.23.1 Detailed Description . . . 687
11.23.2 Constructor & Destructor Documentation . . . 687
11.23.2.1 Socket() . . . 687
11.23.2.2 ~Socket() . . . 687
11.23.3 Member Data Documentation . . . 687
11.23.3.1 d_ . . . 687
11.24 arcs::common_interface::SpiralParameters Struct Reference . . . 688
11.24.1 Detailed Description . . . 688
11.24.2 Member Data Documentation . . . 688
11.24.2.1 angle . . . 688
11.24.2.2 frame . . . 689
11.24.2.3 helix . . . 689
11.24.2.4 plane . . . 689
11.24.2.5 spiral . . . 689
11.25 arcs::common_interface::SyncMove Class Reference . . . 689
11.25.1 Detailed Description . . . 691
11.25.2 Constructor & Destructor Documentation . . . 691
11.25.2.1 SyncMove() . . . 691
11.25.2.2 ~SyncMove() . . . 691
11.25.3 Member Data Documentation . . . 692
11.25.3.1 d_ . . . 692

11.26 arcs::common_interface::SystemInfo Class Reference . . . 692
11.26.1 Detailed Description . . . 692
11.26.2 Constructor & Destructor Documentation . . . 692
11.26.2.1 SystemInfo() . . . 692
11.26.2.2 ~SystemInfo() . . . 693
11.26.3 Member Data Documentation . . . 693
11.26.3.1 d_ . . . 693
11.27 arcs::common_interface::Trace Class Reference . . . 693
11.27.1 Detailed Description . . . 693
11.27.2 Constructor & Destructor Documentation . . . 694
11.27.2.1 Trace() . . . 694
11.27.2.2 ~Trace() . . . 694
11.27.3 Member Function Documentation . . . 694
11.27.3.1 notify() . . . 694
11.27.4 Member Data Documentation . . . 694
11.27.4.1 d_ . . . 694
11.28 arcs::common_interface::TrajConfig Struct Reference . . . 695
11.28.1 Detailed Description . . . 696
11.28.2 Member Data Documentation . . . 696
11.28.2.1 envelopings . . . 696
11.28.2.2 init_joint . . . 696
11.28.2.3 lower_joint_bound . . . 696
11.28.2.4 max_acceleration . . . 696
11.28.2.5 max_velocity . . . 696
11.28.2.6 move_axis . . . 696
11.28.2.7 upper_joint_bound . . . 697
11.29 arcs::common_interface::VibrationRecalibrationParameter Struct Reference . . . 697
11.29.1 Detailed Description . . . 698
11.29.2 Member Data Documentation . . . 698
11.29.2.1 cog . . . 698
11.29.2.2 inertia . . . 698
11.29.2.3 mass . . . 698
11.29.2.4 points . . . 698
11.29.2.5 stiff_param . . . 698
11.29.2.6 stiff_section . . . 698
11.30 arcs::common_interface::WObjectData Struct Reference . . . 699
11.30.1 Detailed Description . . . 699
11.30.2 Member Data Documentation . . . 699
11.30.2.1 attach_frame . . . 699
11.30.2.2 obj_coord . . . 700
11.30.2.3 remote_tool . . . 700
11.30.2.4 user_coord . . . 700

12 File Documentation 701
12.1 doc/sdk_overview_en.md File Reference . . . 701
12.2 include/aubo/aubo_api.h File Reference . . . 701
12.2.1 Detailed Description . . . 702
12.3 aubo_api.h . . . 702
12.4 include/aubo/axis_interface.h File Reference . . . 706
12.5 axis_interface.h . . . 707
12.6 include/aubo/error_stack/error_stack.h File Reference . . . 715
12.6.1 Detailed Description . . . 716
12.6.2 Macro Definition Documentation . . . 716
12.6.2.1 _D [1/6] . . . 716
12.6.2.2 _D [2/6] . . . 717
12.6.2.3 _D [3/6] . . . 717
12.6.2.4 _D [4/6] . . . 717
12.6.2.5 _D [5/6] . . . 717
12.6.2.6 _D [6/6] . . . 718
12.6.2.7 _PH1_ . . . 718
12.6.2.8 _PH2_ . . . 718
12.6.2.9 _PH3_ . . . 718
12.6.2.10 _PH4_ . . . 718
12.6.2.11 ARCS_ERROR_CODES . . . 718
12.6.2.12 AUBO_SDK_HAL_ERROR_H . . . 719
12.6.2.13 AUBO_SDK_RTM_ERROR_H . . . 719
12.6.2.14 AUBO_SDK_SYSTEM_ERROR_H . . . 719
12.6.2.15 EX_PEDSTRAL_ERRORS . . . 719
12.6.2.16 EX_TOOL_ERRORS . . . 719
12.6.2.17 EXT_AXIS_ERRORS . . . 719
12.6.2.18 HAL_ERRORS . . . 720
12.6.2.19 HARDWARE_INTERFACE_ERRORS . . . 720
12.6.2.20 JOINT_ERRORS . . . 720
12.6.2.21 JOINT_EX_ERRORS . . . 720
12.6.2.22 PEDSTRAL_ERRORS . . . 720
12.6.2.23 RTM_ERRORS . . . 720
12.6.2.24 SAFETY_INTERFACE_BOARD_ERRORS . . . 721
12.6.2.25 SYSTEM_ERRORS . . . 721
12.6.2.26 TOOL_ERRORS . . . 721
12.7 error_stack.h . . . 722
12.8 include/aubo/error_stack/hal_error.h File Reference . . . 723
12.8.1 Detailed Description . . . 724
12.8.2 Macro Definition Documentation . . . 724
12.8.2.1 EX_PEDSTRAL_ERRORS . . . 724
12.8.2.2 EX_TOOL_ERRORS . . . 724
12.8.2.3 EXT_AXIS_ERRORS . . . 724

12.8.2.4 HAL_ERRORS . . . 725
12.8.2.5 HARDWARE_INTERFACE_ERRORS . . . 725
12.8.2.6 JOINT_ERRORS . . . 725
12.8.2.7 JOINT_EX_ERRORS . . . 725
12.8.2.8 PEDSTRAL_ERRORS . . . 725
12.8.2.9 SAFETY_INTERFACE_BOARD_ERRORS . . . 726
12.8.2.10 TOOL_ERRORS . . . 726
12.9 hal_error.h . . . 727
12.10 include/aubo/error_stack/rtm_error.h File Reference . . . 733
12.10.1 Detailed Description . . . 733
12.10.2 Macro Definition Documentation . . . 733
12.10.2.1 RTM_ERRORS . . . 733
12.11 rtm_error.h . . . 734
12.12 include/aubo/error_stack/system_error.h File Reference . . . 737
12.12.1 Detailed Description . . . 738
12.12.2 Macro Definition Documentation . . . 738
12.12.2.1 SYSTEM_ERRORS . . . 738
12.13 system_error.h . . . 738
12.14 include/aubo/gripper_interface.h File Reference . . . 739
12.14.1 Detailed Description . . . 740
12.15 gripper_interface.h . . . 740
12.16 include/aubo/math.h File Reference . . . 744
12.16.1 Detailed Description . . . 745
12.17 math.h . . . 745
12.18 include/aubo/register_control.h File Reference . . . 758
12.18.1 Detailed Description . . . 759
12.18.2 Enumeration Type Documentation . . . 759
12.18.2.1 ModbusErrorNum . . . 759
12.19 register_control.h . . . 760
12.20 include/aubo/robot/force_control.h File Reference . . . 801
12.20.1 Detailed Description . . . 802
12.21 force_control.h . . . 803
12.22 include/aubo/robot/io_control.h File Reference . . . 833
12.22.1 Detailed Description . . . 834
12.23 io_control.h . . . 835
12.24 include/aubo/robot/motion_control.h File Reference . . . 901
12.24.1 Detailed Description . . . 902
12.25 motion_control.h . . . 903
12.26 include/aubo/robot/robot_algorithm.h File Reference . . . 981
12.26.1 Detailed Description . . . 982
12.27 robot_algorithm.h . . . 983
12.28 include/aubo/robot/robot_config.h File Reference . . . 1007
12.28.1 Detailed Description . . . 1008

12.29 robot_config.h . . . 1009
12.30 include/aubo/robot/robot_manage.h File Reference . . . 1053
12.30.1 Detailed Description . . . 1054
12.31 robot_manage.h . . . 1055
12.32 include/aubo/robot/robot_state.h File Reference . . . 1077
12.32.1 Detailed Description . . . 1078
12.33 robot_state.h . . . 1078
12.34 include/aubo/robot_interface.h File Reference . . . 1119
12.34.1 Detailed Description . . . 1120
12.35 robot_interface.h . . . 1120
12.36 include/aubo/runtime_machine.h File Reference . . . 1125
12.36.1 Detailed Description . . . 1126
12.37 runtime_machine.h . . . 1126
12.38 include/aubo/serial.h File Reference . . . 1144
12.38.1 Detailed Description . . . 1145
12.39 serial.h . . . 1146
12.40 include/aubo/socket.h File Reference . . . 1150
12.40.1 Detailed Description . . . 1151
12.41 socket.h . . . 1152
12.42 include/aubo/sync_move.h File Reference . . . 1159
12.42.1 Detailed Description . . . 1160
12.43 sync_move.h . . . 1161
12.44 include/aubo/system_info.h File Reference . . . 1172
12.44.1 Detailed Description . . . 1173
12.45 system_info.h . . . 1174
12.46 include/aubo/trace.h File Reference . . . 1178
12.46.1 Detailed Description . . . 1179
12.47 trace.h . . . 1179
12.48 include/aubo/type_def.h File Reference . . . 1182
12.48.1 Detailed Description . . . 1186
12.48.2 Macro Definition Documentation . . . 1186
12.48.2.1 CARTESIAN_DOF . . . 1186
12.48.2.2 DECL_TO_STRING_FUNC . . . 1186
12.48.2.3 ENUM_AuboErrorCodes_DECLARES . . . 1186
12.48.2.4 ENUM_AxisModeType_DECLARES . . . 1186
12.48.2.5 ENUM_EnvelopingShape_DECLARES . . . 1187
12.48.2.6 ENUM_HandleModeType_DECLARES . . . 1187
12.48.2.7 ENUM_HandleStateType_DECLARES . . . 1187
12.48.2.8 ENUM_ITEM [1/5] . . . 1187
12.48.2.9 ENUM_ITEM [2/5] . . . 1187
12.48.2.10 ENUM_ITEM [3/5] . . . 1188
12.48.2.11 ENUM_ITEM [4/5] . . . 1188
12.48.2.12 ENUM_ITEM [5/5] . . . 1188

12.48.2.13 ENUM_JointServoModeType_DECLARES . . . 1188
12.48.2.14 ENUM_JointStateType_DECLARES . . . 1189
12.48.2.15 ENUM_OperationalModeType_DECLARES . . . 1189
12.48.2.16 ENUM_PayloadIdentifyMoveAxis_DECLARES . . . 1189
12.48.2.17 ENUM_RobotControlModeType_DECLARES . . . 1189
12.48.2.18 ENUM_RobotEmergencyStopType_DECLARES . . . 1190
12.48.2.19 ENUM_RobotModeType_DECLARES . . . 1190
12.48.2.20 ENUM_RuntimeState_DECLARES . . . 1190
12.48.2.21 ENUM_SafeguedStopType_DECLARES . . . 1190
12.48.2.22 ENUM_SafetyInputAction_DECLARES . . . 1191
12.48.2.23 ENUM_SafetyModeType_DECLARES . . . 1191
12.48.2.24 ENUM_SafetyOutputRunState_DECLARES . . . 1191
12.48.2.25 ENUM_StandardInputAction_DECLARES . . . 1192
12.48.2.26 ENUM_StandardOutputRunState_DECLARES . . . 1192
12.48.2.27 ENUM_TaskFrameType_DECLARES . . . 1192
12.48.2.28 ENUM_TraceLevel_DECLARES . . . 1193
12.48.2.29 SAFETY_CUBIC_NUM . . . 1193
12.48.2.30 SAFETY_PARAM_SELECT_NUM . . . 1193
12.48.2.31 SAFETY_PLANES_NUM . . . 1193
12.48.2.32 TOOL_CONFIGURATION_NUM . . . 1193
12.49 type_def.h . . . 1194

Index 1207

Chapter 1

AuboStudio SDK

Current API Documentation Download: AuboStudio SDK API Documentation.pdf

Overview

AuboStudio SDK provides a complete set of interfaces for plugin development, offering users richer functionality.

Table of Contents

- Principles of Integrating Plugins into the Teach Pendant
- SDK Architecture
- Usage Examples
- Namespace

Principles of Integrating Plugins into the Teach Pendant

A plugin is a software package that runs as a subprocess of aubo_scope. aubo_scope registers plugins and interacts with them in various ways, such as during the startup process of aubo_scope and based on user interactions with aubo_scope.

Each plugin can contribute multiple new features to aubo_scope, and these features are typically called services. There are generally two types of services:

Services:

1. Provide installation node services.
2. Provide program node services.

Example (gripper aubo_caps):

1. Provide installation node services: such as configuring gripper installation and connection.
2. Provide program node services: for example, one program node can open the gripper, and another program node can close the gripper.

Principles of Integrating Plugins into aubo_scope:

Integration process diagram is as follows:

- When aubo_scope starts, it searches for all installed aubo_caps.
- Then aubo_scope calls the activator on each aubo_caps. The activator has two overwritten functions: start() and stop(). During startup, start() is called first to register services using RegisterService; when aubo_scope stops, the stop() function is called.
- After registration, the Service provides functions to configure basic node information, functions to create user interaction interfaces (view classes) for nodes, and functions to create Contribution classes for each node.
- In the contribution class, special functionality APIs can be provided for the node, as well as generating corresponding Script code.
- Usually, settings in the Service can be considered static settings because they are not available during motion and are only called at startup.

Principles of Integrating Program Nodes into aubo_scope:

Integration process diagram is as follows:

Specific Implementation:

ProgramNodeService

ProgramNodeService interface must implement the following functions:

getId()

> Parameters: void Return Value Type: String Return Value Content: myNodeType Call Timing: After service registration. Description: The getId() function returns the unique ID of this type of node. When there are multiple program nodes, this function is used to identify which program nodes belong to the same type of program node.

getTitle()

> Parameters: Locale region Return Value Type: String Return Value Content: Name of the program node. Call Timing: After service registration. Description: It is a static name that should tell the user what the node is for.

getIcon()

> Parameters: void Return Value Type: String Return Value Content: Returns the icon address of the node. Call Timing: After service registration. Description: Since each type of node has a different icon, during the dynamic node creation process, it is necessary to match the corresponding icon. Therefore, we need to obtain the icon link address of this type of node when creating the node.

configureContribution()

Parameters: ProgramContributionConfigurationPtr configuration Return Value Type: void Call Timing: After service registration. Description:

- ProgramContributionConfigurationPtr represents the configuration of this type of node;
- ProgramContributionConfigurationPtr objects can configure their values by calling associated functions;
- For example, setting whether the program node can have child nodes, or whether the node can be inserted by users or used only internally within aubo_caps.

```
configuration.setChildrenAllowed( false );
configuration.setUserInsertable( false );
```

createView()

Parameters: ViewApiProviderPtr api_provider Return Value Type: ProgramNodeViewPtr Return Value Content: Returns a new instance of the view class. Call Timing: Only once, when this type of node is first inserted by the user or loaded into the program. Description:

- createView is called only when the user first uses this type of node.
- The parameter ViewApiProviderPtr object can access SystemAPI and UserInterfaceAPI.
- The parameter ViewApiProviderPtr is usually forwarded to the view class. For example, if a keyboard needs to be provided in a text field, the keyboard is provided by User↩ InterfaceAPI.

createNode()

Parameters:

- ProgramApiProviderPtr api_provider,
- ProgramNodeViewPtr view,
- DataModelPtr model,
- ProgramCreationContextPtr context

Return Value Type: ProgramNodeContributionPtr Return Value Content:

- A new instance of the contribution class for this type of node;
- MyProgramNodeContribution (apiProvider, view, model, context).

Call Timing:

- Whenever a node of this type is created;
- Either when the user inserts a new node, then loads the program containing this type of node;
- Or when another aubo_caps inserts the program.

Description:

- createNode returns a new instance of the contribution class for this type of node. Although there is only one view instance, a new contribution class instance is created each time a new node is created;
- Parameters should be forwarded to the contribution class. ProgramApiProviderPtr can provide access to various APIs related to program nodes. The contribution object can use DataModelPtr to store configuration information for the node completed by the user. If a new node is inserted, DataModelPtr will be empty; if the node is loaded, DataModel↩ Ptr already contains the user's configuration information for the node. The contribution object can use ProgramCreationContextPtr to check whether its creation is the result of a new node insertion, or whether the node has already been loaded.

ProgramNodeView

ProgramNodeView interface must implement the following functions:

buildUi()

Parameters:

- WidgetHandle parent,
- ContributionProviderPtr <ProgramNodeContribution> contribution

Return Value Type: void Call Timing: Only once, when the user first inserts or loads this type of node into the program. Description:

- widgetHandle is the panel displayed to the user in aubo_scope. During the buildUi call, we nest the node's interface into the aubo_scope panel;
- Another parameter is ContributionProviderPtr. Since there can be multiple Contribution instances but only one view instance, when the user interacts with the node, ContributionProviderPtr will provide access to the Contribution. Calling contribution.↩ get() will return the active ProgramNodeContribution instance.

ProgramNodeContribution

ProgramNodeContribution interface must implement the following functions:

openView()

Parameters: void Return Value Type: void Call Timing: Each time the user enters a specific node interface. Description:

- Since there is only one instance of the user interface view for the same type of node, but there may be multiple nodes of this type, and there may also be multiple instances of ProgramNodeContribution, therefore openView() should be called to update the view to display the specific settings of each node;
- Usually, this call reads settings from the dataModel, which can be used to dynamically update related settings in the UI. This function can also be used to start a timer that can dynamically update values in the UI as long as the node is in focus.

closeView()

Parameters: void Return Value Type: void Call Timing: Each time the user leaves the user interface of a specific node. Description: The call to closeView() notifies the Contribution that the user is no longer viewing this node. If a timer or other running task was started in openView(), it should now be stopped.

getTitle()

Parameters: void Return Value Type: String Return Value Content: Returns the name of the program node shown in the program tree. Call Timing: Called frequently, usually when the dataModel is changed. Description:

- The difference between getTitle() in Contribution and getTitle() in Service: The get↩ Title() function in Contribution returns the name of the program node shown in the program tree, which is used to distinguish different nodes of the same type; The get↩ Title() function in Service returns the basic name of the node in the available node list, which is a static name used to distinguish which nodes are of the same type. This also explains that the name of the node in the program tree usually changes based on the node's configuration.
- Usually, this call will change the node's name based on the current configuration. It is used to simplify the user's navigation in the program tree. For example, if this is a program node for an adaptive gripper that can set a specific grip width, when set to a 35mm grip width, it might return "Gripper Move: 35 mm". This helps users find the appropriate node to configure. If the grip width is not set, i.e., the node is undefined, the title might just be "Gripper Move".

isDefined()

Parameters: void Return Value Type: bool Return Value Content: Whether the current node configuration can generate a script. Returns false if data for generating Script is missing, otherwise returns true. Call Timing: Whenever any modification occurs anywhere in the program. Description:

- isDefined() is called whenever any modification occurs anywhere in the program. This means that even if this node is not in focus, isDefined() will be called many times, which requires that the code executed in the callback must be very concise and depend on values in the dataModel.
- The purpose of this function is not to signal whether external hardware is connected; this should be handled at runtime. isDefined() is only used to remind the user whether more configuration is needed for this node.

generateScript()

Parameters: ScriptWriterPtr script_writer Return Value Type: void Call Timing: Usually when starting or saving a program. Description:

- The purpose of program nodes and installation nodes is to generate some LuaScript, which will be executed when the program is run. The generateScript() function generates the corresponding script for these nodes. The generated script depends on the configuration in the dataModel or the static configuration set in the service when creating the node.
- This script should be created using the ScriptWriter object.

Principles of Integrating Installation Nodes into aubo_scope:

Integration process diagram is as follows:

Specific Implementation:

InstallationNodeService

InstallationNodeService interface must implement the following functions:

getTitle()

Parameters: Locale region Return Value Type: String Return Value Content: Returns a static name for this installation node that can hint the user what the node is for. Description: The system Locale is passed as a parameter, which can be used to return the node name localized to the system language.

getIcon()

Parameters: void Return Value Type: String Return Value Content: Returns the icon address of the node. Call Timing: After service registration. Description: Since each type of node has a different icon, during the dynamic node creation process, it is necessary to match the corresponding icon. Therefore, we need to obtain the icon link address of this type of node when creating the node.

configureContribution()

Parameters: InstallationContributionConfigurationPtr configuration Return Value Type:↩ void Call Timing: After service registration. Description: The InstallationContribution↩ ConfigurationPtr object stores the configuration for this type of node. It has been implemented for future use, and currently does not provide any options; it should be left blank.

createView()

Parameters: ViewApiProviderPtr api_provider Return Value Type: InstallationNodeView↩ Ptr Return Value Content: Returns a new instance of the view class. Call Timing: Only once, when this type of node is first inserted by the user or loaded into the program. Description:

- createView is called only when the user first uses this type of node.
- The parameter ViewApiProviderPtr object can access SystemAPI and UserInterfaceAPI.
- The parameter ViewApiProviderPtr is usually forwarded to the view class. For example, if a keyboard needs to be provided in a text field, the keyboard is provided by User↩ InterfaceAPI.

createNode()

Parameters:

- InstallationApiProviderPtr api_provider,
- InstallationNodeViewPtr view,
- DataModelPtr model,
- InstallationCreationContextPtr context

Return Value Type: InstallationNodeContributionPtr Return Value Content:

- A new instance of the contribution class for this type of node;
- MyInstallationNodeContribution (apiProvider, view, model, context).

Call Timing:

- Whenever a node of this type is created;
- Either when the user inserts a new node, then loads the program containing this type of node;
- Or when another aubo_caps inserts the program.

Description:

- createNode returns a new instance of the contribution class for this type of node. Although there is only one view instance, a new contribution class instance is created each time a new node is created;
- Parameters should be forwarded to the contribution class. ProgramApiProviderPtr can provide access to various APIs related to program nodes. The contribution object can use DataModelPtr to store configuration information for the node completed by the user. If a new node is inserted, DataModelPtr will be empty; if the node is loaded (for example, as part of an installation file), DataModelPtr already contains the user's configuration information for the node. The contribution object can use ProgramCreationContextPtr to check whether its creation is the result of a new node insertion, or whether the node has already been loaded.

InstallationNodeView

InstallationNodeView interface must implement the following functions:

buildUi()

Parameters:

- WidgetHandle parent,
- InstallationNodeContributionPtr contribution

Return Value Type: void Call Timing: Only once, when the user first inserts or loads this type of node into the program. Description:

- widgetHandle is the panel displayed to the user in aubo_scope. During the buildUi call, we nest the node's interface into the aubo_scope panel;
- Another parameter is InstallationNodeContributionPtr. Since there can be multiple Contribution instances but only one view instance, when the user interacts with the node, ContributionProviderPtr will provide access to the Contribution. Calling contribution.↩ get() will return the active ProgramNodeContribution instance.

InstallationNodeContribution

InstallationNodeContribution interface must implement the following functions:

openView()

Parameters: void Return Value Type: void Call Timing: Each time the user enters a specific node interface. Description:

- Since there is only one instance of the user interface view for the same type of node, but there may be multiple nodes of this type, and there may also be multiple instances of InstallationNodeContribution, therefore openView() should be called to update the view to display the specific settings of each node;
- Usually, this call reads settings from the dataModel, which can be used to dynamically update related settings in the UI. This function can also be used to start a timer that can dynamically update values in the UI as long as the node is in focus.

closeView()

Parameters: void Return Value Type: void Call Timing: Each time the user leaves the user interface of a specific node. Description: The call to closeView() notifies the Contribution that the user is no longer viewing this node. If a timer or other running task was started in openView(), it should now be stopped.

generateScript()

Parameters: ScriptWriterPtr script_writer Return Value Type: void Call Timing: Usually when starting or saving a program. Description: The purpose of program nodes and installation nodes is to generate some Script, which will be executed when the program is run. The generateScript() function generates the corresponding lua script for these nodes. The generated script depends on the configuration in the dataModel or the static configuration set in the service when creating the node.

SDK Architecture

AuboStudio SDK provides a set of functions for plugins installed in the robot. Its ”core” is Contribution. Contribution is a new service that can be added to aubo_scope, providing users with richer functionality.

There are two general types of Contribution:

1. Installation Node

- Installation nodes are extensions of the robot installation domain;
- There is no order between installation nodes;
- Installation nodes can provide user interaction interfaces and add functions to configure ”environment settings”.

1. Program Node

- Program nodes are extensions of the robot program domain;
- Program nodes are arranged in order in the program tree, which determines the order in which the robot executes instructions when running the program;
- Program nodes can provide new programming ideas for the robot's program flow and provide user interaction interfaces.

The extension generic types (installation nodes, program nodes) provided by plugins have two user interfaces that can access many functions in the SDK. The APIs provided to these nodes are usually called domain APIs. These APIs provide access to detailed domain information and logic of the robot.

Since program nodes and installation nodes are obtained in the context of robot programs, this context has special API capabilities. These capabilities only make sense in the Installation domain or Program domain. These two types of APIs are Installation API and Program API respectively.

Domain APIs

The common APIs provided to general node types are called domain APIs. However, not all functionality in this API is meaningful to all node types at any given time. Therefore, domain APIs are provided to nodes using aubo_caps as reasonable API subtypes.

There are three domain-based APIs.

1. Application API

Provides access to APIs related to robot or application-specific functionality. Such as robot I/O, program variables, or coordinate systems (features).

1. System API

Provides access to APIs containing system information about the robot. Such as serial number, software version, or system language.

1. User Interface API

Provides access to APIs related to user interaction in the UI. Such as creating keyboards, or asking users to define positions.

Installation API

1. Provides functionality that only makes sense in the Program domain.
2. Installation API is an extension of the Application API, including special features available for installation nodes, such as creating Tcp and creating Feature.

Program API

1. Provides functionality that only makes sense in the Installation domain.
2. Program API is an extension of the System API and User Interface API, including features available for program nodes, such as building program node templates.

Access to Installation API and Program API is provided in the node-specific Provider objects.

Layer Structure

Domain APIs
  system_api (System Information)
  user_interface_api (User Interface)
  application_api (Application)
Installation API
  installation_api (Installation)
Program API
  program_api (Program)

Usage Examples

Developing a Service Class

Purpose: The service class provides functions to configure basic node information, functions to create user interaction interfaces (view classes) for nodes, and functions to create Contribution classes for each node.

For example, developing a program node named hello:

```
helloProgramNodeService::helloProgramNodeService(){}
helloProgramNodeService::~helloProgramNodeService(){}

std::string helloProgramNodeService::getId()
{
    return "hello";
}

std::string helloProgramNodeService::getTitle()
{
    return "hello";
}

std::string helloProgramNodeService::getIcon()
{
    return "img:node-comment.png";
}

void helloProgramNodeService::configureContribution(
        ProgramContributionConfigurationPtr configuration)
{
}

ProgramNodeViewPtr helloProgramNodeService::createView(
        ViewApiProviderPtr api_provider)
{
    return std::make_shared<helloProgramNodeView>(api_provider);
}

ProgramNodeContributionPtr helloProgramNodeService::createNode(
        ProgramApiProviderPtr api_provider, ProgramNodeViewPtr view,
        DataModelPtr model, ProgramCreationContextPtr context)
{
    return std::make_shared<helloProgramNodeContribution>(
                api_provider, std::dynamic_pointer_cast<helloProgramNodeView>(view),
                model);
}
```

Registering Services in the Activator

When the service class is complete, this service needs to be registered in the activator. The activator should provide all services that the nodes register to aubo_scope. When aubo_scope starts, the activator's start() method will be called, and the related services should be registered in the bundle context.

```
#include "cppmicroservices/Bundle.h"
#include "cppmicroservices/BundleActivator.h"
#include "cppmicroservices/BundleImport.h"
#include "cppmicroservices/BundleContext.h"
#include "cppmicroservices/BundleResource.h"
#include "cppmicroservices/BundleResourceStream.h"
#include "cppmicroservices/Constants.h"
#include "cppmicroservices/ServiceEvent.h"

#include "hello_program_node_service.h"

using namespace cppmicroservices;

/**
 * This class implements a simple bundle that utilizes the CppMicroServices's
 * event mechanism to listen for service events. Upon receiving a service event,
 * it prints out the event's details.
 */
class Activator : public BundleActivator
{
private:
    /**
     * Implements BundleActivator::Start(). Prints a message and adds a member
```

```
     * function to the bundle context as a service listener.
     *
     * @param context the framework context for the bundle.
     */
    void Start(BundleContext context)
    {
        auto bundle = context.GetBundle();
        auto headers = bundle.GetHeaders();

        cppmicroservices::ServiceProperties props;
        props["Name"] = std::string("MyFirstAuboCap");
        props["VendorName"] = std::string("aubo-robotics");
        props["Version"] = 0;

        //Register service
        context.RegisterService<arcs::aubo_scope::ProgramNodeService>(
            std::make_shared<helloProgramNodeService>(), props);
    }

    /**
     * Implements BundleActivator::Stop(). Prints a message and removes the
     * member function from the bundle context as a service listener.
     *
     * @param context the framework context for the bundle.
     */
    void Stop(BundleContext /*context*/) {}
};

CPPMICROSERVICES_EXPORT_BUNDLE_ACTIVATOR(Activator)
```

Developing a Contribution Class

Purpose: Provide the special APIs needed for the node, and generate Script scripts to execute the functionality of the program node. This code is based on the configuration information stored in data↩ Model.

For example, developing a program node named hello:

```
helloProgramNodeContribution::helloProgramNodeContribution(
        ProgramApiProviderPtr api_provider, helloProgramNodeViewPtr view,
        DataModelPtr model)
    : api_provider_(api_provider), view_(view), model_(model)
{
}

helloProgramNodeContribution::~helloProgramNodeContribution()
{
}

void helloProgramNodeContribution::openView()
{
}

void helloProgramNodeContribution::closeView()
{
}

std::string helloProgramNodeContribution::getTitle()
{
    return "hello";
}

bool helloProgramNodeContribution::isDefined()
{
    return true;
}

void helloProgramNodeContribution::generateScript(ScriptWriterPtr script_writer)
{
}
```

Developing a View Class

Purpose: Provide user interaction interface for the node.

For example, developing a program node named hello:

```
helloProgramNodeView::helloProgramNodeView(ViewApiProviderPtr api_provider,
                                           QWidget *parent)
    : QWidget(parent), ui(new Ui::helloProgramNodeView), view_api_(api_provider)
{
}

helloProgramNodeView::~helloProgramNodeView()
{
    delete ui;
}

void helloProgramNodeView::buildUi(
        WidgetHandle parent,
        ContributionProviderPtr<ProgramNodeContribution> contribution)
{
    setParent((QWidget *)parent);
    ui->setupUi(this);
}
```

Namespace

All interfaces are defined in the following namespaces:

```
namespace arcs {
namespace aubo_scope {
    // All interface definitions
}
}
```

Chapter 2

Topic Index

2.1 Topics

Here is a list of all topics with brief descriptions:

AuboApi ( ) . . . 25
AxisInterface ( ) . . . 30
GripperInterface ( ) . . . 42
Math ( ) . . . 45
RegisterControl ( ) . . . 62
RobotInterface( ) . . . 117
ForceControl( ) . . . 121
IoControl (IO) . . . 158
MotionControl ( ) . . . 246
RobotAlgorithm ( ) . . . 349
RobotConfig ( ) . . . 381
RobotManage ( ) . . . 439
RobotState ( ) . . . 467
RuntimeMachine ( ) . . . 521
Serial ( ) . . . 545
Socket (socket) . . . 552
SyncMove ( ) . . . 563
SystemInfo ( ) . . . 578
Trace ( ) . . . 582

Chapter 3

Directory Hierarchy

3.1 Directories

aubo . . . 587
  error_stack . . . 588
    error_stack.h . . . 715
    hal_error.h . . . 723
    rtm_error.h . . . 733
    system_error.h . . . 737
  robot . . . 589
    force_control.h . . . 801
    io_control.h . . . 833
    motion_control.h . . . 901
    robot_algorithm.h . . . 981
    robot_config.h . . . 1007
    robot_manage.h . . . 1053
    robot_state.h . . . 1077
  aubo_api.h . . . 701
  axis_interface.h . . . 706
  gripper_interface.h . . . 739
  math.h . . . 744
  register_control.h . . . 758
  robot_interface.h . . . 1119
  runtime_machine.h . . . 1125
  serial.h . . . 1144
  socket.h . . . 1150
  sync_move.h . . . 1159
  system_info.h . . . 1172
  trace.h . . . 1178
  type_def.h . . . 1182
doc . . . 588
error_stack . . . 588
  error_stack.h . . . 715
  hal_error.h . . . 723
  rtm_error.h . . . 733
  system_error.h . . . 737
include . . . 589
  aubo . . . 587
    error_stack . . . 588
      error_stack.h . . . 715

hal_error.h . . . 723
rtm_error.h . . . 733
system_error.h . . . 737
robot . . . 589
force_control.h . . . 801
io_control.h . . . 833
motion_control.h . . . 901
robot_algorithm.h . . . 981
robot_config.h . . . 1007
robot_manage.h . . . 1053
robot_state.h . . . 1077
aubo_api.h . . . 701
axis_interface.h . . . 706
gripper_interface.h . . . 739
math.h . . . 744
register_control.h . . . 758
robot_interface.h . . . 1119
runtime_machine.h . . . 1125
serial.h . . . 1144
socket.h . . . 1150
sync_move.h . . . 1159
system_info.h . . . 1172
trace.h . . . 1178
type_def.h . . . 1182
robot . . . 589
force_control.h . . . 801
io_control.h . . . 833
motion_control.h . . . 901
robot_algorithm.h . . . 981
robot_config.h . . . 1007
robot_manage.h . . . 1053
robot_state.h . . . 1077

Chapter 4

Namespace Index

4.1 Namespace List

Here is a list of all namespaces with brief descriptions:

arcs 591
arcs::common_interface 591
arcs::error_stack 608

Chapter 5

Hierarchical Index

5.1 Class Hierarchy

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

arcs::common_interface::AuboApi . . . 611
arcs::common_interface::AxisInterface . . . 616
arcs::common_interface::CircleParameters . . . 618
arcs::common_interface::Enveloping . . . 621
std::exception
  arcs::common_interface::AuboException . . . 613
arcs::common_interface::ForceControl . . . 622
arcs::common_interface::GripperInterface . . . 625
arcs::common_interface::GripperStatus . . . 629
arcs::common_interface::IoControl . . . 632
arcs::common_interface::Math . . . 636
arcs::common_interface::MotionControl . . . 638
arcs::common_interface::RegisterControl . . . 647
arcs::common_interface::RobotAlgorithm . . . 651
arcs::common_interface::RobotConfig . . . 655
arcs::common_interface::RobotInterface . . . 659
arcs::common_interface::RobotManage . . . 661
arcs::common_interface::RobotMsg . . . 663
arcs::common_interface::RobotSafetyParameterRange . . . 665
arcs::common_interface::RobotState . . . 675
arcs::common_interface::RtdeRecipe . . . 680
arcs::common_interface::RuntimeMachine . . . 682
arcs::common_interface::Serial . . . 685
arcs::common_interface::Socket . . . 686
arcs::common_interface::SpiralParameters . . . 688
arcs::common_interface::SyncMove . . . 689
arcs::common_interface::SystemInfo . . . 692
arcs::common_interface::Trace . . . 693
arcs::common_interface::TrajConfig . . . 695
arcs::common_interface::VibrationRecalibrationParameter . . . 697
arcs::common_interface::WObjectData . . . 699

Chapter 6

Class Index

6.1 Class List

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

arcs::common_interface::AuboApi . . . 611
arcs::common_interface::AuboException
Custom exception class AuboException . . . 613
arcs::common_interface::AxisInterface . . . 616
arcs::common_interface::CircleParameters
Circular motion parameters definition . . . 618
arcs::common_interface::Enveloping . . . 621
arcs::common_interface::ForceControl . . . 622
arcs::common_interface::GripperInterface . . . 625
arcs::common_interface::GripperStatus . . . 629
arcs::common_interface::IoControl . . . 632
arcs::common_interface::Math . . . 636
arcs::common_interface::MotionControl . . . 638
arcs::common_interface::RegisterControl . . . 647
arcs::common_interface::RobotAlgorithm . . . 651
arcs::common_interface::RobotConfig . . . 655
arcs::common_interface::RobotInterface . . . 659
arcs::common_interface::RobotManage . . . 661
arcs::common_interface::RobotMsg . . . 663
arcs::common_interface::RobotSafetyParameterRange . . . 665
arcs::common_interface::RobotState . . . 675
arcs::common_interface::RtdeRecipe
RTDE menu . . . 680
arcs::common_interface::RuntimeMachine . . . 682
arcs::common_interface::Serial . . . 685
arcs::common_interface::Socket . . . 686
arcs::common_interface::SpiralParameters . . . 688
arcs::common_interface::SyncMove . . . 689
arcs::common_interface::SystemInfo . . . 692
arcs::common_interface::Trace . . . 693
arcs::common_interface::TrajConfig
Trajectory configuration for payload identification . . . 695
arcs::common_interface::VibrationRecalibrationParameter . . . 697
arcs::common_interface::WObjectData . . . 699

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 . . . 701

include/aubo/axis_interface.h . . . 706

include/aubo/gripper_interface.h . . . 739

include/aubo/math.h
: Mathematic operation interface, such as euler to quaternion conversion, addition/↩ subtraction of poses . . . 744

include/aubo/register_control.h . . . 758

include/aubo/robot_interface.h
: API . . . 1119

include/aubo/runtime_machine.h
: Script interpreter runtime interface, allows pausing the script interpreter and setting/↩ removing breakpoints . . . 1125

include/aubo/serial.h . . . 1144

include/aubo/socket.h
: Socket . . . 1150

include/aubo/sync_move.h . . . 1159

include/aubo/system_info.h . . . 1172

include/aubo/trace.h
: Interface for injecting logs into the controller's logging system . . . 1178

include/aubo/type_def.h
: Enum type definitions . . . 1182

include/aubo/error_stack/error_stack.h . . . 715

include/aubo/error_stack/hal_error.h . . . 723

include/aubo/error_stack/rtm_error.h . . . 733

include/aubo/error_stack/system_error.h . . . 737

include/aubo/robot/force_control.h
801
include/aubo/robot/io_control.h
IO 833
include/aubo/robot/motion_control.h
Motion control interface 901
include/aubo/robot/robot_algorithm.h
. 981
include/aubo/robot/robot_config.h
DH 1007
include/aubo/robot/robot_manage.h
. 1053
include/aubo/robot/robot_state.h
/ 1077

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 syncronous 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( )

Get RobotInterfacePtr based on name

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 syncronous 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.
- int arcs::common_interface::AxisInterface::setExtAxisType (int type)
  Set external axis type.
- 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
- int arcs::common_interface::AxisInterface::setExtAxisVelocity (double velocity)
  Set external axis speed
- double arcs::common_interface::AxisInterface::getExtAxisAcceleration ()
  Get external axis acceleration
- int arcs::common_interface::AxisInterface::setExtAxisAcceleration (double acc)
  Set 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
- int arcs::common_interface::AxisInterface::setExtAxisMaxPositionLimit (double q)
  Set external axis max position
- double arcs::common_interface::AxisInterface::getExtAxisMaxPositionLimit ()

Get external axis max position

- double arcs::common_interface::AxisInterface::getExtMinPosition ()
  Get external axis min position
- int arcs::common_interface::AxisInterface::setExtMinPositionLimit (double q)
  Set external axis min position
- double arcs::common_interface::AxisInterface::getExtMinPositionLimit ()
  Get external axis min position
- double arcs::common_interface::AxisInterface::getExtAxisMaxVelocity ()
  Get external axis max speed
- int arcs::common_interface::AxisInterface::setExtAxisMaxVelocityLimit (double v)
  Set external axis max speed
- double arcs::common_interface::AxisInterface::getExtAxisMaxVelocityLimit ()
  Get external axis max speed
- double arcs::common_interface::AxisInterface::getExtAxisMaxAcceleration ()
  Get external axis max acceleration
- int arcs::common_interface::AxisInterface::setExtAxisMaxAccelerationLimit (double acc)
  Set external axis max acceleration
- double arcs::common_interface::AxisInterface::getExtAxisMaxAccelerationLimit ()
  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 ()
  getErrorCode(Get raw external axis error code)
- int arcs::common_interface::AxisInterface::clearAxisError ()
  Reset axis error
- int arcs::common_interface::AxisInterface::setExtAxisZero ()
  Set current external axis position as zero.
- int arcs::common_interface::AxisInterface::setReductionRatio (double ratio)
  Set axis reduction ratio
- double arcs::common_interface::AxisInterface::getReductionRatio ()
  Get axis reduction ratio

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 ()

getErrorCode(Get raw external axis error code)

Returns

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 getExtAxisMaxAccelerationLimit()

double arcs::common_interface::AxisInterface::getExtAxisMaxAccelerationLimit ()

Get external axis max acceleration

Returns

External axis max acceleration

8.2.2.12 getExtAxisMaxPosition()

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

Get external axis max position

Returns

External axis max position

8.2.2.13 getExtAxisMaxPositionLimit()

double arcs::common_interface::AxisInterface::getExtAxisMaxPositionLimit ()

Get external axis max position

Returns

External axis max position

8.2.2.14 getExtAxisMaxVelocity()

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

Get external axis max speed

Returns

External axis max speed

8.2.2.15 getExtAxisMaxVelocityLimit()

double arcs::common_interface::AxisInterface::getExtAxisMaxVelocityLimit ()

Get external axis max speed

Returns

External axis max speed

8.2.2.16 getExtAxisMountingPose()

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

Get external axis mounting pose

Returns

External axis pose

8.2.2.17 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

8.2.2.18 getExtAxisPosition()

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

Get external axis position

Returns

External axis position

8.2.2.19 getExtAxisTemperature()

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

Get external axis temperature

Returns

External axis temperature

8.2.2.20 getExtAxisType()

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

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

Returns

8.2.2.21 getExtAxisVelocity()

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

Get external axis speed

Returns

External axis speed

8.2.2.22 getExtMinPosition()

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

Get external axis min position

Returns

External axis min position

8.2.2.23 getExtMinPositionLimit()

double arcs::common_interface::AxisInterface::getExtMinPositionLimit ()

Get external axis min position

Returns

External axis min position

8.2.2.24 getReductionRatio()

double arcs::common_interface::AxisInterface::getReductionRatio ()

Get axis reduction ratio

Returns

8.2.2.25 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

8.2.2.26 poweroffExtAxis()

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

Power off.

Returns

8.2.2.27 poweronExtAxis()

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

Power on.

Returns

8.2.2.28 setExtAxisAcceleration()

int arcs::common_interface::AxisInterface::setExtAxisAcceleration (
double acc)

Set external axis acceleration

Parameters

acc	

Returns

Parameters

acc	

Returns

8.2.2.30 setExtAxisMaxPositionLimit()

int arcs::common_interface::AxisInterface::setExtAxisMaxPositionLimit (
double q)

Set external axis max position

Parameters

q	

Returns

8.2.2.31 setExtAxisMaxVelocityLimit()

int arcs::common_interface::AxisInterface::setExtAxisMaxVelocityLimit (
double v)

Set external axis max speed

Parameters

v	

Returns

8.2.2.32 setExtAxisMountingPose()

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

Parameters

pose	

Returns

8.2.2.33 setExtAxisType()

int arcs::common_interface::AxisInterface::setExtAxisType (
int type)

Set external axis type.

Parameters

type	0 for rotation, 1 for linear

Returns

8.2.2.34 setExtAxisVelocity()

int arcs::common_interface::AxisInterface::setExtAxisVelocity (
double velocity)

Set external axis speed

Returns

8.2.2.35 setExtAxisZero()

int arcs::common_interface::AxisInterface::setExtAxisZero ()

Set current external axis position as zero.

Returns

8.2.2.36 setExtMinPositionLimit()

Parameters

q	

Returns

8.2.2.37 setReductionRatio()

int arcs::common_interface::AxisInterface::setReductionRatio (
double ratio)

Set axis reduction ratio

Returns

8.2.2.38 speedExtJoint()

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

Set target speed, acceleration and duration

Parameters

v	
a	
duration	

Returns

8.2.2.39 stopExtJoint()

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

Parameters

a	

Returns

8.2.2.40 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::gripperDelete (
const std::string & name)

8.3.2.4 gripperDisconnect()

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

8.3.2.5 gripperEnable()

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

8.3.2.6 gripperGetHardwareVersion()

std::string arcs::common_interface::GripperInterface::gripperGetHardwareVersion (
const std::string & name)

8.3.2.7 gripperGetNames()

std::vector< std::string > arcs::common_interface::GripperInterface::gripperGetNames ()

8.3.2.8 gripperGetStatusCode()

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

8.3.2.9 gripperGetSupportedModels()

std::vector< std::string > arcs::common_interface::GripperInterface::gripperGetSupportedModels ()

8.3.2.10 gripperGetWorkMode()

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

8.3.2.11 gripperIsConnected()

bool arcs::common_interface::GripperInterface::gripperIsConnected (
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_from_to)

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

  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 > &rotv)

  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,-3.67320510315008 3e-06,1.570796326792424}, {0.5488696249770835,-
0.1214996547187207,0.3599720701808493,-3.14159198038469,-3.6732051029273e-06,1.570796326792423},
{0.5488696249770836,-0.0389996547187214,0.3599720701808496,-3.141591980384691,-3.67320510255747 6e-
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: force_in_b = pose_a_in_b ∗ 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]} \endengish

8.4.2.11 poseAngleDistance()

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

Calculate axis-angle difference between two poses

Parameters

p1	pose 1
p2	pose 2

Returns

axis angle difference

Lua

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

Lua

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]}

8.4.2.17 poseToHomMatrix()

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

poseToHomMatrix: Get homogeneous transformation matrix from pose

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_world->to = T_world->from ∗ T_from->to T_x->to = T_x->from ∗ T_from->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

pose_from	starting pose vector in 3D space
pose_from_to	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_from_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

rpy = 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::addFixedModbusGroup (const std::string &signal_name, int count)

  Create a fixed read group for the specified Modbus register signal to optimize the batch reading efficiency of continuous addresses
- 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::modbusSetOutputRunstate1 (const std::string &robot_name, const std::string &signal_name, StandardOutputRunState runstate, const std::string &object_name, double threshold)

  Set Modbus signal output action with optional condition checking based on waypoint, plane or obstacle thresholds.
- 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 addFixedModbusGroup()

```
int arcs::common_interface::RegisterControl::addFixedModbusGroup (
        const std::string & signal_name,
        int count)
```

Create a fixed read group for the specified Modbus register signal to optimize the batch reading efficiency of continuous addresses

Function: Locate the address of the corresponding 16-bit Modbus register according to the input start signal name, and create a "non-extensible fixed group" based on the start address and continuous count. This group will be managed independently, and will not be merged or expanded with other normal groups. It is suitable for continuous register address segments that require stable and atomic reading; After the fixed group is created, the read operation for this address segment will be strictly executed according to the group range, avoiding read range changes caused by dynamic group expansion.

Parameters

signal_name	Start signal name (used as the reference signal for the fixed group, must be a 16-bit Modbus register signal created via modbusAddSignal, supporting holding register/input register types, not supporting digital signals)
count	Number of continuous signals (registers) contained in the fixed group (greater than or equal to 1): • Value = 1: Only include the single register address corresponding to the start signal (cancel the fixed group containing this signal); • Value > 1: Continuous count register addresses starting from the start signal address, ensure the addresses are continuous and do not exceed the device support range;

Returns

Returns 0 on success, returns an error code on failure.

Note

1. This interface only takes effect for 16-bit Modbus register signals (signal_type is 2/3);
2. The maximum continuous count of a single fixed group is limited by the device (default maximum 30);
3. Repeated creation of a fixed group for the same start signal will overwrite the original configuration (the last count value shall prevail).

Python interface prototype

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

Lua interface prototype

addFixedModbusGroup(signal_name: string, count: number) -> nil

Lua example

– Create a fixed group containing 10 consecutive registers for the start signal ”Modbus_0” add↩ FixedModbusGroup(”Modbus_0”, 10)

JSON-RPC request example

{”jsonrpc”:”2.0”,”method”:”RegisterControl.addFixedModbusGroup”,”params”:[”Modbus_↩ 0”,10],”id”:1}

JSON-RPC response example

{”id”:1,”jsonrpc”:”2.0”,”result”:0}

8.5.2.2 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

8.5.2.3 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

8.5.2.4 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

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

8.5.2.5 deleteVirtualEncoder()

Parameters

encoder_id	

Returns

8.5.2.6 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}

8.5.2.8 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}

8.5.2.9 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}

8.5.2.10 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}

8.5.2.11 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}

8.5.2.12 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}

8.5.2.13 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}

8.5.2.14 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}

8.5.2.15 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}

8.5.2.16 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}

8.5.2.17 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}

8.5.2.18 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}

8.5.2.19 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}

8.5.2.20 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}

8.5.2.21 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”}

8.5.2.22 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"}

8.5.2.23 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]}

8.5.2.24 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]}

8.5.2.25 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]}

8.5.2.26 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]}

8.5.2.27 getWatchDogAction()

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

Get the watchdog action

Parameters

key	

Returns

8.5.2.28 getWatchDogTimeout()

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

Get the watchdog timeout value

Parameters

key	

Returns

8.5.2.29 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}

8.5.2.30 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/ttyRobot↩ Tool,115200,N,8,1",1,264,3,"Modbus_0",false],"id":1}

JSON-RPC response example

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

8.5.2.31 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}

8.5.2.32 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}

8.5.2.33 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}

8.5.2.34 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]}

8.5.2.35 modbusGetSignalIndex()

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}

8.5.2.36 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”]}

8.5.2.37 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}

8.5.2.38 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]}

8.5.2.39 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]}

8.5.2.40 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: number, data: table) -> nil

Lua example

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

JSON-RPC request example

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

JSON-RPC response example

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

8.5.2.41 modbusSetDigitalInputAction()

int arcs::common_interface::RegisterControl::modbusSetDigitalInputAction (

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}

8.5.2.42 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}

8.5.2.43 modbusSetOutputRunstate1()

```
int arcs::common_interface::RegisterControl::modbusSetOutputRunstate1 (
        const std::string & robot_name,
        const std::string & signal_name,
        StandardOutputRunState runstate,
        const std::string & object_name,
        double threshold)
```

Set Modbus signal output action with optional condition checking based on waypoint, plane or obstacle thresholds.

Detailed description: Use this API to control a Modbus output signal when a specified robot condition is met (for example, waypoint arrival or distance to a plane/obstacle). Specify an object type, name and a threshold value to enable conditional triggering. When the condition is satisfied, the given Modbus signal will be set to the requested runstate.

Parameters

robot_name	The target robot name identifying which robot's conditions to monitor.
signal_name	The Modbus signal name to set (must be previously added via modbusAddSignal).
runstate	The desired output action.
object_name	Unique identifier of the condition object
threshold	Condition threshold: angle in radians for waypoints, or distance in meters for planes/obstacles

Returns

Returns 0 on success, non-zero error code on failure.

JSON-RPC request example

{”jsonrpc”:”2.0”,”method”:”RegisterControl.modbusSetOutputRunstate1”,”params”:[”rob1”,”↩
Modbus_0”,”WaypointArrived”,”P1”,3],”id”:1}

JSON-RPC response example

{”id”:1,”jsonrpc”:”2.0”,”result”:0}

8.5.2.44 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

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}

8.5.2.46 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}

8.5.2.47 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}

8.5.2.48 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}

8.5.2.49 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}

8.5.2.50 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}

8.5.2.51 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}

8.5.2.52 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}

8.5.2.53 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}

8.5.2.54 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}

8.5.2.55 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}

8.5.2.56 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}

8.5.2.57 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}

8.5.2.58 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}

8.5.2.59 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}

8.5.2.60 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}

8.5.2.61 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}

8.5.2.62 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}

8.5.2.63 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}

8.5.2.64 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}

8.5.2.65 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}

8.5.2.66 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}

8.5.2.67 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

8.5.2.68 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

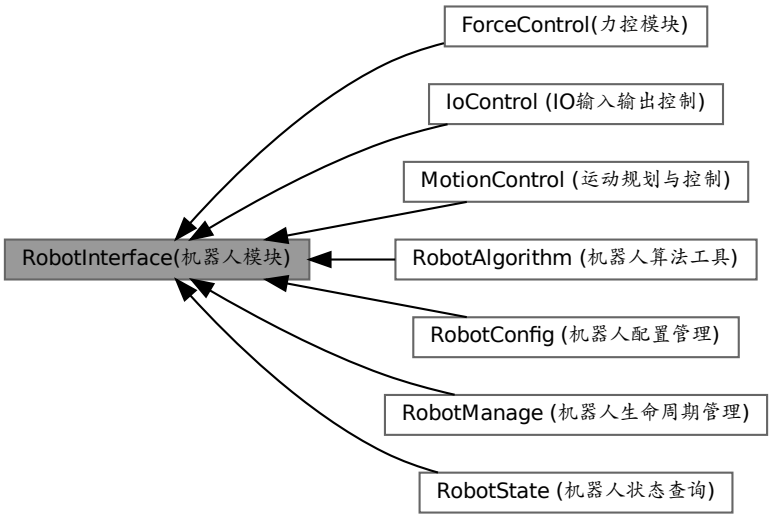
Lua example

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

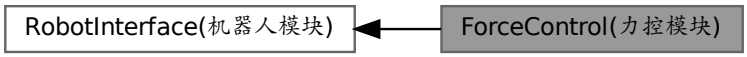
C++ example

```
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 > &center, 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 lateractivated 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)
```


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.

Generated by Doxygen

It is possible to specify that the condition is fulfilled when the speed is outside the specified limits instead.

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 kg · m², length is 6, order is [J1, J2, J3, J4, J5, J6]
- Damping:
  - Cartesian space: unit is N · s/m, order is [x, y, z, Rx, Ry, Rz]
  - Joint space: unit is N · m · s/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 N · m/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

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)
```

Set soft float parameters

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	

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

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

arcs::common_interface::AuboException	

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

arcs::common_interface::AuboException	

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

arcs::common_interface::AuboException	

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()

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.

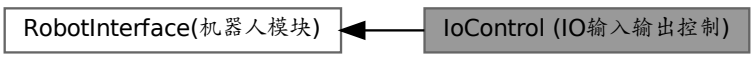
Returns

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.

- int arcs::common_interface::IoControl::setDigitalOutputRunstate (int io_type, int index, StandardOutputRunState runstate, const std::string &object_name, double threshold)

  Set digital output runstate with optional condition checking based on waypoint, plane or obstacle thresholds.

- 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.

- StandardOutputRunState arcs::common_interface::IoControl::getDigitalOutputRunstate (int io_type, int index)

  Get the current runstate of a digital output of the specified type.

- int arcs::common_interface::IoControl::addCrossConnection (const std::string &id, int operator_type, int result_type, int result_pin, const std::vector< int > &actor_types, const std::vector< int > &actor_pins, const std::vector< bool > &actor_inverts)

  Add a DIO Cross Connection logic rule.

- int arcs::common_interface::IoControl::deleteCrossConnection (const std::string &id)

  Delete a DIO Cross Connection logic rule.

- int arcs::common_interface::IoControl::clearCrossConnections ()

  Clear all DIO Cross Connection logic rules.

- 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::addGroupOutput (const std::string &name, int io_type, const std::vector< int > &indexes)
  Register a grouped digital output.
- int arcs::common_interface::IoControl::deleteGroupOutput (const std::string &name)
  Delete a grouped digital output.
- int arcs::common_interface::IoControl::setGroupOutput (const std::string &name, uint32_t value)
  Set the grouped digital output value.
- 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.

- int arcs::common_interface::IoControl::addGroupInput (const std::string &name, int io_type, const std::vector< int > &indexes)

  Register a grouped digital input.
- int arcs::common_interface::IoControl::deleteGroupInput (const std::string &name)

  Delete a grouped digital input.
- uint32_t arcs::common_interface::IoControl::getGroupInput (const std::string &name)

  Get the grouped digital input value.
- 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.
- uint32_t arcs::common_interface::IoControl::getGroupOutput (const std::string &name)

  Get the grouped digital output value.
- 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 addCrossConnection()

```
int arcs::common_interface::IoControl::addCrossConnection (
                const std::string & id,
                int operator_type,
                int result_type,
                int result_pin,
                const std::vector< int > & actor_types,
                const std::vector< int > & actor_pins,
                const std::vector< bool > & actor_inverts)
```

Add a DIO Cross Connection logic rule.

Supports digital/bool DIO signals and virtual DO signals. The rule takes effect immediately after it is added successfully. DI signals may only be read as actors and cannot be written as results. actor_inverts tells the controller whether to invert each actor's raw bool value before evaluating the rule; it does not change the hardware IO direction.

result_type and actor_types use the same numeric type set. As a result, the value denotes the output side; as an actor, it denotes the input side or a readable virtual DO:

- 0: Standard digital IO. result means standard DO, actor means standard DI.
- 1: Tool digital IO. result means tool DO, actor means tool DI.
- 2: Configurable digital IO. result means configurable DO, actor means configurable DI.
- 3: Virtual DO. It does not occupy a real DO pin and can be used as an intermediate result for cascaded rules. As a result it writes the virtual DO; as an actor it reads the virtual DO.

operator_type is interpreted by the controller implementation; the recommended mapping is 0=FWD, 1=NOT, 2=AND, 3=OR. FWD/NOT require exactly one actor, and AND/OR require 2 to 5 actors. actor_types, actor_pins, and actor_inverts must have the same length.

Limits:

- Up to 300 Cross Connection rules are allowed.
- A rule id can contain up to 32 characters and must be unique.
- Virtual DO cascade depth is limited to 20.
- Cyclic dependencies are not allowed.
- Multiple rules cannot write the same result, including real DO and virtual DO results.
- A virtual DO actor must be produced by an earlier rule; reading an undefined virtual DO should fail.
- A virtual DO pin/index only needs to be non-negative and does not occupy a real DO pin.
- A real DO written by Cross Connection should not also have a runstate configured. Later manual set/pulse calls to that DO should be rejected with a message that names the owning Cross Connection.

Parameters

id	Unique rule ID
operator_type	Logic operator type
result_type	Result signal type; must be writable output type or virtual DO
result_pin	Result signal pin/index; virtual DO uses virtual signal index
actor_types	Actor signal type list; may read DI or virtual DO
actor_pins	Actor signal pin/index list; virtual DO uses virtual signal index
actor_inverts	Actor polarity list

Returns

Returns 0 on success, non-zero error code on failure.

Exceptions

arcs::common_interface::AuboException	

Python function prototype

addCrossConnection(self: pyaubo_sdk.IoControl, arg0: str, arg1: int, arg2: int, arg3: int, arg4: List[int], arg5: List[int], arg6: List[bool]) -> int

Lua function prototype

addCrossConnection(id: string, operator_type: number, result_type: number, result_pin: number, actor_types: table, actor_pins: table, actor_inverts: table) -> number

Lua example

addCrossConnection("start_permit", 2, 0, 0, {0, 0}, {3, 4}, {false, true})

JSON-RPC request example

{"jsonrpc":"2.0","method":"rob1.IoControl.addCrossConnection","params":["start_permit",2,0,0,↩
[0,0],[3,4],[false,true]],"id":1}

JSON-RPC virtual DO cascade example

First generate the intermediate signal VirtualDO[0] = DI[0] AND NOT DI[1], then drive DO[0] = VirtualDO[0] OR DI[2]: {"jsonrpc":"2.0","method":"rob1.IoControl.clearCross↩
Connections","params":[],"id":10} {"jsonrpc":"2.0","method":"rob1.IoControl.addCrossConnection","params":↩
["v0_di0_and_not_di1",2,3,0,[0,0],[0,1],[false,true]],"id":11} {"jsonrpc":"2.0","method":"rob1.Io↩
Control.addCrossConnection","params":["v0_or_di2_to_do0",3,0,0,[3,0],[0,2],[false,false]],"id":↩
12}

JSON-RPC response example

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

8.6.4.2.2 addGroupInput()

int arcs::common_interface::IoControl::addGroupInput (
const std::string & name,
int io_type,
const std::vector< int > & indexes)

Register a grouped digital input.

Registers a named digital input group. indexes[0] maps to bit0, indexes[1] maps to bit1, and so on.

Parameters

name	Group name.
io_type	IO type. 0 means Standard, 1 means Tool, and 2 means Configurable.
indexes	Index list of digital inputs in the group.

Returns

Returns 0 on success; error code on failure.

Exceptions

arcs::common_interface::AuboException	

Python function prototype

addGroupInput(self: pyaubo_sdk.IoControl, arg0: str, arg1: int, arg2: List[int]) -> int

Lua function prototype

addGroupInput(name: string, io_type: number, indexes: table) -> nil

Lua example

addGroupInput("gi_state", 0, {0,1,2,3})

JSON-RPC request example

{"jsonrpc":"2.0","method":"rob1.IoControl.addGroupInput","params":["gi_state",0,[0,1,2,3]]↩
,"id":1}

JSON-RPC response example

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

8.6.4.2.3 addGroupOutput()

```
int arcs::common_interface::IoControl::addGroupOutput (
        const std::string & name,
        int io_type,
        const std::vector< int > & indexes)
```

Register a grouped digital output.

Registers a named digital output group. indexes[0] maps to bit0, indexes[1] maps to bit1, and so on.

Parameters

name	Group name.
io_type	IO type. 0 means Standard, 1 means Tool, and 2 means Configurable.
indexes	Index list of digital outputs in the group.

Returns

Returns 0 on success; error code on failure.

Exceptions

arcs::common_interface::AuboException	

Python function prototype

addGroupOutput(self: pyaubo_sdk.IoControl, arg0: str, arg1: int, arg2: List[int]) -> int

Lua function prototype

addGroupOutput(name: string, io_type: number, indexes: table) -> nil

Lua example

addGroupOutput("go_state", 0, {4,5,6,7})

JSON-RPC request example

{"jsonrpc":"2.0","method":"rob1.IoControl.addGroupOutput","params":["go_state",0,[4,5,6,7]]↩
,"id":1}

JSON-RPC response example

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

8.6.4.2.4 clearCrossConnections()

int arcs::common_interface::IoControl::clearCrossConnections ()

Clear all DIO Cross Connection logic rules.

This only clears rules. It does not clear or reset real digital output values.

Returns

Returns 0 on success, non-zero error code on failure.

Exceptions

arcs::common_interface::AuboException	

Python function prototype

clearCrossConnections(self: pyaubo_sdk.IoControl) -> int

Lua function prototype

clearCrossConnections() -> number

JSON-RPC request example

{"jsonrpc":"2.0","method":"rob1.IoControl.clearCrossConnections","params":[],"id":1}

JSON-RPC response example

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

8.6.4.2.5 deleteCrossConnection()

int arcs::common_interface::IoControl::deleteCrossConnection (
const std::string & id)

Delete a DIO Cross Connection logic rule.

Parameters

id	Unique rule ID

Returns

Returns 0 on success, non-zero error code on failure.

Exceptions

arcs::common_interface::AuboException	

Python function prototype

deleteCrossConnection(self: pyaubo_sdk.IoControl, arg0: str) -> int

Lua function prototype

deleteCrossConnection(id: string) -> number

JSON-RPC request example

{"jsonrpc":"2.0","method":"rob1.IoControl.deleteCrossConnection","params":["start_permit"]↩
,"id":1}

JSON-RPC response example

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

8.6.4.2.6 deleteGroupInput()

int arcs::common_interface::IoControl::deleteGroupInput (
const std::string & name)

Delete a grouped digital input.

Parameters

name	Group name.

Returns

Returns 0 on success; error code on failure.

Exceptions

arcs::common_interface::AuboException	

Python function prototype

deleteGroupInput(self: pyaubo_sdk.IoControl, arg0: str) -> int

Lua function prototype

deleteGroupInput(name: string) -> nil

JSON-RPC request example

{"jsonrpc":"2.0","method":"rob1.IoControl.deleteGroupInput","params":["gi_state"],"id":1}

JSON-RPC response example

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

8.6.4.2.7 deleteGroupOutput()

int arcs::common_interface::IoControl::deleteGroupOutput (
const std::string & name)

Delete a grouped digital output.

Parameters

name	Group name.

Returns

Returns 0 on success; error code on failure.

Exceptions

arcs::common_interface::AuboException	

Python function prototype

deleteGroupOutput(self: pyaubo_sdk.IoControl, arg0: str) -> int

Lua function prototype

deleteGroupOutput(name: string) -> nil

JSON-RPC request example

{"jsonrpc":"2.0","method":"rob1.IoControl.deleteGroupOutput","params":["go_state"],"id":1}

JSON-RPC response example

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

8.6.4.2.8 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}

8.6.4.2.9 getConfigurableDigitalInputAction()

StandardInputAction arcs::common_interface::IoControl::getConfigurableDigitalInputAction (
int index)

Get the trigger action for configurable digital input.

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"}

8.6.4.2.10 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}

8.6.4.2.11 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 10101010101010100000000000000110.
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}

8.6.4.2.12 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}

8.6.4.2.13 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}

8.6.4.2.14 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::Standard

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"}

8.6.4.2.15 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 10101010101010100000000000000110. 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

arcs::common_interface::AuboException	

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}

8.6.4.2.16 getDigitalOutputRunstate()

StandardDigitalOutputRunState arcs::common_interface::IoControl::getDigitalOutputRunstate (
int io_type,
int index)

Generated by Doxygen

Parameters

io_type	Numeric IO type (0=standard, 1=tool, 2=configurable)
index	IO pin index starting from 0

Returns

Returns the corresponding StandardOutputRunState enum value

Exceptions

arcs::common_interface::AuboException	

8.6.4.2.17 getEncDecoderType()

int arcs::common_interface::IoControl::getEncDecoderType ()

Get the decoder type of the encoder.

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.getEncDecoderType","params":[],"id":1}

JSON-RPC response example

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

8.6.4.2.18 getEncTickCount()

Get the tick count

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}

8.6.4.2.19 getGroupInput()

uint32_t arcs::common_interface::IoControl::getGroupInput (
const std::string & name)

Get the grouped digital input value.

Looks up the named grouped digital input, reads the group's members, and combines them into an integer value.

Parameters

name	Group name.

Returns

Returns the combined grouped digital input value.

Exceptions

arcs::common_interface::AuboException	

Python function prototype

getGroupInput(self: pyaubo_sdk.IoControl, arg0: str) -> int

Lua function prototype

getGroupInput(name: string) -> number

Lua example

value = getGroupInput(”gi_state”)

JSON-RPC request example

{”jsonrpc”:”2.0”,”method”:”rob1.IoControl.getGroupInput”,”params”:[”gi_state”],”id”:1}

JSON-RPC response example

{”id”:1,”jsonrpc”:”2.0”,”result”:13}

8.6.4.2.20 getGroupOutput()

uint32_t arcs::common_interface::IoControl::getGroupOutput (
const std::string & name)

Get the grouped digital output value.

Looks up the named grouped digital output, reads the group's members, and combines them into an integer value.

Parameters

name	Group name.

Returns

Returns the combined grouped digital output value.

Exceptions

arcs::common_interface::AuboException	

Python function prototype

getGroupOutput(self: pyaubo_sdk.IoControl, arg0: str) -> int

Lua function prototype

getGroupOutput(name: string) -> number

Lua example

value = getGroupOutput("go_state")

JSON-RPC request example

{"jsonrpc":"2.0","method":"rob1.IoControl.getGroupOutput","params":["go_state"],"id":1}

JSON-RPC response example

{"id":1,"jsonrpc":"2.0","result":13}

8.6.4.2.21 getHandleIoStatus()

uint32_t arcs::common_interface::IoControl::getHandleIoStatus ()

Get the status of handle buttons.

Note

Get the status of handle buttons.

Returns

All handle button input values.
For example, if the return value is 2863267846, its binary representation is 10101010101010100000000000000110.
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

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}

8.6.4.2.22 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}

8.6.4.2.23 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}
```

8.6.4.2.24 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}

8.6.4.2.25 getStandardAnalogInputNum()

int arcs::common_interface::IoControl::getStandardAnalogInputNum ()

Get the number of standard analog inputs.

Returns

Number of standard analog inputs.

Exceptions

arcs::common_interface::AuboException	

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}

8.6.4.2.26 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

arcs::common_interface::AuboException	

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}

8.6.4.2.27 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}

8.6.4.2.28 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}

8.6.4.2.29 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::StandardOu

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"}

8.6.4.2.30 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}

8.6.4.2.31 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"}

8.6.4.2.32 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}

8.6.4.2.33 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 10101010101010100000000000000110.
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

arcs::common_interface::AuboException	

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}

8.6.4.2.34 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}

8.6.4.2.35 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}

8.6.4.2.36 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::StandardOu

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"}

8.6.4.2.37 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 10101010101010100000000000000110. 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}

8.6.4.2.38 getStaticLinkInputNum()

int arcs::common_interface::IoControl::getStaticLinkInputNum ()

Get the number of static link inputs.

Returns

Number of static link inputs.

Exceptions

arcs::common_interface::AuboException	

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}

8.6.4.2.39 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 10101010101010100000000000000110.
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}

8.6.4.2.40 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}

8.6.4.2.41 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 10101010101010100000000000000110.
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}

8.6.4.2.42 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}

8.6.4.2.43 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}

8.6.4.2.44 getToolAnalogInputNum()

int arcs::common_interface::IoControl::getToolAnalogInputNum ()

Get the number of tool analog inputs.

Returns

Number of tool analog inputs.

Exceptions

arcs::common_interface::AuboException	

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}

8.6.4.2.45 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

arcs::common_interface::AuboException	

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}

8.6.4.2.46 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}

8.6.4.2.47 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}

8.6.4.2.48 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::StandardOutput

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"}

8.6.4.2.49 getToolButtonStatus()

bool arcs::common_interface::IoControl::getToolButtonStatus ()

Get the status of the tool button.

Returns

Returns true if pressed; otherwise false.

Exceptions

arcs::common_interface::AuboException	

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}

8.6.4.2.50 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}

8.6.4.2.51 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"}

8.6.4.2.52 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}

8.6.4.2.53 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}

8.6.4.2.54 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}

8.6.4.2.55 getToolDigitalOutputNum()

int arcs::common_interface::IoControl::getToolDigitalOutputNum ()

Get the number of tool digital IOs (including digital inputs and outputs).

Exceptions

arcs::common_interface::AuboException	

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}

8.6.4.2.56 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"}

8.6.4.2.57 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}

8.6.4.2.58 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}

8.6.4.2.59 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}

8.6.4.2.60 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}

8.6.4.2.61 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}

8.6.4.2.62 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}

8.6.4.2.63 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}

8.6.4.2.64 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}

8.6.4.2.65 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}

8.6.4.2.66 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}

8.6.4.2.67 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}

8.6.4.2.68 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}

8.6.4.2.69 setDigitalOutputRunstate()

```
int arcs::common_interface::IoControl::setDigitalOutputRunstate (
                int io_type,
                int index,
                StandardOutputRunState runstate,
                const std::string & object_name,
                double threshold)
```

Set digital output runstate with optional condition checking based on waypoint, plane or obstacle thresholds.

Use this API to control a digital output when a specified condition is met (for example, waypoint arrival or distance to a plane/obstacle). Specify an object type, name and a threshold value to enable conditional triggering. When the condition is satisfied, the given digital output will be set to the requested runstate.

Parameters

io_type	Numeric IO type (e.g. 0=standard, 1=tool, 2=configurable)
index	IO pin index starting from 0
runstate	The desired output action
object_name	Unique identifier of the condition object (optional)
threshold	Condition threshold: angle in radians for waypoints, or distance in meters for planes/obstacles (optional)

Returns

Returns 0 on success, non-zero error code on failure.

8.6.4.2.70 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 (StandardOutputRun↩ State::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}

8.6.4.2.71 setEncDecoderType()

int arcs::common_interface::IoControl::setEncDecoderType (
int type

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	

8.6.4.2.72 setEncTickCount()

int arcs::common_interface::IoControl::setEncTickCount (
int tick)

Set the tick count of the integrated encoder.

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	

8.6.4.2.73 setGroupOutput()

int arcs::common_interface::IoControl::setGroupOutput (

Parameters

name	Group name.
value	Integer value to write to the grouped digital output.

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

setGroupOutput(self: pyaubo_sdk.IoControl, arg0: str, arg1: int) -> int

Lua function prototype

setGroupOutput(name: string, value: number) -> nil

Lua example

setGroupOutput("go_state", 13)

JSON-RPC request example

{"jsonrpc":"2.0","method":"rob1.IoControl.setGroupOutput","params":["go_state",13],"id":1}

JSON-RPC response example

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

8.6.4.2.74 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}

8.6.4.2.75 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)

8.6.4.2.76 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)

8.6.4.2.77 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}

8.6.4.2.78 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 output value.
When the output runstate is set, users cannot set the analog output value, and the controller will set it automatically.
For example, when AO0's output runstate is set to indicate hand-guiding (StandardOutputRun↩ State::Handguiding), the robot enters hand-guiding mode and AO0 will automatically become the maximum value. When the robot exits hand-guiding, AO0 will automatically become 0.

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::Standard
-> 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}

8.6.4.2.79 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 value to high will not trigger any robot action.
When a trigger action is set, setting the digital input value to high will trigger the corresponding robot action.
For example, if DI0 is set to trigger Handguide (StandardInputAction::Handguide), setting DI0 to high will enable hand-guiding mode. Setting DI0 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

setStandardDigitalInputAction(self: pyaubo_sdk.IoControl, arg0: int, arg1: arcs::common_interface::StandardInpu -> 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}

8.6.4.2.80 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

arcs::common_interface::AuboException	

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}

8.6.4.2.81 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}

8.6.4.2.82 setStandardDigitalOutputPulse()

int arcs::common_interface::IoControl::setStandardDigitalOutputPulse (
int index,
bool value,
double duration)

Set 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

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}

8.6.4.2.83 setStandardDigitalOutputRunstate()

int arcs::common_interface::IoControl::setStandardDigitalOutputRunstate (
int index,
StandardOutputRunState runstate)

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::Standard
-> 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}

8.6.4.2.84 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

arcs::common_interface::AuboException	

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}

8.6.4.2.85 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}

8.6.4.2.86 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)

8.6.4.2.87 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

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::StandardOutp
-> 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}

8.6.4.2.88 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 (StandardInput↩
Action::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}

8.6.4.2.89 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}

8.6.4.2.90 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}

8.6.4.2.91 setToolDigitalOutputRunstate()

int arcs::common_interface::IoControl::setToolDigitalOutputRunstate (
int index,
StandardOutputRunState runstate)

Set the runstate for tool 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

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}

8.6.4.2.92 setToolIoInput()

int arcs::common_interface::IoControl::setToolIoInput (
int index,
bool input)

Set the specified tool digital IO as input or output.

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

arcs::common_interface::AuboException	

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}

8.6.4.2.93 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}

8.6.4.2.94 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::getEqradius ()
  Get the equivalent radius, in meters.
- int arcs::common_interface::MotionControl::setEqradius (double eqradius)
  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.
- int arcs::common_interface::MotionControl::speedFractionCritical1 (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 moveJoint/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 &param)

  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 &params)

  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 > &center, 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.
- int arcs::common_interface::MotionControl::setConveyorTrackVelThreshold (int encoder_id, double threshold)

  Set the conveyor velocity threshold used to suppress position jumps caused by abnormal encoder spikes.
- 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 &param, 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 &params)

  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)
- int arcs::common_interface::MotionControl::getWeaveDirection ()

  Get current weave trajectory direction

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)

Note

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()

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]],"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]],"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 movePathBuffer motion and the queue in the planner (such as moveJoint and moveLine) 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).

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

arcs::common_interface::AuboException	

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

arcs::common_interface::AuboException	

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

arcs::common_interface::AuboException	

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

arcs::common_interface::AuboException	

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 getWeaveDirection()

int arcs::common_interface::MotionControl::getWeaveDirection ()

Get current weave trajectory direction

Retrieves the directional state of the current trajectory point during weave motion, indicating whether the point is located on the positive or negative side of the weave coordinate system Y-axis.

When no weave trajectory is active, this interface returns 0.

This is a state query interface and does not affect trajectory execution or robot motion.

Returns

Returns current weave direction corresponding to WeaveDirection enum:

- 0 : NONE No active weave trajectory
- 1 : POSITIVE Positive Y-axis side
- 2 : NEGATIVE Negative Y-axis side

Exceptions

arcs::common_interface::AuboException	

Lua function prototype

getWeaveDirection() -> number

Lua example

local dir = getWeaveDirection()

Python function prototype

getWeaveDirection() -> int

Python example

dir = getWeaveDirection()

JSON-RPC request example

{"jsonrpc":"2.0","method":"rob1.MotionControl.getWeaveDirection","params":[],"id":1}

JSON-RPC response example

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

8.6.5.2.30 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":["",[]]}

8.6.5.2.31 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}}

8.6.5.2.32 isBlending()

bool arcs::common_interface::MotionControl::isBlending ()

Whether it is in the blending area

Returns

Returns true if in the blending area; otherwise returns false

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}

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

arcs::common_interface::AuboException	

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

arcs::common_interface::AuboException	

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}

JSON-RPC response example

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

8.6.5.2.41 jointOffsetSet()

int arcs::common_interface::MotionControl::jointOffsetSet (
const std::vector< double > & offset,
int type = 1)

Set joint offset

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”:[[0.1,0,0,0,0,0],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)
```


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},1.2,0.25,0,0,0,0,0,0)

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)
```


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()


```
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

Returns

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)

Spiral motion

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]},0,0.25,1.2,0],"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,
        int size)
```


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

arcs::common_interface::AuboException	

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

arcs::common_interface::AuboException	

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 0 - Disable monitoring 1 - Saturation limit, i.e., maintain maximum pose 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

arcs::common_interface::AuboException	

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

arcs::common_interface::AuboException	

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.

Returns

Returns 0 on success; otherwise returns error code -AUBO_INVL_ARGUMENT -AUBO_BAD↩ _STATE

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	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_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 (

const std::vector< double > & qd,
double a,
double t)

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.↩
5,100],"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 (
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)
```


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 out of range, velocity out of range, or adjacent points angle difference exceeded(±15°)
-AUBO_REQUEST_IGNORE(-13)	Not in servo mode
-AUBO_POS_BOUND_ERR(-18)	Joint position exceeds robot maximum limits
-AUBO_TRAJ_SELF_COLLISION(-22)	Trajectory has self-collision risk
-AUBO_ALGORITHM_PLAN_FAILED(-30)	Trajectory planning failed

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	Mode 0: Tool orientation remains unchanged relative to the arc path point coordinate system 1: Orientation changes linearly, rotating around a fixed axis in space, from the start orientation to the target orientation 2: Orientation changes from the start orientation, through the via point orientation, to the target orientation

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

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

arcs::common_interface::AuboException	

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

arcs::common_interface::AuboException	

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 setConveyorTrackVelThreshold()

int arcs::common_interface::MotionControl::setConveyorTrackVelThreshold (
int encoder_id,
double threshold)

Set the conveyor velocity threshold used to suppress position jumps caused by abnormal encoder spikes.

Parameters

encoder_id	Reserved
threshold	The maximum allowable conveyor speed during operation, unit: meter/second. Default: 0.2 m/s.

Returns

0 on success; otherwise a non-zero error code.

Exceptions

arcs::common_interface::AuboException	

Python function prototype

setConveyorTrackVelThreshold(self: pyaubo_sdk.MotionControl, arg0: int, arg1: double) -> int

Lua function prototype

setConveyorTrackVelThreshold(encoder_id: number, threshold: number) -> int

JSON-RPC request example

{"jsonrpc":"2.0","method":"rob1.MotionControl.setConveyorTrackVelThreshold","params":[0, 0.↩
2],"id":1}

JSON-RPC response example

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

8.6.5.2.87 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}

8.6.5.2.88 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

arcs::common_interface::AuboException	

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)

8.6.5.2.89 setFuturePointSamplePeriod()

int arcs::common_interface::MotionControl::setFuturePointSamplePeriod (
double sample_time)

Set the sampling interval for points on the future path

Parameters

sample_time	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	

8.6.5.2.90 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}

8.6.5.2.91 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}

8.6.5.2.92 setServoMode()

ARCS_DEPRECATED int arcs::common_interface::MotionControl::setServoMode (

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}

8.6.5.2.93 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}

8.6.5.2.94 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}

8.6.5.2.95 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}

8.6.5.2.96 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}

8.6.5.2.97 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})

8.6.5.2.98 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() -> int

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}

8.6.5.2.99 speedFractionCritical1()

int arcs::common_interface::MotionControl::speedFractionCritical1 (
bool enable)

Speed fraction critical section.

When enabled, the speed fraction is forced to the currently effective speed fraction. When disabled, the previous speed fraction is restored.

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

speedFractionCritical1() -> int

Lua example

speedFractionCritical1(true)

JSON-RPC request example

{"jsonrpc":"2.0","method":"rob1.MotionControl.speedFractionCritical1","params":[true],"id":1}

JSON-RPC response example

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

8.6.5.2.100 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}

8.6.5.2.101 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({0.25,0,0,0,0,0},1.2,100)

JSON-RPC request example

{”jsonrpc”:”2.0”,”method”:”rob1.MotionControl.speedLine”,”params”:[[0.25,0,0,0,0,0],1.2,100]↩
,”id”:1}

JSON-RPC response example

{”id”:1,”jsonrpc”:”2.0”,”result”:0}

8.6.5.2.102 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}

8.6.5.2.104 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}

8.6.5.2.105 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.

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)

8.6.5.2.106 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()

8.6.5.2.107 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}

8.6.5.2.108 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}

8.6.5.2.109 weaveEnd()

```
int arcs::common_interface::MotionControl::weaveEnd ()
```

End weaving

Returns

Returns 0 on success; otherwise returns an error code AUBO_BUSY AUBO_BAD_STATE - 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}

8.6.5.2.110 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 ↩ [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 (m · 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

arcs::common_interface::AuboException	

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,\"dire
```

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],\"
```

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_di
```

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
```

JSON-RPC response example

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


8.6.5.2.111 weaveUpdateParameters()

int arcs::common_interface::MotionControl::weaveUpdateParameters (

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

- ResultWithErrno arcs::common_interface::RobotAlgorithm::inverseKinematics2 (const std::vector< double > &qnear, const std::vector< double > &pose)

  Inverse kinematics with the same robot configuration and minimum joint change Calculate the inverse kinematics solution for the target TCP pose according to the robot configuration of qnear, and select the solution with the minimum joint change relative to qnear among solutions with the same configuration.

- 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 > > &qs, 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 &param1, const VibrationRecalibrationParameter &param2, double threshold)
- int arcs::common_interface::RobotAlgorithm::addCollisionBox (const std::string &name, const std::string &link_name, const std::vector< std::vector< double > > &sizes, const std::vector< std::vector< double > > &poses)

  Add a box collision object
- int arcs::common_interface::RobotAlgorithm::removeCollisionObject (const std::string &name)

  Remove a collision object by name
- 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 without external collision checks.
- int arcs::common_interface::RobotAlgorithm::validatePath1 (int type, const std::vector< double > &start, double r1, const std::vector< double > &end, double r2, double d, bool check_external_↩ collision)

  Validate the reachability of the robot's motion path from start point to end point, with optional external collision checks.

8.6.6.1 Detailed Description

Interfaces related to robot algorithms

8.6.6.2 Function Documentation

8.6.6.2.1 addCollisionBox()

```
int arcs::common_interface::RobotAlgorithm::addCollisionBox (
                const std::string & name,
                const std::string & link_name,
                const std::vector< std::vector< double > > & sizes,
                const std::vector< std::vector< double > > & poses)
```

Add a box collision object

This interface adds a named group of box collision objects into the current robot algorithm scene.

Parameters

name	Unique identifier of the collision object
link_name	Name of the reference link the collision object is attached to. Common names include: world, base_link, shoulder_Link, upperArm_Link, foreArm_Link, wrist1_Link, wrist2_Link, wrist3_Link, and end_effector
sizes	List of box sizes. Each element is in the format [length, width, height], unit: m
poses	List of collision object poses. Each element is a pose relative to the reference link in the format [x, y, z, rx, ry, rz]

Returns

0 on success, non-zero on failure

Lua function prototype

addCollisionBox(name, link_name, sizes, poses) -> integer

Lua example

– Add an environment collision box attached to world addCollisionBox(”env_box”, ”world”, {{0.4, 0.4, 0.2}}, {{0.6, 0.0, 0.1, 0.0, 0.0, 0.0}})

– Add a tool collision box attached to end_effector addCollisionBox(”tool_box”, ”end_effector”, {{0.1, 0.08, 0.06}}, {{0.0, 0.0, 0.05, 0.0, 0.0, 0.0}})

Python function prototype

addCollisionBox(name: str, link_name: str, sizes: list[list[float]], poses: list[list[float]]) -> int

Python example

Add an environment collision box attached to world

robot_algorithm.addCollisionBox(”env_box”, ”world”, [[0.4, 0.4, 0.2]], [[0.6, 0.0, 0.1, 0.0, 0.0, 0.0]])

Add a tool collision box attached to end_effector

robot_algorithm.addCollisionBox(”tool_box”, ”end_effector”, [[0.1, 0.08, 0.06]], [[0.0, 0.0, 0.05, 0.0, 0.0, 0.0]])

JSON-RPC request example

{”jsonrpc”:”2.0”,”method”:”rob1.RobotAlgorithm.addCollisionBox”, ”params”:[”env_box”,”world”,↩ [[0.4,0.4,0.2]], [[0.6,0.0,0.1,0.0,0.0,0.0]]],”id”:1}

JSON-RPC response example

{”id”:1,”jsonrpc”:”2.0”,”result”:0}

8.6.6.2.2 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]}

8.6.6.2.3 calibrateTcpForceSensor()

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)

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})

8.6.6.2.4 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	

8.6.6.2.5 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	

8.6.6.2.6 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)

8.6.6.2.7 calibVibrationParams1()

```
ResultWithErrno1 arcs::common_interface::RobotAlgorithm::calibVibrationParams1 (
        const std::string & record_cache_name,
        const std::vector< double > & tool_offset)
```

8.6.6.2.8 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.↩
397},1)
```


8.6.6.2.9 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})

8.6.6.2.10 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})

8.6.6.2.11 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]}

8.6.6.2.12 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

8.6.6.2.13 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,-1.571961859294079 8], [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]}

8.6.6.2.14 forwardToolKinematics()

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]}

8.6.6.2.15 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})

8.6.6.2.16 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 abs(traj_conf.init_joint[4]) >= 0.3(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	

8.6.6.2.17 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,[],[],[]]}

8.6.6.2.18 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]}

8.6.6.2.19 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]}

8.6.6.2.20 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]}

8.6.6.2.21 inverseKinematics2()

ResultWithErrno arcs::common_interface::RobotAlgorithm::inverseKinematics2 (
const std::vector< double > & qnear,
const std::vector< double > & pose)

Inverse kinematics with the same robot configuration and minimum joint change Calculate the inverse kinematics solution for the target TCP pose according to the robot configuration of qnear, and select the solution with the minimum joint change relative to qnear among solutions with the same configuration.

This interface is based on the activated TCP offset, which is the latest parameter set by setTcpOffset.

Parameters

qnear	Reference joint angles

pose	TCP pose, based on the activated TCP offset, which is the latest parameter set by setTcpOffset

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 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 same-configuration minimum-joint-change 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

inverseKinematics2(self: pyaubo_sdk.RobotAlgorithm, arg0: List[float], arg1: List[float]) -> Tuple[List[float], int]

Lua function prototype

inverseKinematics2(qnear: table, pose: table) -> table, int

Lua example

joint,ik_result = inverseKinematics2({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.inverseKinematics2","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]}

8.6.6.2.22 inverseKinematicsAll()

ResultWithErrno1 arcs::common_interface::RobotAlgorithm::inverseKinematicsAll (
const std::vector< double > & pose)

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,-
0.1623365054641728,1.081775890307679,0.2699325026322 4805,0.9738255833642309,0.00057255627720845]↩
, [0.4116449425067969,-1.1931664523907126,-1.0822709833775688,-0.8665964106161371,0.9732141569888207,0.↩
006484919654891586]],0]}

8.6.6.2.23 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]}

8.6.6.2.24 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

arcs::common_interface::AuboException	

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]}

8.6.6.2.25 inverseToolKinematicsAll()

ResultWithErrno1 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.↩
8665312056504818,0.9732632365861417,0.006395831160177175]],0]}

8.6.6.2.26 needVibrationRecalib()

```
int arcs::common_interface::RobotAlgorithm::needVibrationRecalib (
        const VibrationRecalibrationParameter & param1,
        const VibrationRecalibrationParameter & param2,
        double threshold)
```

8.6.6.2.27 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 movel 2-movel and movej 3-movel and movel
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)

8.6.6.2.28 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	

8.6.6.2.29 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}

8.6.6.2.30 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

8.6.6.2.31 payloadIdentify1()

```
int arcs::common_interface::RobotAlgorithm::payloadIdentify1 (
        const std::string & file_name)
```

New version of payload identification algorithm interface based on current

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

8.6.6.2.32 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}

8.6.6.2.33 removeCollisionObject()

int arcs::common_interface::RobotAlgorithm::removeCollisionObject (
const std::string & name)

Remove a collision object by name

Parameters

name	Name of the collision object to remove

Returns

0 on success, non-zero on failure

Lua function prototype

removeCollisionObject(name) -> integer

Lua example

removeCollisionObject(”tool_box”)

Python function prototype

removeCollisionObject(name: str) -> int

Python example

robot_algorithm.removeCollisionObject(”tool_box”)

JSON-RPC request example

{”jsonrpc”:”2.0”,”method”:”rob1.RobotAlgorithm.removeCollisionObject”, ”params”:[”tool_↩ box”],”id”:1}

JSON-RPC response example

{”id”:1,”jsonrpc”:”2.0”,”result”:0}

8.6.6.2.34 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 without external collision checks.

This interface is kept for compatibility with existing SDK, Lua, JSON-RPC, and ROS service calls.

Parameters

type	Path type identifier, used to specify the data types of start and end: 0 - Start: Joint angles, End: Joint angles 1 - Start: Pose, End: Pose
start	Start point data of the path
r1	Start point blending radius, unit: m (meters)
end	End point data of the path
r2	End point blending radius, unit: m (meters)
d	Path sampling interval, unit: m (meters). A smaller interval improves validation accuracy but takes more time.

Returns

Path reachability result code

8.6.6.2.35 validatePath1()

```
int arcs::common_interface::RobotAlgorithm::validatePath1 (
                int type,
                const std::vector< double > & start,
                double r1,
                const std::vector< double > & end,
                double r2,
                double d,
                bool check_external_collision)
```

Validate the reachability of the robot's motion path from start point to end point, with optional external collision checks.

This interface verifies whether the specified path has unreachable conditions such as out-of-limit, self-collision, collision with external objects, singularity, etc., by sampling. It supports two path type validations: Joint-Joint and Pose-Pose.

Parameters

type	Path type identifier, used to specify the data types of start and end: 0 - Start: Joint angles, End: Joint angles 1 - Start: Pose, End: Pose
start	Start point data of the path
r1	Start point blending radius, unit: m (meters)
end	End point data of the path
r2	End point blending radius, unit: m (meters)
d	Path sampling interval, unit: m (meters). A smaller interval improves validation accuracy but takes more time.

check_external_collision	Whether to enable external collision checks: false - Keep the original reachability and self-collision checks true - Add collision checks against external objects

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

Exceptions

arcs::common_interface::AuboException	Thrown when parameters are invalid.

Lua function prototype

validatePath1(type, start, r1, end, r2, d, check_external_collision) -> integer

Lua example

result = validatePath1(0, start_joint, 0.01, end_joint, 0.01, 0.05, true)

Python function prototype

validatePath1(type: int, start: list[float], r1: float, end: list[float], r2: float, d: float, check_↩ external_collision: bool) -> int

Python example

result = validatePath1(1, start_joint, 0.01, end_pose, 0.01, 0.005, True)

JSON-RPC request example

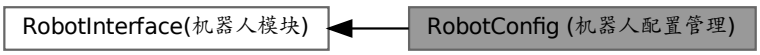
{"jsonrpc":"2.0","method":"rob1.RobotAlgorithm.validatePath1","params":[0, [0.0,0.0,1.57,0.0,1.↩ 57,0.0], 0.01, [0.52,0.0,1.57,0.0,1.57,0.0], 0.01, 0.005, true],"id":1}

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::getKinematicsPa (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::getKinematicsCo (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 > > &param)

  Set standard DH compensation to the robot.
- int arcs::common_interface::RobotConfig::setPersistentParameters (const std::string &param)

  Set parameters to be saved to the interface board base.
- int arcs::common_interface::RobotConfig::setHardwareCustomParameters (const std::string &param)

  Set custom parameters for the hardware abstraction layer.
- std::string arcs::common_interface::RobotConfig::getHardwareCustomParameters (const std::↩ string &param)

  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 &parameters)

  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 RobotSafetyParameterRang &parameters)

  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 > &inertia)

  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).
- SafeguedStopType 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 > &inertia)
- 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

arcs::common_interface::AuboException	

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}

JSON-RPC response example

{"id":1,"jsonrpc":"2.0","result":"None"}

8.6.7.2.6 firmwareUpdate()

int arcs::common_interface::RobotConfig::firmwareUpdate (
const std::string & fw)

Initiate a firmware upgrade request.

The controller software will enter firmware upgrade mode.

Parameters

fw	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, /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. /tmp/firm_XXX.firm#all means upgrade all nodes with version check, /tmp/firm_XXX.firm#all! means force upgrade all nodes.

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}

8.6.7.2.7 getBaseForceOffset()

std::vector< double > arcs::common_interface::RobotConfig::getBaseForceOffset ()

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

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_ISStation"}

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/s^2.

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/s^2.

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

handguuiding 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).

Returns

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":[3.892084231947355,3.892084231947355,3.892084231947355,3.1066860685499065,3.1066860685499065,3.1066860685499065]}

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

arcs::common_interface::AuboException	

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::getKinematics↩
Compensate (
double ref_temperature)

Get DH parameter compensation values (alpha, a, d, theta, beta) at the specified temperature.

Parameters

ref_temperature	Reference temperature in ℃, default is 20℃

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":[0.0,0.000678930024150759,0.00243821996264159 7,0.0049148499965667725,0.00048695001169107854,0.004321440123021603], "beta":[0.0,0.0,0.0,0.0,0.0,0.0,0.0],"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,-1.5701173967707458,3.1440308735524347,3.14650750358636,-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

arcs::common_interface::AuboException	

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

arcs::common_interface::AuboException	

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.↩
1066861152648926,3.1066861152648926,3.1066861152648926]}

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∗m^2, format (Ixx, Iyy, Izz, Ixy, Ixz, Iyz)

Exceptions

arcs::common_interface::AuboException	

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],[0.0,0.0,0.0,0.0,0.0,0.0,0.0]]}

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

arcs::common_interface::AuboException	

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

arcs::common_interface::AuboException	

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

Lua function prototype

selectBaseForceSensor(name: string) -> nil

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})

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, 60.0,54.6111111, 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"}

8.6.7.2.62 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]],”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})

JSON-RPC request example

{”jsonrpc”:”2.0”,”method”:”rob1.RobotConfig.setMountingPose”,”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}

8.6.7.2.69 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_INVL↩
_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_INVL_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

arcs::common_interface::AuboException	

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})

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}, {0,0,0,0,0,0,0,0,0})

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}

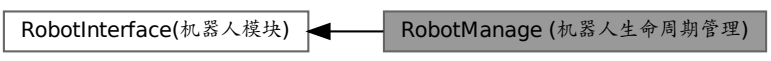
JSON-RPC response example

{"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 handguide↩ Mode instead: handguideMode({1,1,1,1,1}, {0,0,0,0,0,0})
- int arcs::common_interface::RobotManage::setHandguideParams (const std::vector< int > &free↩ Axes, 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

Exceptions

arcs::common_interface::AuboException	

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}, {0,0,0,0,0,0})

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 =
rpc_cli->getRobotInterface(robot_name)->getRobotManage()->getOperationalMode();
```


8.6.8.2.10 getRecordCache()

Parameters

name	Cache name
frames	Frame limit: • 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();
```


8.6.8.2.14 isFreedriveEnabled()

Exceptions

arcs::common_interface::AuboException	

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

Returns

Returns 0 on success; negative error code on failure

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

arcs::common_interface::AuboException	

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)

Free and clear recorded data of the specified memory cache

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)
```

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 (
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)

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 ()

Clear protective stop, including collision stop

Returns

Exceptions

arcs::common_interface::AuboException	

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

file_name	

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	

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")

8.6.8.2.33 startup()

int arcs::common_interface::RobotManage::startup ()

Exceptions

arcs::common_interface::AuboException	

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

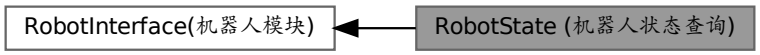
Lua example

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::getElbowPosistion ()
  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::getHandleUniqueId ()
  Get the globally unique ID of the Handle
- int arcs::common_interface::RobotState::getHandleFirmwareVersion ()
  Get the Handle firmware version
- int arcs::common_interface::RobotState::getHandleHardwareVersion ()

Get the Handle hardware version

- HandleStateType arcs::common_interface::RobotState::getHandleState ()
  Get the Handle state
- HandleModeType arcs::common_interface::RobotState::getHandleMode ()
  Get the Handle mode
- std::string arcs::common_interface::RobotState::getPowerlossBoardUniqueId ()
  Get the globally unique ID of the PowerlossBoard
- int arcs::common_interface::RobotState::getPowerlossBoardFirmwareVersion ()
  Get the PowerlossBoard firmware version
- int arcs::common_interface::RobotState::getPowerlossBoardHardwareVersion ()
  Get the PowerlossBoard 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

- double arcs::common_interface::RobotState::getPedestalBrakeResTemp ()
  Get the Pedestal Brake Res Temp
- 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 getHandleFirmwareVersion()

int arcs::common_interface::RobotState::getHandleFirmwareVersion ()

Get the Handle firmware version

Returns

Handle firmware version

Exceptions

arcs::common_interface::AuboException	

Python function prototype

getHandleFirmwareVersion(self: pyaubo_sdk.RobotState) -> int

Lua function prototype

getHandleFirmwareVersion() -> number

Lua example

HandleFirmwareVersion = getHandleFirmwareVersion()

JSON-RPC request example

{”jsonrpc”:”2.0”,”method”:”rob1.RobotState.getHandleFirmwareVersion”,”params”:[],”id”:1}

JSON-RPC response example

Exceptions

arcs::common_interface::AuboException	

Python function prototype

getHandleHardwareVersion(self: pyaubo_sdk.RobotState) -> int

Lua function prototype

getHandleHardwareVersion() -> number

Lua example

HandleHardwareVersion = getHandleHardwareVersion()

JSON-RPC request example

{"jsonrpc":"2.0","method":"rob1.RobotState.getHandleHardwareVersion","params":[],"id":1}

JSON-RPC response example

{"id":1,"jsonrpc":"2.0","result":6030098}

8.6.9.2.9 getHandleMode()

HandleModeType arcs::common_interface::RobotState::getHandleMode ()

Get the Handle mode

Returns

Handle mode

Exceptions

arcs::common_interface::AuboException	

Python function prototype

getHandleMode(self: pyaubo_sdk.RobotState) -> int

Lua function prototype

getHandleMode() -> number

Lua example

HandleMode = getHandleMode()

JSON-RPC request example

{"jsonrpc":"2.0","method":"rob1.RobotState.getHandleMode","params":[],"id":1}

JSON-RPC response example

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

8.6.9.2.10 getHandleState()

HandleStateType arcs::common_interface::RobotState::getHandleState ()

Get the Handle state

Returns

Handle state

Exceptions

arcs::common_interface::AuboException	

Python function prototype

getHandleState(self: pyaubo_sdk.RobotState) -> int

Lua function prototype

getHandleState() -> number

Lua example

HandleState = getHandleState()

JSON-RPC request example

{"jsonrpc":"2.0","method":"rob1.RobotState.getHandleState","params":[],"id":1}

JSON-RPC response example

{"id":1,"jsonrpc":"2.0","result":6030098}

8.6.9.2.11 getHandleUniqueId()

std::string arcs::common_interface::RobotState::getHandleUniqueId ()

Get the globally unique ID of the Handle

Returns

Globally unique ID of the Handle

Exceptions

arcs::common_interface::AuboException	

Python function prototype

getHandleUniqueId(self: pyaubo_sdk.RobotState) -> str

Lua function prototype

getHandleUniqueId() -> string

Lua example

HandleUniqueId = getHandleUniqueId()

JSON-RPC request example

{"jsonrpc":"2.0","method":"rob1.RobotState.getHandleUniqueId","params":[],"id":1}

JSON-RPC response example

{"id":1,"jsonrpc":"2.0","result":"73657263008000000000000"}

8.6.9.2.12 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]}

8.6.9.2.13 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]}

8.6.9.2.14 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]}

8.6.9.2.15 getJointFirmwareVersions()

std::vector< int > arcs::common_interface::RobotState::getJointFirmwareVersions ()

Get the joint firmware versions

Returns

Joint firmware versions

Exceptions

arcs::common_interface::AuboException	

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]}

8.6.9.2.16 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

arcs::common_interface::AuboException	

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]}

8.6.9.2.17 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]}

8.6.9.2.18 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

Parameters

steps	Number of steps

Returns

Historical joint positions of the manipulator

8.6.9.2.20 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"]↩
}

8.6.9.2.21 getJointSpeeds()

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]}

8.6.9.2.22 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"]↩
}

8.6.9.2.23 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]}

8.6.9.2.24 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]}

8.6.9.2.25 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]}

8.6.9.2.26 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":[0.0,0.0002422710150939977 3,0.0016521760307419697,0.0026521060731088397,0.0,0.0]}

8.6.9.2.27 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

arcs::common_interface::AuboException	

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]}

8.6.9.2.28 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]}

8.6.9.2.29 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]}

8.6.9.2.30 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":["00800020ffffffff31405153","00800020ffffffff3e3f5153","00800020ffffffff414b5153","00
}

8.6.9.2.31 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]}

8.6.9.2.32 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}

8.6.9.2.33 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}

8.6.9.2.34 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}

8.6.9.2.35 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}

8.6.9.2.36 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"}

8.6.9.2.37 getPedestalBrakeResTemp()

double arcs::common_interface::RobotState::getPedestalBrakeResTemp ()

Get the Pedestal Brake Res Temp

Returns

Pedestal Brake Res Temp

Exceptions

arcs::common_interface::AuboException	

Python function prototype

getPedestalBrakeResTemp(self: pyaubo_sdk.RobotState) -> float

Lua function prototype

getPedestalBrakeResTemp() -> number

Lua example

PedestalBrakeResTemp = getPedestalBrakeResTemp()

JSON-RPC request example

{"jsonrpc":"2.0","method":"rob1.RobotState.getPedestalBrakeResTemp","params":[],"id":1}

JSON-RPC response example

{"id":1,"jsonrpc":"2.0","result":24.235}

8.6.9.2.38 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}

8.6.9.2.39 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}

8.6.9.2.40 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":"20525753354306524854433 9"}

8.6.9.2.41 getPowerlossBoardFirmwareVersion()

int arcs::common_interface::RobotState::getPowerlossBoardFirmwareVersion ()

Get the PowerlossBoard firmware version

Returns

PowerlossBoard firmware version

Exceptions

arcs::common_interface::AuboException	

Python function prototype

getPowerlossBoardFirmwareVersion(self: pyaubo_sdk.RobotState) -> int

Lua function prototype

getPowerlossBoardFirmwareVersion() -> number

Lua example

PowerlossBoardFirmwareVersion = getPowerlossBoardFirmwareVersion()

JSON-RPC request example

{"jsonrpc":"2.0","method":"rob1.RobotState.getPowerlossBoardFirmwareVersion","params":[]↩
,"id":1}

JSON-RPC response example

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

8.6.9.2.42 getPowerlossBoardHardwareVersion()

int arcs::common_interface::RobotState::getPowerlossBoardHardwareVersion ()

Get the PowerlossBoard hardware version

Returns

PowerlossBoard hardware version

Exceptions

arcs::common_interface::AuboException	

Python function prototype

getPowerlossBoardHardwareVersion(self: pyaubo_sdk.RobotState) -> int

Lua function prototype

getPowerlossBoardHardwareVersion() -> number

Lua example

PowerlossBoardHardwareVersion = getPowerlossBoardHardwareVersion()

JSON-RPC request example

{"jsonrpc":"2.0","method":"rob1.RobotState.getPowerlossBoardHardwareVersion","params":[]↩
,"id":1}

JSON-RPC response example

{"id":1,"jsonrpc":"2.0","result":6030098}

8.6.9.2.43 getPowerlossBoardUniqueId()

std::string arcs::common_interface::RobotState::getPowerlossBoardUniqueId ()

Get the globally unique ID of the PowerlossBoard

Returns

Globally unique ID of the PowerlossBoard

Exceptions

arcs::common_interface::AuboException	

Python function prototype

getPowerlossBoardUniqueId(self: pyaubo_sdk.RobotState) -> str

Lua function prototype

getPowerlossBoardUniqueId() -> string

Lua example

PowerlossBoardUniqueId = getPowerlossBoardUniqueId()

JSON-RPC request example

{"jsonrpc":"2.0","method":"rob1.RobotState.getPowerlossBoardUniqueId","params":[],"id":1}

JSON-RPC response example

{"id":1,"jsonrpc":"2.0","result":"736572630080000000000000"}

8.6.9.2.44 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}

8.6.9.2.45 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"}

8.6.9.2.46 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}

8.6.9.2.47 getSafetyModeType()

SafetyModeType arcs::common_interface::RobotState::getSafetyModeType ()

Get the safety mode

Returns

The safety mode

Exceptions

arcs::common_interface::AuboException	

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"}

8.6.9.2.48 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}

8.6.9.2.49 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}

8.6.9.2.50 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”:”73657263008000000000000”}

8.6.9.2.51 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}

8.6.9.2.52 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

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]}

8.6.9.2.53 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]}

8.6.9.2.54 getTcpForceSensors()

std::vector< double > arcs::common_interface::RobotState::getTcpForceSensors ()

Get the TCP force sensor readings

Returns

TCP force sensor readings

Exceptions

arcs::common_interface::AuboException	

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]}

8.6.9.2.55 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}

8.6.9.2.56 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,-0.13250000000000012,0.20928451364415995,3.↩
1415792312578987,0.0,1.5707963267948963]}

8.6.9.2.57 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]}

8.6.9.2.58 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]}

8.6.9.2.59 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,-0.13250000000000012,0.2092832117232601,3.↩
1415812372223217,0.0,1.5707963267948963]}

8.6.9.2.60 getTcpTargetSpeed()

std::vector< double > arcs::common_interface::RobotState::getTcpTargetSpeed ()

Get the TCP target speed

Returns

TCP target speed

Exceptions

arcs::common_interface::AuboException	

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]}

8.6.9.2.61 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}

8.6.9.2.62 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}

8.6.9.2.63 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}

8.6.9.2.64 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,-0.13250000000000012,0.20928410288421018,3.↩
141579231257899,0.0,1.5707963267948963]}

8.6.9.2.65 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"}

8.6.9.2.66 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}

8.6.9.2.67 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}

8.6.9.2.68 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}

8.6.9.2.69 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}

8.6.9.2.70 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}

8.6.9.2.71 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

{"jsonrpc":"2.0","method":"RuntimeMachine.deleteTask","params":[26],"id":1}

JSON-RPC Response Example

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

8.7.2.6 detachTask()

int arcs::common_interface::RuntimeMachine::detachTask (
int tid)

Wait for the task to finish naturally

Parameters

tid	

Returns

JSON-RPC Request Example

{"jsonrpc":"2.0","method":"RuntimeMachine.detachTask","params":[26],"id":1}

JSON-RPC Response Example

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}

JSON-RPC Response Example

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

8.7.2.8 exitCritical()

int arcs::common_interface::RuntimeMachine::exitCritical ()

Exit the critical section

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)

Get the pointer of the most recently interpreted instruction

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}

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()

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

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 (
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}

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}

8.7.2.22 loadProgram()

int arcs::common_interface::RuntimeMachine::loadProgram (
const std::string & program)

Load a local project file.

For Lua scripts, only the file name is required (no extension), and it will be searched in the ${ARCS_↩ WS}/program directory.

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()

int arcs::common_interface::RuntimeMachine::removeBreakPoint (
int lineno)

Remove a breakpoint

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()

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}

8.7.2.31 setLabel()

```
int arcs::common_interface::RuntimeMachine::setLabel (
        int lineno,
        const std::string & comment)
```

Mark the line number and comment of the recorded instruction

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.

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.

8.10.2.7 axisGroupGetTargetPositions()

std::vector< double > arcs::common_interface::SyncMove::axisGroupGetTargetPositions (

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)
```


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.

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.

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.

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 @endcoe

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 @endcoe

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 @endcoe

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.peek","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 responose example

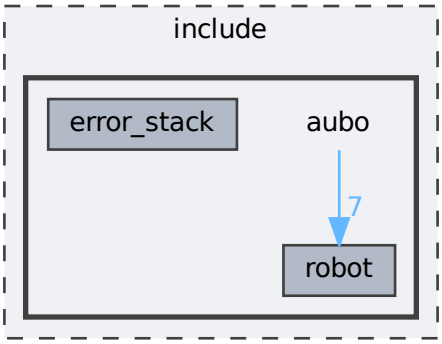
{”id”:1,”jsonrpc”:”2.0”,”result”:0}

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

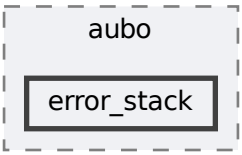
  API for controlling the robot and external axis.
- file axis_interface.h
- file gripper_interface.h

- 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 breakpoints.
- 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

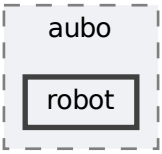
Directory dependency graph for include:

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 HandleStateType : int { ENUM_HandleStateType_DECLARES }
- enum class HandleModeType : int { ENUM_HandleModeType_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 SafeguedStopType : int { ENUM_SafeguedStopType_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 639 of file axis_interface.h.

10.2.1.3 Box

using arcs::common_interface::Box = std::vector<double>

Definition at line 841 of file type_def.h.

10.2.1.4 Cylinder

using arcs::common_interface::Cylinder = std::vector<double>

Definition at line 848 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 832 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 822 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 827 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 891 of file type_def.h.

10.2.1.11 IoControlPtr

using arcs::common_interface::IoControlPtr = std::shared_ptr<IoControl>

Definition at line 5774 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 6779 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 818 of file type_def.h.

10.2.1.15 RegisterControlPtr

using arcs::common_interface::RegisterControlPtr = std::shared_ptr<RegisterControl>

Definition at line 3548 of file register_control.h.

10.2.1.16 ResultWithErrno

using arcs::common_interface::ResultWithErrno = std::tuple<std::vector<double>, int>

Definition at line 812 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 813 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 814 of file type_def.h.

10.2.1.19 ResultWithErrno3

using arcs::common_interface::ResultWithErrno3 = std::tuple<int, int>

Definition at line 815 of file type_def.h.

10.2.1.20 RobotAlgorithmPtr

using arcs::common_interface::RobotAlgorithmPtr = std::shared_ptr<RobotAlgorithm>

Definition at line 1986 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 872 of file type_def.h.

10.2.1.25 RobotStatePtr

using arcs::common_interface::RobotStatePtr = std::shared_ptr<RobotState>

Definition at line 3510 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 853 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 562 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 577 of file type_def.h.

10.2.2.3 EnvelopingShape

enum arcs::common_interface::EnvelopingShape : int

Enumerator

ENUM_EnvelopingShape_DECLARES	

Definition at line 664 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 906 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 918 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 729 of file type_def.h.

10.2.2.7 HandleModeType

enum class arcs::common_interface::HandleModeType : int [strong]

Enumerator

ENUM_HandleModeType_DECLARES	

Definition at line 587 of file type_def.h.

10.2.2.8 HandleStateType

enum class arcs::common_interface::HandleStateType : int [strong]

Enumerator

ENUM_HandleStateType_DECLARES	

Definition at line 582 of file type_def.h.

10.2.2.9 JointServoModeType

enum class arcs::common_interface::JointServoModeType : int [strong]

Joint Servo Mode.

Enumerator

ENUM_JointServoModeType_DECLARES	

Definition at line 620 of file type_def.h.

10.2.2.10 JointStateType

enum class arcs::common_interface::JointStateType : int [strong]

Joint State.

Enumerator

ENUM_JointStateType_DECLARES	

Definition at line 628 of file type_def.h.

10.2.2.11 OperationalModeType

enum class arcs::common_interface::OperationalModeType : int [strong]

Operational Mode.

Enumerator

ENUM_OperationalModeType_DECLARES	

Definition at line 604 of file type_def.h.

10.2.2.12 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.

10.2.2.13 PayloadIdentifyMoveAxis

enum arcs::common_interface::PayloadIdentifyMoveAxis : int

Enumerator

ENUM_PayloadIdentifyMoveAxis_DECLARES	

Definition at line 669 of file type_def.h.

10.2.2.14 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 737 of file type_def.h.

10.2.2.15 RobotControlModeType

enum class arcs::common_interface::RobotControlModeType : int [strong]

Robot Control Mode.

Enumerator

ENUM_RobotControlModeType_DECLARES	

Definition at line 612 of file type_def.h.

10.2.2.16 RobotEmergencyStopType

enum arcs::common_interface::RobotEmergencyStopType : int

Enumerator

ENUM_RobotEmergencyStopType_DECLARES	

Definition at line 684 of file type_def.h.

10.2.2.17 RobotModeType

enum class arcs::common_interface::RobotModeType : int [strong]

Enumerator

ENUM_RobotModeType_DECLARES	

Definition at line 572 of file type_def.h.

10.2.2.18 RuntimeState

enum class arcs::common_interface::RuntimeState : int [strong]

Enumerator

ENUM_RuntimeState_DECLARES	

Definition at line 567 of file type_def.h.

10.2.2.19 SafeguedStopType

enum arcs::common_interface::SafeguedStopType : int

Enumerator

ENUM_SafeguedStopType_DECLARES	

Definition at line 679 of file type_def.h.

10.2.2.20 SafetyInputAction

enum class arcs::common_interface::SafetyInputAction : int [strong]

Enumerator

ENUM_SafetyInputAction_DECLARES	

Definition at line 649 of file type_def.h.

10.2.2.21 SafetyModeType

enum class arcs::common_interface::SafetyModeType : int [strong]

Enumerator

ENUM_SafetyModeType_DECLARES	

Definition at line 596 of file type_def.h.

10.2.2.22 SafetyOutputRunState

enum class arcs::common_interface::SafetyOutputRunState : int [strong]

Enumerator

ENUM_SafetyOutputRunState_DECLARES	

Definition at line 654 of file type_def.h.

10.2.2.23 StandardInputAction

enum class arcs::common_interface::StandardInputAction : int [strong]

The StandardInputAction enum.

Enumerator

ENUM_StandardInputAction_DECLARES	

Definition at line 644 of file type_def.h.

10.2.2.24 StandardOutputRunState

enum class arcs::common_interface::StandardOutputRunState : int [strong]

Standard Output Run State.

Enumerator

ENUM_StandardOutputRunState_DECLARES	

Definition at line 636 of file type_def.h.

Enumerator

ENUM_TaskFrameType_DECLARES	

Definition at line 659 of file type_def.h.

10.2.2.26 TraceLevel

enum arcs::common_interface::TraceLevel

Enumerator

ENUM_TraceLevel_DECLARES	

Definition at line 674 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 760 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 788 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 774 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 806 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 972 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)

Enumerator

ARCS_ERROR_CODES	

Definition at line 56 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 103 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 73 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 63 of file error_stack.h.

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 syncronous 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:

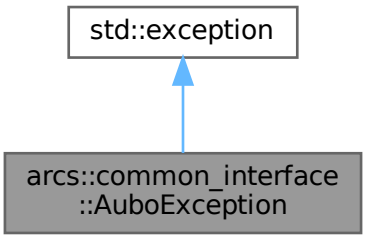
- include/aubo/aubo_api.h

11.2 arcs::common_interface::AuboException Class Reference

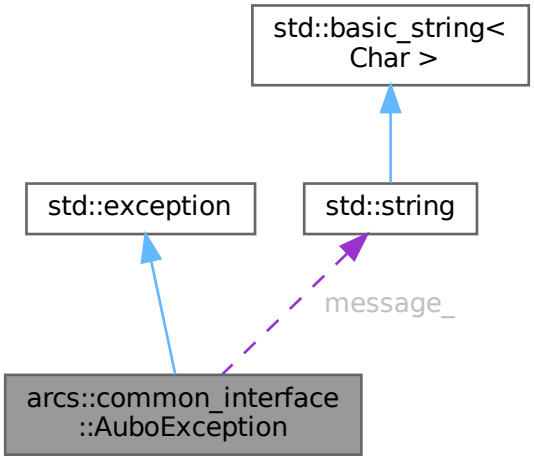
Custom exception class AuboException.

#include <type_def.h>

Inheritance diagram for arcs::common_interface::AuboException:



Collaboration diagram for arcs::common_interface::AuboException:



Public Member Functions

- AuboException (int code, const std::string &prefix, const std::string &message) noexcept
- AuboException (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 938 of file type_def.h.

11.2.2 Constructor & Destructor Documentation

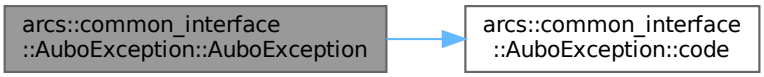
11.2.2.1 AuboException() [1/2]

arcs::common_interface::AuboException::AuboException (
int code,
const std::string & prefix,
const std::string & message) [inline], [noexcept]

Definition at line 941 of file type_def.h.

References code().

Here is the call graph for this function:



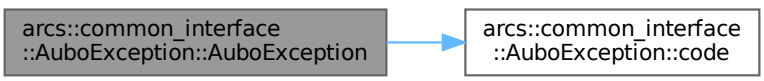
11.2.2.2 AuboException() [2/2]

arcs::common_interface::AuboException::AuboException (
int code,
const std::string & message) [inline], [noexcept]

Definition at line 947 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()

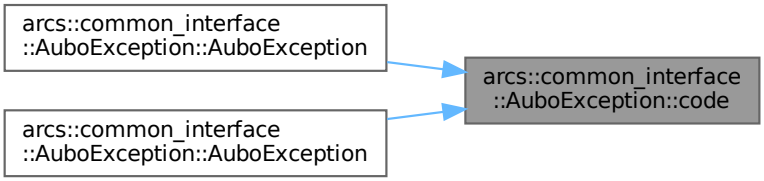
int arcs::common_interface::AuboException::code () const [inline]

Definition at line 964 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 952 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 965 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 968 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 969 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.
- int setExtAxisType (int type)
  Set external axis type.
- 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
- int setExtAxisVelocity (double velocity)

Set external axis speed

- double getExtAxisAcceleration ()

  Get external axis acceleration
- int setExtAxisAcceleration (double acc)

  Set 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
- int setExtAxisMaxPositionLimit (double q)

  Set external axis max position
- double getExtAxisMaxPositionLimit ()

  Get external axis max position
- double getExtMinPosition ()

  Get external axis min position
- int setExtMinPositionLimit (double q)

  Set external axis min position
- double getExtMinPositionLimit ()

  Get external axis min position
- double getExtAxisMaxVelocity ()

  Get external axis max speed
- int setExtAxisMaxVelocityLimit (double v)

  Set external axis max speed
- double getExtAxisMaxVelocityLimit ()

  Get external axis max speed
- double getExtAxisMaxAcceleration ()

  Get external axis max acceleration
- int setExtAxisMaxAccelerationLimit (double acc)

  Set external axis max acceleration
- double getExtAxisMaxAccelerationLimit ()

  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 ()

  getErrorCode(Get raw external axis error code)
- int clearAxisError ()

  Reset axis error
- int setExtAxisZero ()

  Set current external axis position as zero.
- int setReductionRatio (double ratio)

  Set axis reduction ratio
- double getReductionRatio ()

  Get axis reduction ratio

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 637 of file axis_interface.h.

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

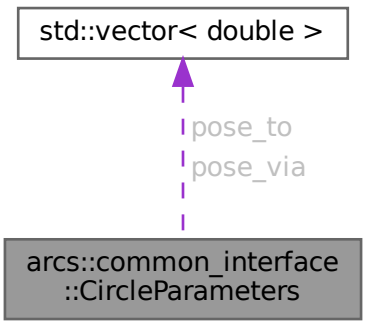
- include/aubo/axis_interface.h

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/s^2.
- 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 746 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 750 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 752 of file type_def.h.

11.4.2.3 direction

double arcs::common_interface::CircleParameters::direction

Definition at line 756 of file type_def.h.

11.4.2.4 duration

double arcs::common_interface::CircleParameters::duration

Running time, unit: s.

Definition at line 753 of file type_def.h.

11.4.2.5 helix

double arcs::common_interface::CircleParameters::helix

Definition at line 754 of file type_def.h.

11.4.2.6 loop_times

int arcs::common_interface::CircleParameters::loop_times

Currently not supported.

Definition at line 757 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 749 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 748 of file type_def.h.

11.4.2.9 spiral

double arcs::common_interface::CircleParameters::spiral

Definition at line 755 of file type_def.h.

11.4.2.10 v

double arcs::common_interface::CircleParameters::v

Speed, unit: m/s.

Definition at line 751 of file type_def.h.

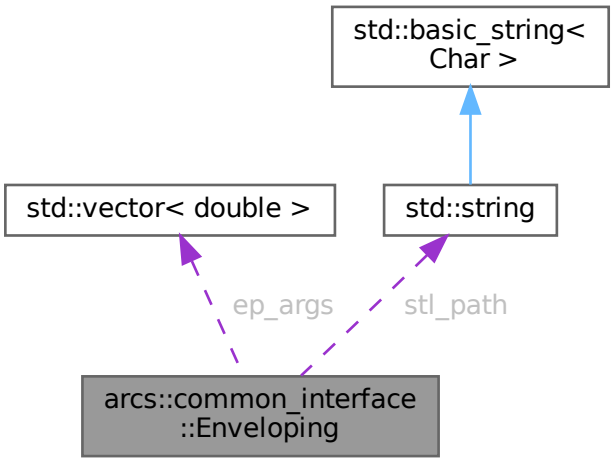
The documentation for this struct was generated from the following file:

- 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 779 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 782 of file type_def.h.

11.5.2.2 shape

EnvelopingShape arcs::common_interface::Enveloping::shape

Definition at line 781 of file type_def.h.

11.5.2.3 stl_path

std::string arcs::common_interface::Enveloping::stl_path

Definition at line 785 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 > &center, 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)

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.

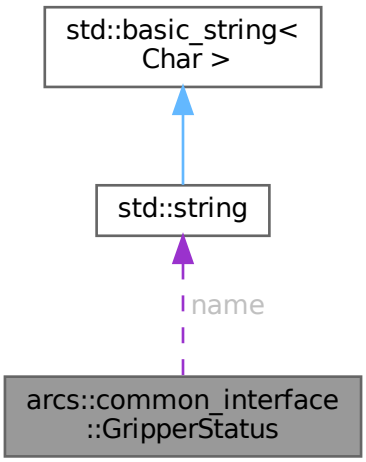
The documentation for this class was generated from the following file:

- 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 874 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 882 of file type_def.h.

11.8.2.2 force

double arcs::common_interface::GripperStatus::force

Definition at line 881 of file type_def.h.

11.8.2.3 is_connected

bool arcs::common_interface::GripperStatus::is_connected

Definition at line 877 of file type_def.h.

11.8.2.4 is_enabled

bool arcs::common_interface::GripperStatus::is_enabled

Definition at line 878 of file type_def.h.

11.8.2.5 motion_state

bool arcs::common_interface::GripperStatus::motion_state

Definition at line 886 of file type_def.h.

11.8.2.6 name

std::string arcs::common_interface::GripperStatus::name

Definition at line 876 of file type_def.h.

11.8.2.7 object_detection

bool arcs::common_interface::GripperStatus::object_detection

Definition at line 885 of file type_def.h.

11.8.2.8 position

double arcs::common_interface::GripperStatus::position

Definition at line 879 of file type_def.h.

11.8.2.9 r_velocity

double arcs::common_interface::GripperStatus::r_velocity

Definition at line 883 of file type_def.h.

11.8.2.10 status_code

int arcs::common_interface::GripperStatus::status_code

Definition at line 889 of file type_def.h.

11.8.2.11 temperature

double arcs::common_interface::GripperStatus::temperature

Definition at line 888 of file type_def.h.

11.8.2.12 torque

double arcs::common_interface::GripperStatus::torque

Definition at line 884 of file type_def.h.

11.8.2.13 velocity

double arcs::common_interface::GripperStatus::velocity

Definition at line 880 of file type_def.h.

11.8.2.14 voltage

double arcs::common_interface::GripperStatus::voltage

Definition at line 887 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.
- int setDigitalOutputRunstate (int io_type, int index, StandardOutputRunState runstate, const std::string &object_name, double threshold)

  Set digital output runstate with optional condition checking based on waypoint, plane or obstacle thresholds.
- 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.
- StandardOutputRunState getDigitalOutputRunstate (int io_type, int index)

  Get the current runstate of a digital output of the specified type.
- int addCrossConnection (const std::string &id, int operator_type, int result_type, int result_pin, const std::vector< int > &actor_types, const std::vector< int > &actor_pins, const std::vector< bool > &actor_inverts)

  Add a DIO Cross Connection logic rule.
- int deleteCrossConnection (const std::string &id)

  Delete a DIO Cross Connection logic rule.
- int clearCrossConnections ()

  Clear all DIO Cross Connection logic rules.
- 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 addGroupOutput (const std::string &name, int io_type, const std::vector< int > &indexes)

  Register a grouped digital output.
- int deleteGroupOutput (const std::string &name)

  Delete a grouped digital output.
- int setGroupOutput (const std::string &name, uint32_t value)

  Set the grouped digital output value.
- 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.
- int addGroupInput (const std::string &name, int io_type, const std::vector< int > &indexes)

  Register a grouped digital input.
- int deleteGroupInput (const std::string &name)

  Delete a grouped digital input.
- uint32_t getGroupInput (const std::string &name)

  Get the grouped digital input value.
- 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.
- uint32_t getGroupOutput (const std::string &name)

  Get the grouped digital output value.
- 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 47 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 5772 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_from_to)

  Pose transformation
- std::vector< double > poseTransInv (const std::vector< double > &pose_from, const std::vector< double > &pose_to_from)

  Pose inverse transformation
- 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 > &rotv)

  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.
- int speedFractionCritical1 (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 &param)

  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 &params)

  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 > &center, 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.
- int setConveyorTrackVelThreshold (int encoder_id, double threshold)

  Set the conveyor velocity threshold used to suppress position jumps caused by abnormal encoder spikes.
- 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 &param, 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 &params)

  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)
- int getWeaveDirection ()
  Get current weave trajectory direction

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 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}

11.11.3.3 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)
```

11.11.3.4 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 6777 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 addFixedModbusGroup (const std::string &signal_name, int count)

  Create a fixed read group for the specified Modbus register signal to optimize the batch reading efficiency of continuous addresses
- 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 modbusSetOutputRunstate1 (const std::string &robot_name, const std::string &signal_name, StandardOutputRunState runstate, const std::string &object_name, double threshold)

  Set Modbus signal output action with optional condition checking based on waypoint, plane or obstacle thresholds.
- 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 3546 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
- ResultWithErrno inverseKinematics2 (const std::vector< double > &qnear, const std::vector< double > &pose)

  Inverse kinematics with the same robot configuration and minimum joint change Calculate the inverse kinematics solution for the target TCP pose according to the robot configuration of qnear, and select the solution with the minimum joint change relative to qnear among solutions with the same configuration.
- 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 &param1, const VibrationRecalibrationParameter &param2, double threshold)
- int addCollisionBox (const std::string &name, const std::string &link_name, const std::vector< std::vector< double > > &sizes, const std::vector< std::vector< double > > &poses)

  Add a box collision object
- int removeCollisionObject (const std::string &name)

  Remove a collision object by name
- 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 without external collision checks.
- int validatePath1 (int type, const std::vector< double > &start, double r1, const std::vector< double > &end, double r2, double d, bool check_external_collision)

  Validate the reachability of the robot's motion path from start point to end point, with optional external collision checks.

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 1984 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/s^2.
- double getDefaultToolSpeed ()
  Get the default tool speed, in m/s.
- double getDefaultJointAcc ()
  Get the default joint acceleration, in rad/s^2.
- 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 > > &param)

  Set standard DH compensation to the robot.
- int setPersistentParameters (const std::string &param)

  Set parameters to be saved to the interface board base.
- int setHardwareCustomParameters (const std::string &param)

  Set custom parameters for the hardware abstraction layer.
- std::string getHardwareCustomParameters (const std::string &param)

  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 &parameters)

  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 &parameters)

  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).
- SafeguedStopType 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 handguide↩ Mode instead: handguideMode({1,1,1,1,1}, {0,0,0,0,0,0})
- 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.

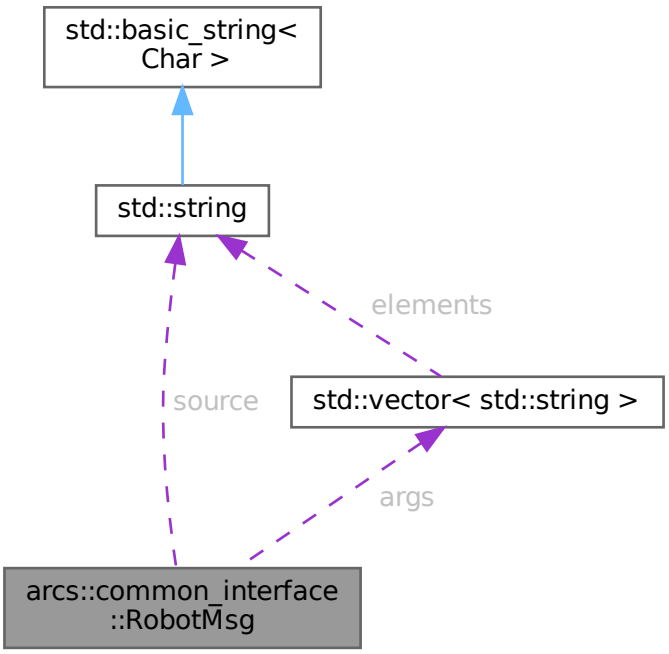
The documentation for this class was generated from the following file:

- 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 855 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 870 of file type_def.h.

11.17.2.2 code

int arcs::common_interface::RobotMsg::code

Error code.

Definition at line 860 of file type_def.h.

11.17.2.3 level

TraceLevel arcs::common_interface::RobotMsg::level

Log level.

Definition at line 859 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 862 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 857 of file type_def.h.

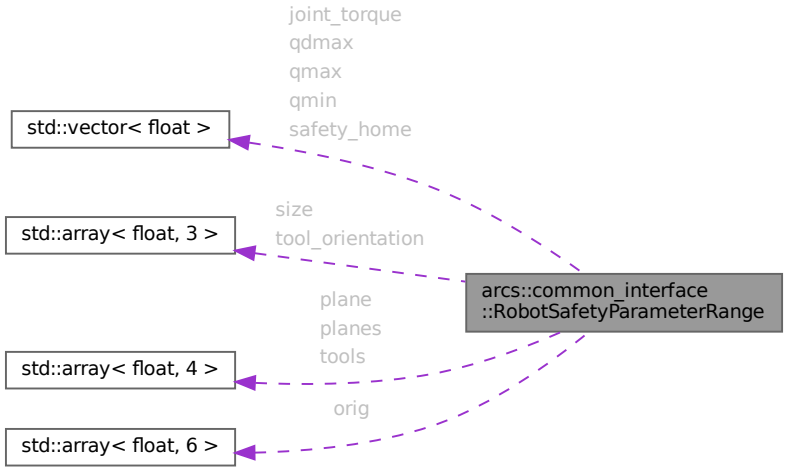
The documentation for this struct was generated from the following file:

- 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, qdmax, 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_safetyguard_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 getHandleUniqueId ()

  Get the globally unique ID of the Handle
- int getHandleFirmwareVersion ()

  Get the Handle firmware version
- int getHandleHardwareVersion ()

  Get the Handle hardware version
- HandleStateType getHandleState ()

  Get the Handle state
- HandleModeType getHandleMode ()

  Get the Handle mode
- std::string getPowerlossBoardUniqueId ()

  Get the globally unique ID of the PowerlossBoard
- int getPowerlossBoardFirmwareVersion ()

  Get the PowerlossBoard firmware version
- int getPowerlossBoardHardwareVersion ()

  Get the PowerlossBoard 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
- double getPedestalBrakeResTemp ()
  Get the Pedestal Brake Res Temp
- 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 3508 of file robot_state.h.

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

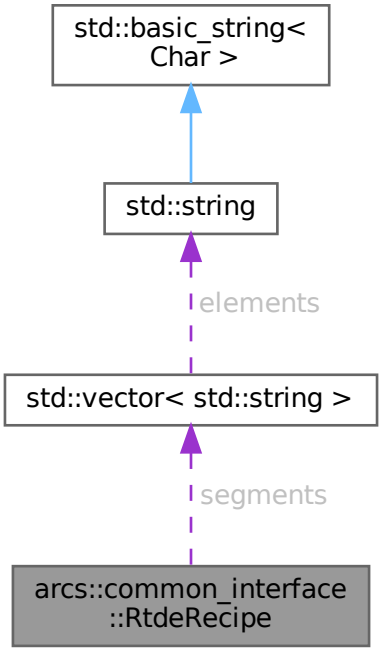
- include/aubo/robot/robot_state.h

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 894 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 897 of file type_def.h.

11.20.2.2 frequency

double arcs::common_interface::RtdeRecipe::frequency

Update frequency.

Definition at line 898 of file type_def.h.

11.20.2.3 segments

std::vector<std::string> arcs::common_interface::RtdeRecipe::segments

segment list

Definition at line 902 of file type_def.h.

11.20.2.4 to_server

bool arcs::common_interface::RtdeRecipe::to_server

Input/Output.

Definition at line 896 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 899 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.

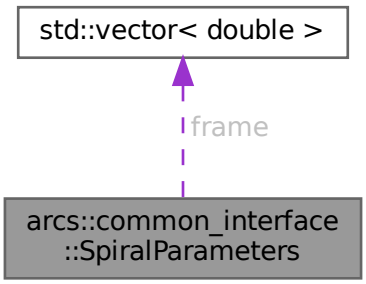
The documentation for this class was generated from the following file:

- 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 765 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 769 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 767 of file type_def.h.

11.24.2.3 helix

double arcs::common_interface::SpiralParameters::helix

Positive upward.

Definition at line 771 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 768 of file type_def.h.

11.24.2.5 spiral

double arcs::common_interface::SpiralParameters::spiral

Positive outward.

Definition at line 770 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:

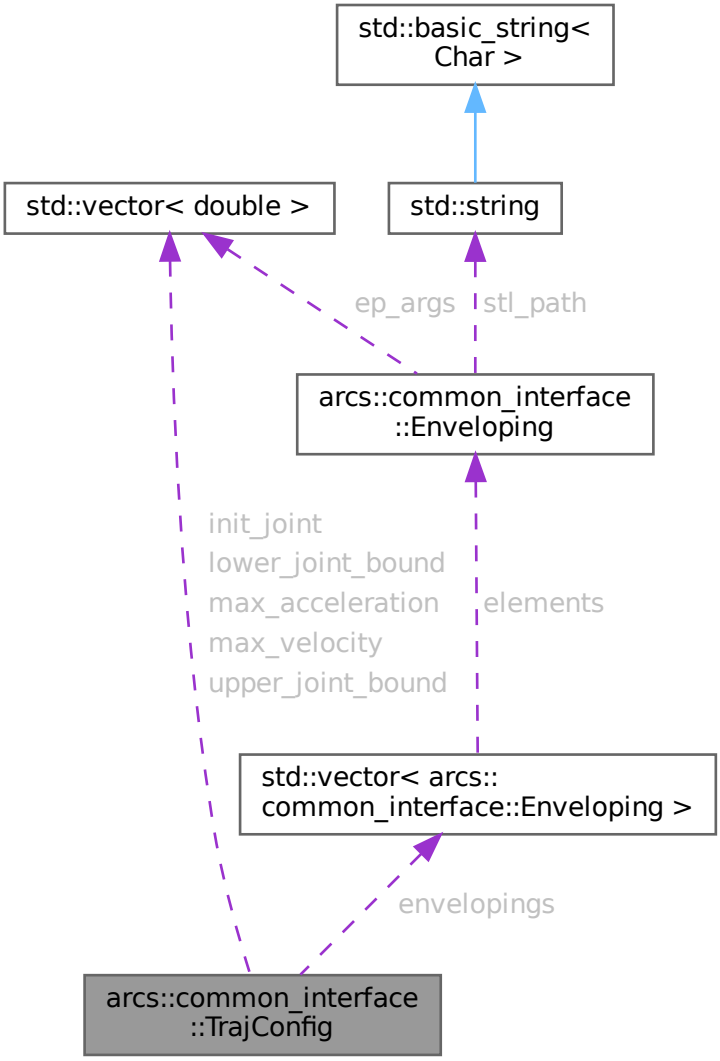
- include/aubo/trace.h

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 795 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 797 of file type_def.h.

11.28.2.2 init_joint

std::vector<double> arcs::common_interface::TrajConfig::init_joint

Definition at line 799 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 801 of file type_def.h.

11.28.2.4 max_acceleration

std::vector<double> arcs::common_interface::TrajConfig::max_acceleration

Definition at line 803 of file type_def.h.

11.28.2.5 max_velocity

std::vector<double> arcs::common_interface::TrajConfig::max_velocity

Definition at line 802 of file type_def.h.

11.28.2.6 move_axis

PayloadIdentifyMoveAxis arcs::common_interface::TrajConfig::move_axis

Definition at line 798 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 800 of file type_def.h.

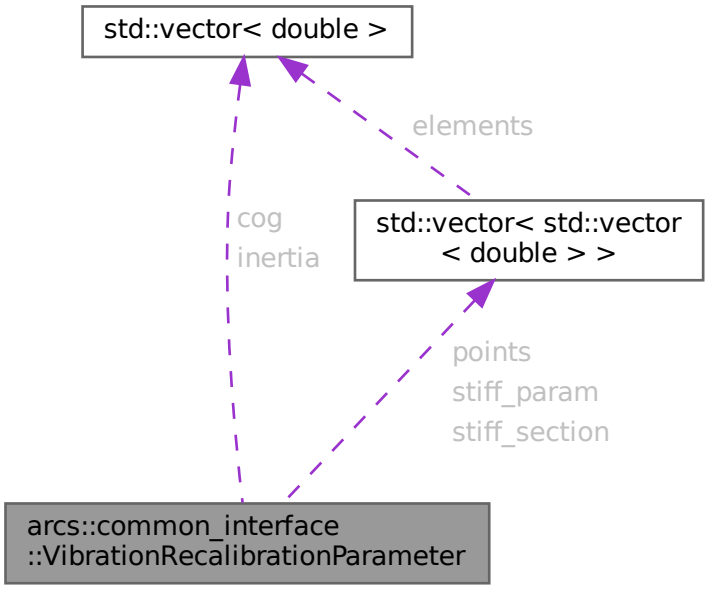
The documentation for this struct was generated from the following file:

- 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.

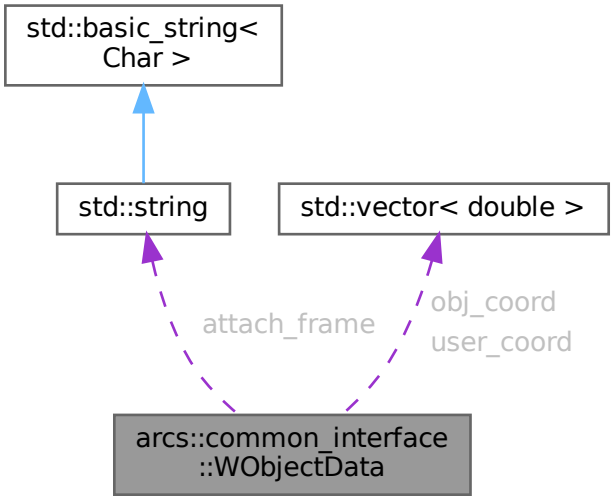
The documentation for this struct was generated from the following file:

- 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

Chapter 12

File Documentation

12.1 doc/sdk_overview_en.md File Reference

12.2 include/aubo/aubo_api.h File Reference

API for controlling the robot and external axis.

```
#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:

Classes

- class arcs::common_interface::AuboApi

Namespaces

- namespace arcs
- namespace arcs::common_interface

Typedefs

- 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     *
```

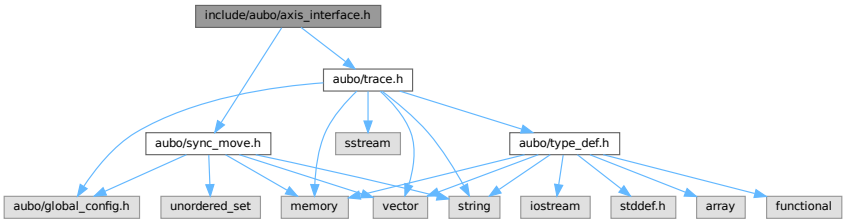
```
00244     * @param name Robot name
00245     * @return Shared pointer to a RobotInterface object
00246     *
00247     * @par Python function prototype
00248     * getRobotInterface(self: pyaubo_sdk.AuboApi, arg0: str) ->
00249     * pyaubo_sdk.RobotInterface
00250     *
00251     * @par C++ example
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 syncronous move interface
00359      *
00360      * \~chinese @return SyncMovePtr
00361      * \~english @return Shared pointer to SyncMove object
00362      */
00363     SyncMovePtr getSyncMove(const std::string &name);
00364
00365     /**
00366      * @ingroup AuboApi
00367      * @ref Trace
00368      * \~chinese
00369      * \~english Get alert interface
00370      *
00371      * \~chinese @return TracePtr
00372      * \~english @return Shared pointer of trace object
00373      */
00374     TracePtr getTrace(const std::string &name);
00375
00376     /**
00377      * @ingroup AuboApi
00378      * @ref GripperInterface
00379      * \~chinese
00380      * \~english Get gripper interface
00381      *
00382      * \~chinese @return GripperInterfacePtr
00383      * \~english @return Shared pointer of gripper object
00384      */
00385     GripperInterfacePtr getGripperInterface();
00386
00387 protected:
00388     void *d_{ nullptr };
00389 };
00390 using AuboApiPtr = std::shared_ptr<AuboApi>;
00391
00392 } // namespace common_interface
00393 } // namespace arcs
00394
00395 #endif // AUBO_SDK_AUBO_API_H
```

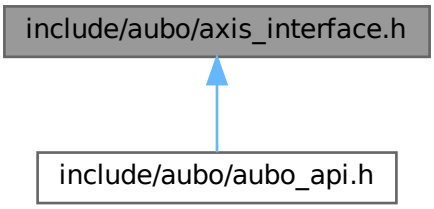
12.4 include/aubo/axis_interface.h File Reference

#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     * @param type 0     1
00131     *
00132     * \~english Set external axis type
00133     * @param type: 0 for rotation, 1 for linear
00134     * @return
00135     */
00136    int setExtAxisType(int type);
00137
00138    /**
00139     * @ingroup AxisInterface
00140     * \chinese
00141     *
00142     *
00143     * @return
00144     * \endchinese
00145     *
00146     * \english
00147     * Get external axis status
00148     *
00149     * @return Current exteral axis status
00150     * \endenglish
00151     */
00152    AxisModeType getAxisModeType();
00153
00154    /**
00155     * @ingroup AxisInterface
00156     * \chinese
00157     *
00158     *
00159     * @return
00160     * \endchinese
00161     *
00162     * \english
00163     * Get external axis mounting pose
00164     *
00165     * @return External axis pose
00166     * \endenglish
00167     */
00168    std::vector<double> getExtAxisMountingPose();
00169
00170    /**
00171     * @ingroup AxisInterface
00172     * \chinese
00173     *
00174     *
00175     * @return
00176     * \endchinese
00177     *
00178     * \english
00179     * Get pose wrt mounting coordinate system, axis can be positioner or linear
00180     * rail
00181     *
```

```
00182     * @return Pose wrt mounting coordinate system
00183     * \endenglish
00184     */
00185    std::vector<double> getExtAxisPose();
00186
00187    /**
00188     * @ingroup AxisInterface
00189     * \chinese
00190     *
00191     *
00192     * @return
00193     * \endchinese
00194     *
00195     * \english
00196     * Get external axis position
00197     *
00198     * @return External axis position
00199     * \endenglish
00200     */
00201    double getExtAxisPosition();
00202
00203    /**
00204     * @ingroup AxisInterface
00205     * \chinese
00206     *
00207     *
00208     * @return
00209     * \endchinese
00210     *
00211     * \english
00212     * Get external axis speed
00213     *
00214     * @return External axis speed
00215     * \endenglish
00216     */
00217    double getExtAxisVelocity();
00218
00219    /**
00220     * @ingroup AxisInterface
00221     * \chinese
00222     *
00223     *
00224     * @return
00225     * \endchinese
00226     *
00227     * \english
00228     * Set external axis speed
00229     *
00230     * @return
00231     * \endenglish
00232     */
00233    int setExtAxisVelocity(double velocity);
00234
00235    /**
00236     * @ingroup AxisInterface
00237     * \chinese
00238     *
00239     *
00240     * @return
00241     * \endchinese
00242     *
00243     * \english
00244     * Get external axis acceleration
00245     *
00246     * @return External axis acceleration
00247     * \endenglish
00248     */
00249    double getExtAxisAcceleration();
00250
00251    /**
00252     * @ingroup AxisInterface
00253     * \chinese
00254     *
00255     *
00256     * @param acc
00257     * @return
00258     * \endchinese
00259     *
00260     * \english
00261     * Set external axis acceleration
00262     *
00263     * @param acc
00264     * @return
00265     * \endenglish
00266     */
00267    int setExtAxisAcceleration(double acc);
00268
```

```
00269     /**
00270      * @ingroup AxisInterface
00271      * \chinese
00272      *
00273      *
00274      * @return
00275      * \endchinese
00276      *
00277      * \english
00278      * Get external axis current
00279      *
00280      * @return External axis current
00281      * \endenglish
00282      */
00283     double getExtAxisCurrent();
00284
00285     /**
00286      * @ingroup AxisInterface
00287      * \chinese
00288      *
00289      *
00290      * @return
00291      * \endchinese
00292      *
00293      * \english
00294      * Get external axis temperature
00295      *
00296      * @return External axis temperature
00297      * \endenglish
00298      */
00299     double getExtAxisTemperature();
00300
00301     /**
00302      * @ingroup AxisInterface
00303      * \chinese
00304      *
00305      *
00306      * @return
00307      * \endchinese
00308      *
00309      * \english
00310      * Get external axis voltage
00311      *
00312      * @return External axis voltage
00313      * \endenglish
00314      */
00315     double getExtAxisBusVoltage();
00316
00317     /**
00318      * @ingroup AxisInterface
00319      * \chinese
00320      *
00321      *
00322      * @return
00323      * \endchinese
00324      *
00325      * \english
00326      * Get external axis current
00327      *
00328      * @return external axis current
00329      * \endenglish
00330      */
00331     double getExtAxisBusCurrent();
00332
00333     /**
00334      * @ingroup AxisInterface
00335      * \chinese
00336      *       (   )
00337      *
00338      * @return
00339      * \endchinese
00340      *
00341      * \english
00342      * Get external axis max position
00343      *
00344      * @return External axis max position
00345      * \endenglish
00346      */
00347     double getExtAxisMaxPosition();
00348
00349     /**
00350      * @ingroup AxisInterface
00351      * \chinese
00352      *       ( /   )
00353      *
00354      * @param q
00355      * @return
```

```
00356     * \endchinese
00357     *
00358     * \english
00359     * Set external axis max position
00360     *
00361     * @param q
00362     * @return
00363     * \endenglish
00364     */
00365    int setExtAxisMaxPositionLimit(double q);
00366
00367    /**
00368     * @ingroup AxisInterface
00369     * \chinese
00370     *       ( /   )
00371     *
00372     * @return
00373     * \endchinese
00374     *
00375     * \english
00376     * Get external axis max position
00377     *
00378     * @return External axis max position
00379     * \endenglish
00380     */
00381    double getExtAxisMaxPositionLimit();
00382
00383    /**
00384     * @ingroup AxisInterface
00385     * \chinese
00386     *       (   )
00387     *
00388     * @return
00389     * \endchinese
00390     *
00391     * \english
00392     * Get external axis min position
00393     *
00394     * @return External axis min position
00395     * \endenglish
00396     */
00397    double getExtMinPosition();
00398
00399    /**
00400     * @ingroup AxisInterface
00401     * \chinese
00402     *       ( /   )
00403     *
00404     * @param q
00405     * @return
00406     * \endchinese
00407     *
00408     * \english
00409     * Set external axis min position
00410     *
00411     * @param q
00412     * @return
00413     * \endenglish
00414     */
00415    int setExtMinPositionLimit(double q);
00416
00417    /**
00418     * @ingroup AxisInterface
00419     * \chinese
00420     *       ( /   )
00421     *
00422     * @return
00423     * \endchinese
00424     *
00425     * \english
00426     * Get external axis min position
00427     *
00428     * @return External axis min position
00429     * \endenglish
00430     */
00431    double getExtMinPositionLimit();
00432
00433    /**
00434     * @ingroup AxisInterface
00435     * \chinese
00436     *       (   )
00437     *
00438     * @return
00439     * \endchinese
00440     *
00441     * \english
00442     * Get external axis max speed
```

```
00443     *
00444     * @return External axis max speed
00445     * \endenglish
00446     */
00447    double getExtAxisMaxVelocity();
00448
00449    /**
00450     * @ingroup AxisInterface
00451     * \chinese
00452     *       ( /   )
00453     *
00454     * @param v
00455     * @return
00456     * \endchinese
00457     *
00458     * \english
00459     * Set external axis max speed
00460     *
00461     * @param v
00462     * @return
00463     * \endenglish
00464     */
00465    int setExtAxisMaxVelocityLimit(double v);
00466
00467    /**
00468     * @ingroup AxisInterface
00469     * \chinese
00470     *       ( /   )
00471     *
00472     * @return
00473     * \endchinese
00474     *
00475     * \english
00476     * Get external axis max speed
00477     *
00478     * @return External axis max speed
00479     * \endenglish
00480     */
00481    double getExtAxisMaxVelocityLimit();
00482
00483    /**
00484     * @ingroup AxisInterface
00485     * \chinese
00486     *       (   )
00487     *
00488     * @return
00489     * \endchinese
00490     *
00491     * \english
00492     * Get external axis max acceleration
00493     *
00494     * @return External axis max acceleration
00495     * \endenglish
00496     */
00497    double getExtAxisMaxAcceleration();
00498
00499    /**
00500     * @ingroup AxisInterface
00501     * \chinese
00502     *       ( /   )
00503     *
00504     * @param acc
00505     * @return
00506     * \endchinese
00507     *
00508     * \english
00509     * Set external axis max acceleration
00510     *
00511     * @param acc
00512     * @return
00513     * \endenglish
00514     */
00515    int setExtAxisMaxAccelerationLimit(double acc);
00516
00517    /**
00518     * @ingroup AxisInterface
00519     * \chinese
00520     *       ( /   )
00521     *
00522     * @return
00523     * \endchinese
00524     *
00525     * \english
00526     * Get external axis max acceleration
00527     *
00528     * @return External axis max acceleration
00529     * \endenglish
```

```
00530     */
00531     double getExtAxisMaxAccelerationLimit();
00532
00533     /**
00534      * @ingroup AxisInterface
00535      * \chinese
00536      *        (      )
00537      *
00538      * @param target_name
00539      * @param phase
00540      * @param err
00541      * @return
00542      * \endchinese
00543      *
00544      * \english
00545      * Follow motion of another external axis  not to be used during motion
00546      *
00547      * @param target_name name of target axis
00548      * @param phase phase difference
00549      * @param err max error when following motion
00550      * @return
00551      * \endenglish
00552      */
00553     int followAnotherAxis(const std::string &target_name, double phase,
00554                           double err);
00555
00556     /**
00557      * @ingroup AxisInterface
00558      * \~chinese @brief stopFollowAnotherAxis(       )
00559      * \~english @brief stopFollowAnotherAxis(not to be used during motion)
00560      * @return
00561      */
00562     int stopFollowAnotherAxis();
00563
00564     /**
00565      * @ingroup AxisInterface
00566      * \~chinese @brief getErrorCode(      )
00567      * \~english @brief getErrorCode(Get raw external axis error code)
00568      * @return
00569      */
00570     int getErrorCode();
00571
00572     /**
00573      * @ingroup AxisInterface
00574      * \chinese
00575      *
00576      *
00577      * @return
00578      * \endchinese
00579      *
00580      * \english
00581      * Reset axis error
00582      *
00583      * @return
00584      * \endenglish
00585      */
00586     int clearAxisError();
00587
00588     /**
00589      * @ingroup AxisInterface
00590      * \chinese
00591      *
00592      *
00593      * @return
00594      * \endchinese
00595      *
00596      * \english
00597      * Set current external axis position as zero.
00598      *
00599      * @return
00600      * \endenglish
00601      */
00602     int setExtAxisZero();
00603
00604     /**
00605      * @ingroup AxisInterface
00606      * \chinese
00607      *
00608      *
00609      * @return
00610      * \endchinese
00611      *
00612      * \english
00613      * Set axis reduction ratio
00614      *
00615      * @return
00616      * \endenglish
```

```
00617     */
00618     int setReductionRatio(double ratio);
00619
00620     /**
00621      * @ingroup AxisInterface
00622      * \chinese
00623      *
00624      *
00625      * @return
00626      * \endchinese
00627      *
00628      * \english
00629      * Get axis reduction ratio
00630      *
00631      * @return
00632      * \endenglish
00633      */
00634     double getReductionRatio();
00635
00636 protected:
00637     void *d_;
00638 };
00639 using AxisInterfacePtr = std::shared_ptr<AxisInterface>;
00640
00641 } // namespace common_interface
00642 } // namespace arcs
00643
00644 #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 JOINT_EX_ERRORS
- #define EXT_AXIS_ERRORS
- #define TOOL_ERRORS
- #define EX_TOOL_ERRORS
- #define PEDSTRAL_ERRORS
- #define EX_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 58 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 58 of file error_stack.h.

12.6.2.3 _D [3/6]

```
#define _D(
                n,
                v,
                s,
                r)
```

Value:

```
n##_INDEX,
```

Definition at line 58 of file error_stack.h.

12.6.2.4 _D [4/6]

```
#define _D(
                n,
                v,
                s,
                r)
```

Value:

```
s,
```

Definition at line 58 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 58 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 58 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                \
JOINT_EX_ERRORS             \
EXT_AXIS_ERRORS             \
SAFETY_INTERFACE_BOARD_ERRORS \
RTM_ERRORS                  \
TOOL_ERRORS                 \
EX_TOOL_ERRORS              \
PEDSTRAL_ERRORS             \
EX_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 EX_PEDSTRAL_ERRORS

#define EX_PEDSTRAL_ERRORS

Value:

```
_D(EX_BASE_EC_LOW_VOLTAGE, 50101, "pedstral error: low voltage", "check power supply voltage and battery status") \
_D(EX_BASE_EC_OVER_TEMPERATURE_RES, 50102, "pedstral error: resistor over temperature", "check braking resistor
    temperature and cooling condition") \
_D(EX_BASE_EC_OVER_TARGET_BRAKE_OPEN_VOLT, 50103, "pedstral error: input voltage close to or exceeds
    regenerative brake activation voltage", "check input power voltage and regenerative braking configuration") \
_D(EX_BASE_EC_RES_BREAKAGE, 50104, "pedstral error: brake resistor breakage", "check whether the brake resistor is
    disconnected or damaged") \
_D(EX_BASE_EC_IMU_CALIBRATE, 50105, "pedstral error: IMU calibration required", "perform IMU calibration according
    to maintenance procedure") \
_D(EX_BASE_EC_TEMP_SENSOR_SHORT, 50106, "pedstral error: temperature sensor short circuit", "check temperature
    sensor wiring and solder joints for short circuit") \
_D(EX_BASE_EC_TEMP_SENSOR_BREAK, 50107, "pedstral error: temperature sensor open circuit", "check temperature
    sensor connection and cable continuity") \
_D(EX_BASE_EC_96V_INSTANT_OVER_VOLTAGE, 50108, "pedstral error: 96V instantaneous over voltage after
    regenerative braking", "check regenerative braking behavior and power bus voltage") \
_D(EX_BASE_EC_UNKNOWN, 50200, "pedstral error: unknown error", "check controller logs and hardware status")
```

12.6.2.16 EX_TOOL_ERRORS

#define EX_TOOL_ERRORS

Value:

```
_D(EX_TOOL_EC_LOW_VOLTAGE, 40101, "tool error: low voltage", "check tool power supply voltage") \
_D(EX_TOOL_EC_FORCESENSOR_COMM, 40102, "tool error: external force sensor communication error", "check external
    force sensor communication connection") \
_D(EX_TOOL_485_SENDFULL_COMM, 40103, "tool error: 485 transparent transmission buffer full", "check 485
    communication load and transmission frequency") \
_D(EX_TOOL_FORCESENSOR_FILTER_ZERO, 40104, "tool error: force sensor filter parameter is zero", "check force sensor
    filter parameter configuration") \
_D(EX_TOOL_EC_UNKNOWN, 40200, "tool error: unknown error", "check controller logs and hardware status")
```

12.6.2.17 EXT_AXIS_ERRORS

#define EXT_AXIS_ERRORS

12.6.2.18 HAL_ERRORS

#define HAL_ERRORS

Value:

```
JOINT_ERRORS                \
JOINT_EX_ERRORS               \
EXT_AXIS_ERRORS               \
SAFETY_INTERFACE_BOARD_ERRORS \
TOOL_ERRORS                 \
EX_TOOL_ERRORS                \
PEDSTRAL_ERRORS               \
EX_PEDSTRAL_ERRORS              \
HARDWARE_INTERFACE_ERRORS
```

12.6.2.19 HARDWARE_INTERFACE_ERRORS

#define HARDWARE_INTERFACE_ERRORS

12.6.2.20 JOINT_ERRORS

#define JOINT_ERRORS

12.6.2.21 JOINT_EX_ERRORS

#define JOINT_EX_ERRORS

12.6.2.22 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(PEDESTAL_ERR_BOOTLOADER,  50003, "pedestal error: The pedestal is in bootloader mode. Retry firmware update. ",
      "suggest...")
```

12.6.2.23 RTM_ERRORS

#define RTM_ERRORS

12.6.2.24 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 postion 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...") \
_D(HANDLE_OFFLINE, 20026, "handle error: handle may be offline", "(a) Check handle's hardware. (b) Check handle's id.") \
_D(HANDLE_ERR_BOOTLOADER, 20027, "handle error: The handle is in bootloader mode. Retry firmware update. ",
    "suggest...") \
_D(IFB_POWERLOSS_OFFLINE, 20028, "interface powerloss board error: interface powerloss board may be offline", "(a)
    Check interface powerloss board's hardware. (b) Check interface powerloss board's id.") \
_D(IFB_POWERLOSS_ERR_BOOTLOADER, 20029, "interface powerloss board error: The interface powerloss board is in
    bootloader mode. Retry firmware update. ", "suggest...") \
_D(HANDLE_COMM_ERROR, 20030, "handle error: handle comm error", "(a) Check handle's hardware. (b) Check handle's
    id.")
```

12.6.2.25 SYSTEM_ERRORS

#define SYSTEM_ERRORS

12.6.2.26 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_UNDIFINED, 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

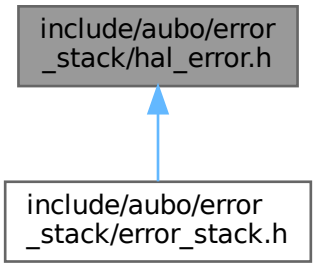
Go to the documentation of this file.

```
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     JOINT_EX_ERRORS               \
00045     EXT_AXIS_ERRORS               \
00046     SAFETY_INTERFACE_BOARD_ERRORS \
00047     RTM_ERRORS                    \
00048     TOOL_ERRORS                   \
00049     EX_TOOL_ERRORS                \
00050     PEDSTRAL_ERRORS               \
00051     EX_PEDSTRAL_ERRORS            \
00052     HARDWARE_INTERFACE_ERRORS     \
00053     _D(ARCS_MAX_ERROR_CODE, -1, "Max error code", "suggest...")
00054
00055 //
00056 enum ErrorCodes
00057 {
00058 #define _D(n, v, s, r) n = (int)v,
00059     ARCS_ERROR_CODES
00060 #undef _D
00061 };
00062
00063 inline int str2ErrorCode(const char *err_code_name)
00064 {
00065 #define _D(n, v, s, r)                \
00066     if (strcmp(#n, err_code_name) == 0) \
00067         return v;
00068     ARCS_ERROR_CODES
00069 #undef _D
00070     return ARCS_MAX_ERROR_CODE;
00071 }
00072
00073 inline const char *errorCode2Str(int err_code)
00074 {
00075     static const char *errcode_str[] = {
00076 #define _D(n, v, s, r) s,
00077         ARCS_ERROR_CODES
00078 #undef _D
00079     };
00080
00081     enum arcs_index
00082     {
```

```
00083 #define _D(n, v, s, r) n##_INDEX,
00084         ARCS_ERROR_CODES
00085 #undef _D
00086     };
00087
00088     int index = -1;
00089
00090 #define _D(n, v, s, r) \
00091     if (err_code == v) \
00092         index = n##_INDEX;
00093     ARCS_ERROR_CODES
00094 #undef _D
00095
00096     if (index == -1) {
00097         index = ARCS_MAX_ERROR_CODE_INDEX;
00098     }
00099
00100     return errcode_str[(unsigned)index];
00101 }
00102
00103 inline std::ostream &dump(std::ostream &os)
00104 {
00105 #define _D(n, v, s, r)                                          \
00106     os « std::setw(20) « #n « "\t" « v « "\t" « s « "\t" « r \
00107        « std::endl;
00108
00109     ARCS_ERROR_CODES
00110 #undef _D
00111
00112     return os;
00113 }
00114
00115 } // namespace error_stack
00116 } // namespace arcs
00117
00118 #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 JOINT_EX_ERRORS
- #define EXT_AXIS_ERRORS
- #define TOOL_ERRORS

- #define EX_TOOL_ERRORS
- #define PEDSTRAL_ERRORS
- #define EX_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 EX_PEDSTRAL_ERRORS

#define EX_PEDSTRAL_ERRORS

Value:

```
_D(EX_BASE_EC_LOW_VOLTAGE, 50101, "pedstral error: low voltage", "check power supply voltage and battery status") \
_D(EX_BASE_EC_OVER_TEMPERATURE_RES, 50102, "pedstral error: resistor over temperature", "check braking resistor
    temperature and cooling condition") \
_D(EX_BASE_EC_OVER_TARGET_BRAKE_OPEN_VOLT, 50103, "pedstral error: input voltage close to or exceeds
    regenerative brake activation voltage", "check input power voltage and regenerative braking configuration") \
_D(EX_BASE_EC_RES_BREAKAGE, 50104, "pedstral error: brake resistor breakage", "check whether the brake resistor is
    disconnected or damaged") \
_D(EX_BASE_EC_IMU_CALIBRATE, 50105, "pedstral error: IMU calibration required", "perform IMU calibration according
    to maintenance procedure") \
_D(EX_BASE_EC_TEMP_SENSOR_SHORT, 50106, "pedstral error: temperature sensor short circuit", "check temperature
    sensor wiring and solder joints for short circuit") \
_D(EX_BASE_EC_TEMP_SENSOR_BREAK, 50107, "pedstral error: temperature sensor open circuit", "check temperature
    sensor connection and cable continuity") \
_D(EX_BASE_EC_96V_INSTANT_OVER_VOLTAGE, 50108, "pedstral error: 96V instantaneous over voltage after
    regenerative braking", "check regenerative braking behavior and power bus voltage") \
_D(EX_BASE_EC_UNKNOWN, 50200, "pedstral error: unknown error", "check controller logs and hardware status")
```

Definition at line 258 of file error_stack.h.

12.8.2.2 EX_TOOL_ERRORS

#define EX_TOOL_ERRORS

Value:

```
_D(EX_TOOL_EC_LOW_VOLTAGE, 40101, "tool error: low voltage", "check tool power supply voltage") \
_D(EX_TOOL_EC_FORCESENSOR_COMM, 40102, "tool error: external force sensor communication error", "check external
    force sensor communication connection") \
_D(EX_TOOL_485_SENDFULL_COMM, 40103, "tool error: 485 transparent transmission buffer full", "check 485
    communication load and transmission frequency") \
_D(EX_TOOL_FORCESENSOR_FILTER_ZERO, 40104, "tool error: force sensor filter parameter is zero", "check force sensor
    filter parameter configuration") \
_D(EX_TOOL_EC_UNKNOWN, 40200, "tool error: unknown error", "check controller logs and hardware status")
```

Definition at line 246 of file error_stack.h.

12.8.2.3 EXT_AXIS_ERRORS

#define EXT_AXIS_ERRORS

Definition at line 157 of file error_stack.h.

12.8.2.4 HAL_ERRORS

#define HAL_ERRORS

Value:

```
JOINT_ERRORS                \
JOINT_EX_ERRORS               \
EXT_AXIS_ERRORS               \
SAFETY_INTERFACE_BOARD_ERRORS \
TOOL_ERRORS                 \
EX_TOOL_ERRORS                \
PEDSTRAL_ERRORS               \
EX_PEDSTRAL_ERRORS              \
HARDWARE_INTERFACE_ERRORS
```

Definition at line 339 of file error_stack.h.

12.8.2.5 HARDWARE_INTERFACE_ERRORS

#define HARDWARE_INTERFACE_ERRORS

Definition at line 301 of file error_stack.h.

12.8.2.6 JOINT_ERRORS

#define JOINT_ERRORS

Definition at line 20 of file error_stack.h.

12.8.2.7 JOINT_EX_ERRORS

#define JOINT_EX_ERRORS

Definition at line 67 of file error_stack.h.

12.8.2.8 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(PEDESTAL_ERR_BOOTLOADER,  50003, "pedestal error: The pedestal is in bootloader mode. Retry firmware update. ",
      "suggest...")
```

Definition at line 253 of file error_stack.h.

12.8.2.9 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 postion 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...") \
_D(HANDLE_OFFLINE, 20026, "handle error: handle may be offline", "(a) Check handle's hardware. (b) Check handle's id.") \
_D(HANDLE_ERR_BOOTLOADER, 20027, "handle error: The handle is in bootloader mode. Retry firmware update. ",
    "suggest...") \
_D(IFB_POWERLOSS_OFFLINE, 20028, "interface powerloss board error: interface powerloss board may be offline", "(a)
    Check interface powerloss board's hardware. (b) Check interface powerloss board's id.") \
_D(IFB_POWERLOSS_ERR_BOOTLOADER, 20029, "interface powerloss board error: The interface powerloss board is in
    bootloader mode. Retry firmware update. ", "suggest...") \
_D(HANDLE_COMM_ERROR, 20030, "handle error: handle comm error", "(a) Check handle's hardware. (b) Check handle's
    id.")
```

Definition at line 269 of file error_stack.h.

12.8.2.10 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_UNDIFINED, 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...")
```

Definition at line 225 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_CURRENET,  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 distributer, 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 distributer, 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 // Joint extended error codes (ex_err_code), starting from 10401
00066 #define JOINT_EX_ERRORS \
00067     _D(EX_JOINT_EC_SHORT_CIRCUIT, 10401, "joint" _PH1_ " error: short circuit protection", "suggest...") \
00068     _D(EX_JOINT_EC_SHORT_CURRENT, 10402, "joint" _PH1_ " error: over current", "suggest...") \
00069     _D(EX_JOINT_EC_PHASEA_CURRENT, 10403, "joint" _PH1_ " error: phase A over current", "suggest...") \
00070     _D(EX_JOINT_EC_PHASEB_CURRENT, 10404, "joint" _PH1_ " error: phase B over current", "suggest...") \
00071     _D(EX_JOINT_EC_PHASEC_CURRENT, 10405, "joint" _PH1_ " error: phase C over current", "suggest...") \
00072     _D(EX_JOINT_EC_PHASE_CURRENT, 10406, "joint" _PH1_ " error: phase over current", "suggest...") \
00073     _D(EX_JOINT_EC_MOTOR_PHASE_LOSE, 10407, "joint" _PH1_ " error: motor phase loss", "suggest...") \
00074     _D(EX_JOINT_EC_BUS_OVER_VOLTAGE, 10408, "joint" _PH1_ " error: bus over voltage", "suggest...") \
00075     _D(EX_JOINT_EC_BUS_LOW_VOLTAGE, 10409, "joint" _PH1_ " error: bus under voltage", "suggest...") \
00076     _D(EX_JOINT_EC_OVERLOAD, 10410, "joint" _PH1_ " error: overload", "suggest...") \
00077     _D(EX_JOINT_EC_IPM_OVER_TEMP, 10411, "joint" _PH1_ " error: IPM over temperature", "suggest...") \
00078     _D(EX_JOINT_EC_REDUCER_OVER_TEMP, 10412, "joint" _PH1_ " error: reducer over temperature", "suggest...") \
00079     _D(EX_JOINT_EC_ADC_ZERO_OFFSET, 10413, "joint" _PH1_ " error: ADC zero offset", "suggest...") \
00080     _D(EX_JOINT_EC_REDUCER_NTC, 10414, "joint" _PH1_ " error: reducer NTC fault", "suggest...") \
00081     _D(EX_JOINT_EC_IPM_NTC, 10415, "joint" _PH1_ " error: IPM NTC fault", "suggest...") \
00082     _D(EX_JOINT_EC_TORQUE_SENSOR, 10416, "joint" _PH1_ " error: torque sensor fault", "suggest...") \
00083     _D(EX_JOINT_EC_TORQUE_SENSOR_COMM, 10417, "joint" _PH1_ " error: torque sensor communication fault", "suggest...") \
00084     _D(EX_JOINT_EC_MOTOR_ABS_ENC_COMM, 10418, "joint" _PH1_ " error: motor-side absolute encoder communication fault", "suggest...") \
00085     _D(EX_JOINT_EC_REDUCER_ABS_ENC_COMM, 10419, "joint" _PH1_ " error: reducer-side absolute encoder communication fault", "suggest...") \
00086     _D(EX_JOINT_EC_REDUCER_ABS_ENC_DATA, 10420, "joint" _PH1_ " error: reducer-side absolute encoder data channel disabled warning", "suggest...") \
00087     _D(EX_JOINT_EC_REDUCER_ABS_ENC_CMD, 10421, "joint" _PH1_ " error: reducer-side absolute encoder command invalid warning", "suggest...") \
00088     _D(EX_JOINT_EC_REDUCER_ABS_ENC_ERR, 10422, "joint" _PH1_ " error: reducer-side absolute encoder fault", "suggest...") \
00089     _D(EX_JOINT_EC_REDUCER_ABS_ENC_WARNING, 10423, "joint" _PH1_ " error: reducer-side absolute encoder warning", "suggest...") \
00090     _D(EX_JOINT_EC_BRAKE, 10424, "joint" _PH1_ " error: brake fault", "suggest...") \
00091     _D(EX_JOINT_EC_COMM_HWL, 10425, "joint" _PH1_ " error: communication hardware layer error", "suggest...") \
00092     _D(EX_JOINT_EC_FIRMWARE_UPDATE, 10426, "joint" _PH1_ " error: firmware update failed", "suggest...") \
00093     _D(EX_JOINT_EC_FLASH_OP, 10427, "joint" _PH1_ " error: flash operation failed", "suggest...") \
00094     _D(EX_JOINT_EC_MU_SAVE, 10428, "joint" _PH1_ " error: multi-turn data error", "suggest...") \
00095     _D(EX_JOINT_EC_DEMADATA_LOST, 10429, "joint" _PH1_ " error: calibration zero point data lost", "suggest...") \
00096     _D(EX_JOINT_EC_PARAMETER, 10430, "joint" _PH1_ " error: parameter error", "suggest...") \
00097     _D(EX_JOINT_EC_UVW_LOGIC, 10431, "joint" _PH1_ " error: hall signal fault", "suggest...") \
00098     _D(EX_JOINT_EC_UVW_ABZ, 10432, "joint" _PH1_ " error: incremental encoder fault", "suggest...") \
00099     _D(EX_JOINT_EC_ENC_Z_LOST, 10433, "joint" _PH1_ " error: encoder Z signal lost", "suggest...") \
00100     _D(EX_JOINT_EC_ENC_POLLUTE, 10434, "joint" _PH1_ " error: encoder pollution", "suggest...") \
00101     _D(EX_JOINT_EC_ENC_CALI, 10435, "joint" _PH1_ " error: encoder calibration failed", "suggest...") \
00102     _D(EX_JOINT_EC_MT_ABS_DATA, 10436, "joint" _PH1_ " error: multi-turn absolute data error", "suggest...") \
00103     _D(EX_JOINT_EC_ENC_TYPE_INFO, 10437, "joint" _PH1_ " error: encoder type identified and saved", "suggest...") \
00104     _D(EX_JOINT_EC_ENC_TYPE_ERROR, 10438, "joint" _PH1_ " error: encoder type error", "suggest...") \
00105     _D(EX_JOINT_EC_ENC_VERIFY, 10439, "joint" _PH1_ " error: encoder verification failed", "suggest...") \
00106     _D(EX_JOINT_EC_DUAL_ENC_ERROR, 10440, "joint" _PH1_ " error: dual encoder deviation too large", "suggest...") \
00107     _D(EX_JOINT_EC_DUAL_ENC_EANGLE, 10441, "joint" _PH1_ " error: dual encoder electrical angle deviation too large", "suggest...") \
00108     _D(EX_JOINT_EC_OBJECT_DICT_ERROR, 10442, "joint" _PH1_ " error: object dictionary data error", "suggest...") \
00109     _D(EX_JOINT_EC_MOTOR_STALL, 10443, "joint" _PH1_ " error: motor stall protection", "suggest...") \
00110     _D(EX_JOINT_EC_ABS_ENC_LOW_VOLT, 10444, "joint" _PH1_ " error: absolute encoder low voltage", "suggest...") \
00111     _D(EX_JOINT_EC_MT_BATTERY_LOW, 10445, "joint" _PH1_ " error: multi-turn battery low voltage", "suggest...") \
00112     _D(EX_JOINT_EC_POS_CMD, 10446, "joint" _PH1_ " error: position command discontinuous", "suggest...") \
00113     _D(EX_JOINT_EC_POS_OVER_LIMIT, 10447, "joint" _PH1_ " error: position over limit", "suggest...") \
00114     _D(EX_JOINT_EC_COMM_PROTO, 10448, "joint" _PH1_ " error: communication protocol layer error", "suggest...") \
00115     _D(EX_JOINT_EC_GRAVITY_PARA_WARNING, 10449, "joint" _PH1_ " error: gravity compensation parameter invalid", "suggest...") \
00116     _D(EX_JOINT_EC_GRAVITY_COMPENSATE_ERROR, 10450, "joint" _PH1_ " error: gravity compensation value sudden change", "suggest...") \
00117     _D(EX_JOINT_EC_LOW_RIGIDITY, 10451, "joint" _PH1_ " error: collision soft float abnormal", "suggest...") \
00118     _D(EX_JOINT_EC_POS_CMD_WARNING, 10452, "joint" _PH1_ " error: position command unchanged during motion", "suggest...") \
00119     _D(EX_JOINT_EC_SLAVE_COMM, 10453, "joint" _PH1_ " error: master-slave MCU communication fault", "suggest...") \
00120     _D(EX_JOINT_EC_POS_ERR, 10454, "joint" _PH1_ " error: position following error too large", "suggest...") \
```

```
00121     _D(EX_JOINT_EC_DUAL_POS_ERR, 10455, "joint" _PH1_ " error: dual servo position sync error too large", "suggest...") \
00122     _D(EX_JOINT_EC_DUAL_COMM_ERR, 10456, "joint" _PH1_ " error: dual servo communication", "suggest...") \
00123     _D(EX_JOINT_EC_CAN_BUSOFF, 10457, "joint" _PH1_ " error: CAN bus-off warning", "suggest...") \
00124     _D(EX_JOINT_EC_SYNC_SNAKE, 10458, "joint" _PH1_ " error: sync frame jitter warning", "suggest...") \
00125     _D(EX_JOINT_EC_SYNC_DISCON, 10459, "joint" _PH1_ " error: sync frame discontinuous warning", "suggest...") \
00126     _D(EX_JOINT_EC_RPDO_LOST, 10460, "joint" _PH1_ " error: RPDO lost warning", "suggest...") \
00127     _D(EX_JOINT_EC_RPDO_MANY, 10461, "joint" _PH1_ " error: multiple RPDO in one sync cycle warning", "suggest...") \
00128     _D(EX_JOINT_EC_GRAVITY_LOST, 10462, "joint" _PH1_ " error: gravity compensation value lost", "suggest...") \
00129     _D(EX_JOINT_EC_MAININT_TIME_WARN, 10463, "joint" _PH1_ " error: servo main interrupt runtime warning", "suggest...") \
00130     _D(EX_JOINT_EC_MAININT_TIME_ERROR, 10464, "joint" _PH1_ " error: servo main interrupt runtime error", "suggest...") \
00131     _D(EX_JOINT_EC_SPDINT_TIME_WARN, 10465, "joint" _PH1_ " error: servo speed loop interrupt runtime warning", "suggest...") \
00132     _D(EX_JOINT_EC_SPDINT_TIME_ERROR, 10466, "joint" _PH1_ " error: servo speed loop interrupt runtime error", "suggest...") \
00133     _D(EX_JOINT_EC_POS_CMD_JUMP_WARNING, 10467, "joint" _PH1_ " error: position command sudden jump during motion", "suggest...") \
00134     _D(EX_JOINT_EC_SYNC_TIMCOARSE, 10468, "joint" _PH1_ " error: clock sync compensation status", "suggest...") \
00135     _D(EX_JOINT_EC_ENC_Z_CNTS_ERR, 10469, "joint" _PH1_ " error: incremental encoder Z signal or CNTS abnormal", "suggest...") \
00136     _D(EX_JOINT_EC_SLAVE_OVER_CURRENT, 10470, "joint" _PH1_ " error: slave MCU detected motor phase current over safe threshold", "suggest...") \
00137     _D(EX_JOINT_EC_SLAVE_OVER_VOLTAGE, 10471, "joint" _PH1_ " error: slave MCU detected DC bus voltage over upper threshold", "suggest...") \
00138     _D(EX_JOINT_EC_SLAVE_UNDER_VOLTAGE, 10472, "joint" _PH1_ " error: slave MCU detected DC bus voltage below lower threshold", "suggest...") \
00139     _D(EX_JOINT_EC_SLAVE_POS_ERR, 10473, "joint" _PH1_ " error: slave MCU detected master-slave motor position deviation out of range", "suggest...") \
00140     _D(EX_JOINT_EC_SLAVE_SPEED_ERR, 10474, "joint" _PH1_ " error: slave MCU detected master-slave motor speed deviation out of range", "suggest...") \
00141     _D(EX_JOINT_EC_SLAVE_TRACE_ERR, 10475, "joint" _PH1_ " error: slave MCU detected position following error out of control range", "suggest...") \
00142     _D(EX_JOINT_EC_SLAVE_ABS_ERR, 10476, "joint" _PH1_ " error: slave MCU detected absolute encoder feedback abnormal", "suggest...") \
00143     _D(EX_JOINT_EC_SLAVE_ADC_ZERO_OFFSET, 10477, "joint" _PH1_ " error: slave MCU detected current sampling zero offset out of calibration range", "suggest...") \
00144     _D(EX_JOINT_EC_SLAVE_ENC_POLLUTE, 10478, "joint" _PH1_ " error: slave MCU detected encoder signal quality degradation", "suggest...") \
00145     _D(EX_JOINT_EC_SLAVE_ENC_Z_LOST, 10479, "joint" _PH1_ " error: slave MCU detected encoder Z reference signal lost", "suggest...") \
00146     _D(EX_JOINT_EC_SLAVE_COMM_OVER_TM, 10480, "joint" _PH1_ " error: slave MCU detected master-slave communication timeout", "suggest...") \
00147     _D(EX_JOINT_EC_SLAVE_TRQ_ERR, 10481, "joint" _PH1_ " error: slave MCU detected master-slave motor torque deviation out of range", "suggest...") \
00148     _D(EX_JOINT_EC_BRAKE_TYPE_ERR, 10482, "joint" _PH1_ " error: brake type config mismatch with hardware", "suggest...") \
00149     _D(EX_JOINT_EC_PHASE_ALIGN, 10483, "joint" _PH1_ " error: phase alignment failed", "suggest...") \
00150     _D(EX_JOINT_EC_POS_OVER_LIMIT_WARNING, 10484, "joint" _PH1_ " error: position over limit", "suggest...") \
00151     _D(EX_JOINT_EC_PHASE_ALIGN_WARNING, 10485, "joint" _PH1_ " error: phase alignment warning", "suggest...") \
00152     _D(EX_JOINT_EC_TASK_STACK_SHORTAGE, 10486, "joint" _PH1_ " error: task stack insufficient", "suggest...") \
00153     _D(EX_JOINT_EC_TASK_STACK_OVERFLOW, 10487, "joint" _PH1_ " error: task stack overflow", "suggest...") \
00154     _D(EX_JOINT_EC_UNKNOWN, 10800, "joint" _PH1_ " error: unknown error", "suggest...")
00155
00156 #define EXT_AXIS_ERRORS                                                
```

```
calibration; check encoder.") \
00170     _D(EXT_AXIS_ERR_IMU,                11014, "ext axis" _PH1_ " error: IMU fault", "Check IMU sensor and
connection.") \
00171     _D(EXT_AXIS_ERR_TEMP_SENSOR,        11015, "ext axis" _PH1_ " error: temperature sensor fault", "Check
temp sensor wiring; reboot.") \
00172     _D(EXT_AXIS_ERR_ECAT_BUS,           11016, "ext axis" _PH1_ " error: EtherCAT bus error", "Check
EtherCAT cabling/topology/sync; reboot master/drive.") \
00173     _D(EXT_AXIS_ERR_ECAT_CONFIG,        11017, "ext axis" _PH1_ " error: EtherCAT config/ESI/SM/PDO
fault", "Check ESI, Mailbox/SM/PDO mapping, vendor/product/revision match.") \
00174     _D(EXT_AXIS_ERR_ECAT_SYNC,          11018, "ext axis" _PH1_ " error: EtherCAT sync/frame/period fault",
"Check DC sync, cycle time, frame loss; verify NIC/IRQ affinity.") \
00175     _D(EXT_AXIS_ERR_SYS_CUR,            11019, "ext axis" _PH1_ " error: system current fault", "Check current
loop and load; reboot.") \
00176     _D(EXT_AXIS_ERR_SYS_POS,            11020, "ext axis" _PH1_ " error: position out of range", "Check
encoder/scale/limits; reboot.") \
00177     _D(EXT_AXIS_ERR_OVER_SPEED,         11021, "ext axis" _PH1_ " error: over speed", "Check command limits
and tuning parameters.") \
00178     _D(EXT_AXIS_ERR_OVER_ACC,           11022, "ext axis" _PH1_ " error: over acceleration", "Reduce
acceleration/jerk; check tuning.") \
00179     _D(EXT_AXIS_ERR_FOLLOW_ERROR,       11023, "ext axis" _PH1_ " error: following error", "Check gains,
load, saturation; verify feedback.") \
00180     _D(EXT_AXIS_ERR_TAG_POS_OVER,       11024, "ext axis" _PH1_ " error: target position out of range",
"Check target limits and homing.") \
00181     _D(EXT_AXIS_ERR_TAG_SPEED_OVER,     11025, "ext axis" _PH1_ " error: target speed out of range",
"Clamp speed; check profile settings.") \
00182     _D(EXT_AXIS_ERR_TAG_CURRENT_OVER,   11026, "ext axis" _PH1_ " error: target current out of range",
"Clamp current/torque; check load.") \
00183     _D(EXT_AXIS_ERR_COLLISION,          11027, "ext axis" _PH1_ " error: collision", "Remove obstruction; check
torque/force limits.") \
00184     _D(EXT_AXIS_ERR_ADC_ZERO_OFFSET,    11028, "ext axis" _PH1_ " error: ADC zero offset", "Reboot; check
ADC/current sensor offset.") \
00185     _D(EXT_AXIS_ERR_IPM_NTC,            11029, "ext axis" _PH1_ " error: IPM NTC fault", "Check power module
temperature sensing.") \
00186     _D(EXT_AXIS_ERR_SHORT_CIRCUIT,      11030, "ext axis" _PH1_ " error: short circuit", "Check motor phase
wiring; insulation test.") \
00187     _D(EXT_AXIS_ERR_MOTOR_STALL,        11031, "ext axis" _PH1_ " error: motor stall", "Check mechanical
jam/load; reduce accel; reboot.") \
00188     _D(EXT_AXIS_ERR_ABS_MULTITURN,      11032, "ext axis" _PH1_ " error: abs encoder multiturn fault",
"Check abs encoder battery/params; reboot.") \
00189     _D(EXT_AXIS_ERR_PHASE_LOST,         11033, "ext axis" _PH1_ " error: motor phase lost", "Check phase
wiring/connector; measure continuity.") \
00190     _D(EXT_AXIS_ERR_BRAKE,              11034, "ext axis" _PH1_ " error: brake fault", "Check brake wiring/power;
verify brake release.") \
00191     _D(EXT_AXIS_ERR_REDUCER_OVER_TEMP,  11035, "ext axis" _PH1_ " error: reducer over temperature",
"Check reducer temperature/cooling.") \
00192     _D(EXT_AXIS_ERR_REDUCER_NTC,        11036, "ext axis" _PH1_ " error: reducer NTC fault", "Check reducer
temperature sensor.") \
00193     _D(EXT_AXIS_ERR_FIRMWARE_UPDATE,    11037, "ext axis" _PH1_ " error: firmware update fault", "Retry
update; check power stability.") \
00194     _D(EXT_AXIS_ERR_FLASH_OP,           11038, "ext axis" _PH1_ " error: flash operation fault", "Retry; if
persistent replace drive.") \
00195     _D(EXT_AXIS_ERR_EXT_ABS_ENC,        11039, "ext axis" _PH1_ " error: motor-side abs encoder comm fault",
"Check external abs encoder link/power.") \
00196     _D(EXT_AXIS_ERR_DRIVE_FAULT,        11040, "ext axis" _PH1_ " error: drive fault", "Check drive alarm code;
reboot; replace if repeated.") \
00197     _D(EXT_AXIS_ERR_OVERLOAD,           11041, "ext axis" _PH1_ " error: overload", "Reduce load; check
mechanics and tuning.") \
00198     _D(EXT_AXIS_ERR_HARDWARE_LIMIT,     11042, "ext axis" _PH1_ " error: hardware limit triggered", "Move
away from limit; check limit switch.") \
00199     _D(EXT_AXIS_ERR_SERVO_MODE_TIMEOUT, 11043, "ext axis" _PH1_ " error: switch servo mode timeout",
"Check mode transition and comm; reboot.") \
00200     _D(EXT_AXIS_ERR_UVW_ABZ,            11044, "ext axis" _PH1_ " error: UVW/ABZ fault", "Check
phase/encoder signals wiring.") \
00201     _D(EXT_AXIS_ERR_BATTERY_LOW,        11045, "ext axis" _PH1_ " error: battery low", "Replace encoder
battery; reboot.") \
00202     _D(EXT_AXIS_ERR_PHASE_ALIGN,        11046, "ext axis" _PH1_ " error: phase align fail", "Redo phase
alignment; check motor params.") \
00203     _D(EXT_AXIS_ERR_POS_DISCONTINUOUS,  11047, "ext axis" _PH1_ " error: position command
discontinuous", "Check trajectory generation and limits.") \
00204     _D(EXT_AXIS_ERR_POS_INIT,           11048, "ext axis" _PH1_ " error: position initialization failure", "Check
encoder init/homing procedure.") \
00205     _D(EXT_AXIS_ERR_TORQUE_SENSOR,      11049, "ext axis" _PH1_ " error: torque sensor fault", "Check torque
sensor wiring/calibration.") \
00206     _D(EXT_AXIS_ERR_ABS_ENC_LOW_VOLT,   11050, "ext axis" _PH1_ " error: abs encoder low voltage",
"Check encoder supply voltage and cable.") \
00207     _D(EXT_AXIS_ERR_OFFLINE,            11051, "ext axis" _PH1_ " error: ext axis offline", "Check hardware and
axis id; check EtherCAT state.") \
00208     _D(EXT_AXIS_ERR_BOOTLOADER,         11052, "ext axis" _PH1_ " error: ext axis in bootloader", "Retry
firmware update.") \
00209     _D(EXT_AXIS_ERR_SLAVE_OFFLINE,      11053, "ext axis slave" _PH1_ " error: slave offline", "Check slave
hardware and id.") \
00210     _D(EXT_AXIS_ERR_SLAVE_BOOTLOADER,   11054, "ext axis slave" _PH1_ " error: slave in bootloader",
"Retry firmware update.") \
00211     _D(EXT_AXIS_ERR_EEPROM,             11055, "ext axis" _PH1_ " error: EEPROM/param store fault", "Check
EEPROM/parameter storage; power cycle.") \
00212     _D(EXT_AXIS_ERR_PARAM_CONFIG,       11056, "ext axis" _PH1_ " error: parameter/config fault", "Verify
parameter set; restore defaults if needed.") \
```

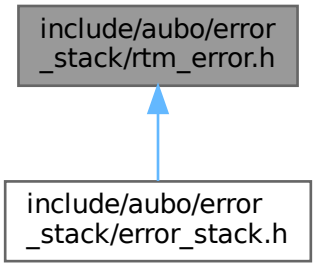
```
00213     _D(EXT_AXIS_ERR_STO,                11057, "ext axis" _PH1_ " error: STO safety fault", "Check STO wiring/safety chain; reset safety.") \
00214     _D(EXT_AXIS_ERR_ENCRYPT_CHIP,       11058, "ext axis" _PH1_ " error: encrypt chip/key fault", "Check encryption chip/keys/firmware compatibility.") \
00215     _D(EXT_AXIS_ERR_BRAKE_RES_OVERLOAD, 11059, "ext axis" _PH1_ " error: brake resistor overload", "Check brake resistor/regen circuit; duty cycle.") \
00216     _D(EXT_AXIS_ERR_POWER_LINE_OPEN,    11060, "ext axis" _PH1_ " error: motor power line open", "Check motor power cable continuity/connector.") \
00217     _D(EXT_AXIS_ERR_HOMING,             11061, "ext axis" _PH1_ " error: homing fault", "Check homing sensor/origin procedure; retry.") \
00218     _D(EXT_AXIS_ERR_TUNING_FAIL,        11062, "ext axis" _PH1_ " error: tuning fail", "Redo tuning; reduce resonance; check mechanics.") \
00219     _D(EXT_AXIS_ERR_INERTIA_ID_FAIL,    11063, "ext axis" _PH1_ " error: inertia identification fail", "Check load; redo inertia ID; adjust conditions.") \
00220     _D(EXT_AXIS_ERR_FLYAWAY,            11064, "ext axis" _PH1_ " error: flyaway", "Emergency stop; check feedback polarity/scale; inspect drive params.") \
00221     _D(EXT_AXIS_ERR_SPEED_PULSE_OVER,   11065, "ext axis" _PH1_ " error: feedback pulse overspeed", "Check encoder feedback and scaling; reduce speed.") \
00222     _D(EXT_AXIS_ERR_CTRL_LOOP,          11066, "ext axis" _PH1_ " error: control loop/timeout fault", "Check sampling/current loop/comm timeout; reboot.")
00223
00224 #define TOOL_ERRORS \
00225     _D(TOOL_FLASH_VERIFY_FAILED, 40001, "Flash write verify failed", "suggest...") \
00226     _D(TOOL_PROGRAM_CRC_FAILED, 40002, "Program flash checksum failed during bootloading", "suggest...") \
00227     _D(TOOL_PROGRAM_CRC_FAILED2, 40003, "Program flash checksum failed at runtime", "suggest...") \
00228     _D(TOOL_ID_UNDIFINED, 40004, "Tool ID is undefined", "suggest...") \
00229     _D(TOOL_ILLEGAL_BL_CMD, 40005, "Illegal bootloader command", "suggest...") \
00230     _D(TOOL_FW_WRONG, 40006, "Wrong firmware at the joint", "suggest...") \
00231     _D(TOOL_HW_INVALID, 40007, "Invalid hardware revision", "suggest...") \
00232     _D(TOOL_SHORT_CURCUIT_H, 40011, "Short circuit detected on Digital Output: " _PH1_ " high side", "suggest...") \
00233     _D(TOOL_SHORT_CURCUIT_L, 40012, "Short circuit detected on Digital Output: " _PH1_ " low side", "suggest...") \
00234     _D(TOOL_AVERAGE_CURR_HIGH, 40013, "10 second Average tool IO Current of " _PH1_ " A is outside of the allowed range.", "suggest...") \
00235     _D(TOOL_POWER_PIN_OVER_CURR, 40014, "Current of " _PH1_ " A on the POWER pin is outside of the allowed range.", "suggest...") \
00236     _D(TOOL_DOUT_PIN_OVER_CURR, 40015, "Current of " _PH1_ " A on the Digital Output pins is outside of the allowed range.", "suggest...") \
00237     _D(TOOL_GROUND_PIN_OVER_CURR, 40016, "Current of " _PH1_ " A on the ground pin is outside of the allowed range.", "suggest...") \
00238     _D(TOOL_RX_FRAMING, 40021, "RX framing error", "suggest...") \
00239     _D(TOOL_RX_PARITY, 40022, "RX Parity error", "suggest...") \
00240     _D(TOOL_48V_LOW, 40031, "48V input is too low", "suggest...") \
00241     _D(TOOL_48V_HIGH, 40032, "48V input is too high", "suggest...") \
00242     _D(TOOL_ERR_OFFLINE,  40033, "tool error: tool may be offline", "(a) Check tool's hardware. (b) Check joint's id.") \
00243     _D(TOOL_ERR_BOOTLOADER,  40034, "tool error: The tool is in bootloader mode. Retry firmware update. ", "suggest...")
00244
00245 #define EX_TOOL_ERRORS \
00246     _D(EX_TOOL_EC_LOW_VOLTAGE, 40101, "tool error: low voltage", "check tool power supply voltage") \
00247     _D(EX_TOOL_EC_FORCESENSOR_COMM, 40102, "tool error: external force sensor communication error", "check external force sensor communication connection") \
00248     _D(EX_TOOL_485_SENDFULL_COMM, 40103, "tool error: 485 transparent transmission buffer full", "check 485 communication load and transmission frequency") \
00249     _D(EX_TOOL_FORCESENSOR_FILTER_ZERO, 40104, "tool error: force sensor filter parameter is zero", "check force sensor filter parameter configuration") \
00250     _D(EX_TOOL_EC_UNKNOWN, 40200, "tool error: unknown error", "check controller logs and hardware status")
00251
00252 #define PEDSTRAL_ERRORS \
00253     _D(PKG_LOST, 50001, "Lost package from pedestal", "suggest...") \
00254     _D(PEDSTRAL_OFFLINE,  50002, "pedestal error: pedestal may be offline", "(a) Check pedestal's hardware. (b) Check pedestal's id.") \
00255     _D(PEDESTAL_ERR_BOOTLOADER,  50003, "pedestal error: The pedestal is in bootloader mode. Retry firmware update. ", "suggest...")
00256
00257 #define EX_PEDSTRAL_ERRORS \
00258     _D(EX_BASE_EC_LOW_VOLTAGE, 50101, "pedstral error: low voltage", "check power supply voltage and battery status") \
00259     _D(EX_BASE_EC_OVER_TEMPERATURE_RES, 50102, "pedstral error: resistor over temperature", "check braking resistor temperature and cooling condition") \
00260     _D(EX_BASE_EC_OVER_TARGET_BRAKE_OPEN_VOLT, 50103, "pedstral error: input voltage close to or exceeds regenerative brake activation voltage", "check input power voltage and regenerative braking configuration") \
00261     _D(EX_BASE_EC_RES_BREAKAGE, 50104, "pedstral error: brake resistor breakage", "check whether the brake resistor is disconnected or damaged") \
00262     _D(EX_BASE_EC_IMU_CALIBRATE, 50105, "pedstral error: IMU calibration required", "perform IMU calibration according to maintenance procedure") \
00263     _D(EX_BASE_EC_TEMP_SENSOR_SHORT, 50106, "pedstral error: temperature sensor short circuit", "check temperature sensor wiring and solder joints for short circuit") \
00264     _D(EX_BASE_EC_TEMP_SENSOR_BREAK, 50107, "pedstral error: temperature sensor open circuit", "check temperature sensor connection and cable continuity") \
00265     _D(EX_BASE_EC_96V_INSTANT_OVER_VOLTAGE, 50108, "pedstral error: 96V instantaneous over voltage after regenerative braking", "check regenerative braking behavior and power bus voltage") \
00266     _D(EX_BASE_EC_UNKNOWN, 50200, "pedstral error: unknown error", "check controller logs and hardware status")
00267
00268 #define SAFETY_INTERFACE_BOARD_ERRORS \
```

```
00269     _D(IFB_ERR_ROBOTTYPE, 20001, "Robot error type!", "suggest...") \
00270     _D(IFB_ERR_ADXL_SENS, 20002, "Base Acceleration sensor error!", "suggest...") \
00271     _D(IFB_ERR_EN_LINE, 20003, "Encoder line error!", "suggest...") \
00272     _D(IFB_ERR_ENTER_HDG_MODE, 20004, "Robot enter handguide mode!", "suggest...") \
00273     _D(IFB_ERR_EXIT_HDG_MODE, 20005, "Robot exit handguide mode!", "suggest...") \
00274     _D(IFB_ERR_MAC_DATA_BREAK, 20006, "MAC data break!", "suggest...") \
00275     _D(IFB_ERR_DRV_FIRMWARE_VERSION, 20007, "Motor driver firmware version error!", "suggest...") \
00276     _D(INIT_ERR_EN_DRV, 20008, "Motor driver enable failed!", "suggest...") \
00277     _D(INIT_ERR_EN_AUTO_BACK, 20009, "Motor driver enable auto back failed!", "suggest...") \
00278     _D(INIT_ERR_EN_CUR_LOOP, 20010, "Motor driver enable current loop failed!", "suggest...") \
00279     _D(INIT_ERR_SET_TAG_CUR, 20011, "Motor driver set target current failed!", "suggest...") \
00280     _D(INIT_ERR_RELEASE_BRAKE, 20012, "Motor driver release brake failed!", "suggest...") \
00281     _D(INIT_ERR_EN_POS_LOOP, 20013, "Motor driver enable postion loop failed!", "suggest...") \
00282     _D(INIT_ERR_SET_MAX_ACC, 20014, "Motor set max accelerate failed!", "suggest...") \
00283     _D(SAFETY_ERR_PROTECTION_STOP_TIMEOUT, 20015, "Protective stop timeout!", "suggest...") \
00284     _D(SAFETY_ERR_REDUCED_MODE_TIMEOUT, 20016, "Reduced mode timeout!", "suggest...") \
00285     _D(SYS_ERR_MCU_COM, 20017, "Robot system error: mcu communication error!", "suggest...") \
00286     _D(SYS_ERR_RS485_COM, 20018, "Robot system error: RS485 communication error!", "suggest...") \
00287     _D(IFB_ERR_DISCONNECTED, 20019, "Interface board may be disconnected. Please check connection between IPC
          and Interface board.", "suggest...")\
00288     _D(IFB_ERR_PAYLOAD_ERROR, 20020, "Payload error.", "suggest...") \
00289     _D(IFB_OFFLINE,  20021, "ifaceboard error: ifaceboard may be offline", "(a) Check ifaceboard's hardware. (b) Check
          ifaceboard's id.") \
00290     _D(IFB_ERR_BOOTLOADER,  20022, "ifaceboard error: The ifaceboard is in bootloader mode. Retry firmware
          update. ", "suggest...") \
00291     _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.") \
00292     _D(IFB_SLAVE_ERR_BOOTLOADER,  20024, "interface slave board error: The interface slave board is in
          bootloader mode. Retry firmware update. ", "suggest...") \
00293     _D(IFB_TOOL_ERR_ADXL_SENS, 20025, "Tool Acceleration sensor error!", "suggest...") \
00294     _D(HANDLE_OFFLINE,  20026, "handle error: handle may be offline", "(a) Check handle's hardware. (b) Check
          handle's id.") \
00295     _D(HANDLE_ERR_BOOTLOADER,  20027, "handle error: The handle is in bootloader mode. Retry firmware update.
          ", "suggest...") \
00296     _D(IFB_POWERLOSS_OFFLINE,  20028, "interface powerloss board error: interface powerloss board may be offline",
          "(a) Check interface powerloss board's hardware. (b) Check interface powerloss board's id.") \
00297     _D(IFB_POWERLOSS_ERR_BOOTLOADER,  20029, "interface powerloss board error: The interface powerloss
          board is in bootloader mode. Retry firmware update. ", "suggest...") \
00298     _D(HANDLE_COMM_ERROR,  20030, "handle error: handle comm error", "(a) Check handle's hardware. (b) Check
          handle's id.")
00299
00300 #define HARDWARE_INTERFACE_ERRORS \
00301     _D(HW_SCB_SETUP_FAILED, 60001, "Setup of Interface Board failed", "suggest...") \
00302     _D(HW_PKG_CNT_DISAGEE, 60002, "Packet counter disagreements", "suggest...") \
00303     _D(HW_SCB_DISCONNECT, 60003, "Connection to Interface Board lost", "suggest...") \
00304     _D(HW_SCB_PKG_LOST, 60004, "Package lost from Interface Board", "suggest...") \
00305     _D(HW_SCB_CONN_INIT_FAILED, 60005, "Ethernet connection initialization with Interface Board failed",
          "suggest...") \
00306     _D(HW_LOST_JOINT_PKG, 60006, "Lost package from joint  " __PH1__ "", "suggest...") \
00307     _D(HW_LOST_TOOL_PKG, 60007, "Lost package from tool", "suggest...") \
00308     _D(HW_JOINT_PKG_CNT_DISAGREE, 60008, "Packet counter disagreement in packet from joint " __PH1__ "",
          "suggest...") \
00309     _D(HW_TOOL_PKG_CNT_DISAGREE, 60009, "Packet counter disagreement in packet from tool", "suggest...") \
00310     _D(HW_JOINTS_FAULT, 60011, "" __PH1__ " joint entered the Fault State", "suggest...") \
00311     _D(HW_JOINTS_VIOLATION, 60012, "" __PH1__ " joint entered the Violation State", "suggest...") \
00312     _D(HW_TP_FAULT, 60013, "Teach Pendant entered the Fault State", "suggest...") \
00313     _D(HW_TP_VIOLATION, 60014, "Teach Pendant entered the Violation State", "suggest...") \
00314     _D(HW_JOINT_MV_TOO_FAR, 60021, "" __PH1__ " joint moved too far before robot entered RUNNING State",
          "suggest...") \
00315     _D(HW_JOINT_STOP_NOT_FAST, 60022, "Joint Not stopping fast enough", "suggest...") \
00316     _D(HW_JOINT_MV_LIMIT, 60023, "Joint moved more than allowable limit", "suggest...") \
00317     _D(HW_FT_SENSOR_DATA_INVALID, 60024, "Force-Torque Sensor data invalid", "suggest...") \
00318     _D(HW_NO_FT_SENSOR, 60025, "Force-Torque sensor is expected, but it cannot be detected", "suggest...") \
00319     _D(HW_FT_SENSOR_NOT_CALIB, 60026, "Force-Torque sensor is detected but not calibrated", "suggest...") \
00320     _D(HW_RELEASE_BRAKE_FAILED, 60030, "Robot was not able to brake release, see log for details", "suggest...") \
00321     _D(HW_OVERCURR_SHUTDOWN, 60040, "Overcurrent shutdown", "suggest...") \
00322     _D(HW_ENERGEY_SURPLUS, 60050, "Energy surplus shutdown", "suggest...") \
00323     _D(HW_IDLE_POWER_HIGH, 60060, "Idle power consumption to high", "suggest...") \
00324     _D(HW_ENTER_COLLISION_TIMEOUT, 60071, "Enter collision stop procedure timeout", "suggest...") \
00325     _D(HW_POWERON_TIMEOUT, 60072, "Poweron robot timeout", "suggest...") \
00326     _D(HW_NO_NIC_FOUND, 60073, "No network cards found.", "suggest...") \
00327     _D(HW_IFB_NOT_FOUND, 60074, "No Interface Board found.", "suggest...") \
00328     _D(HW_IFB_BOOTLOAD, 60075, "The Interface Board is in bootloader mode. Update firmware firstly.", "suggest...")
          \
00329     _D(HW_TOOL_NOT_FOUND, 60076, "No Tool Board found.", "suggest...") \
00330     _D(HW_BASE_NOT_FOUND, 60077, "No Base Board found.", "suggest...") \
00331     _D(HW_BRINGUP_TIMEOUT, 60078, "Poweron robot timeout", "suggest...") \
00332     _D(HW_COLLISION_RECOVERY_FAILED, 60079, "Collision recovery failed", "suggest...") \
00333     _D(HW_TP_ENABLED, 60080, "Teach pendant enabled status changed to " __PH1__, "suggest...")
00334
00335 // clang-format on
00336
00337 //
00338 #define HAL_ERRORS              \
00339     JOINT_ERRORS                \
00340     JOINT_EX_ERRORS               \
00341     EXT_AXIS_ERRORS               \
```

```
00342     SAFETY_INTERFACE_BOARD_ERRORS \
00343     TOOL_ERRORS                   \
00344     EX_TOOL_ERRORS                \
00345     PEDSTRAL_ERRORS               \
00346     EX_PEDSTRAL_ERRORS            \
00347     HARDWARE_INTERFACE_ERRORS
00348
00349 #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
      mounting settings") \
00011     _D(PSTOP_ELBOW_POS, 30002, "Protective Stop: Elbow position close to safety plane limits.","Please move robot
      Elbow joint away from the safety plane") \
00012     _D(PSTOP_STOP_TIME, 30003, "Protective Stop: Exceeding user safety settings for stopping time.","(a) Check
      speeds and accelerations in the program (b) Check usage of TCP,payload and CoG correctly (c) Check external
      equipmentactivation if correctly set") \
00013     _D(PSTOP_STOP_DISTANCE, 30004, "Protective Stop: Exceeding user safety settings for stopping distance.","(a)
      Check speeds and accelerations in the program (b) Check usage of TCP,payload and CoG correctly (c) Check external
      equipmentactivation if correctly set") \
00014     _D(PSTOP_CLAMP, 30005, "Protective Stop: Danger of clamping between the Robot's lower arm and tool.","(a)
      Check speeds and accelerations in the program (b) Check usage of TCP,payload and CoG correctly (c) Check external
      equipmentactivation if correctly set") \
00015     _D(PSTOP_POS_LIMIT, 30006, "Protective Stop: Position close to joint limits", "suggest...") \
00016     _D(PSTOP_ORI_LIMIT, 30007, "Protective Stop: Tool orientation close to limits", "suggest...") \
00017     _D(PSTOP_PLANE_LIMIT, 30008, "Protective Stop: Position close to safety plane limits", "suggest...") \
00018     _D(PSTOP_POS_DEVIATE, 30009, "Protective Stop: Position deviates from path", "Check payload, center of gravity
      and acceleration settings.") \
00019     _D(JOINT_CHK_PAYLOAD, 30010, "Joint " _PH1_ ": Check payload, center of gravity and acceleration settings.
      Log screen may contain additional information.", "suggest...") \
00020     _D(PSTOP_SINGULARITY, 30011, "Protective Stop: Position in singularity.","Please use MoveJ or change the
      motion") \
00021     _D(PSTOP_CANNOT_MAINTAIN, 30012, "Protective Stop: Robot cannot maintain its position, check if payload is
      correct", "suggest...") \
00022     _D(PSTOP_WRONG_PAYLOAD, 30013, "Protective Stop: Wrong payload or mounting detected, or something is
      pushing the robot when entering Freedrive mode","Verify that the TCP configuration and mounting in the used
      installation is correct") \
00023     _D(PSTOP_JOINT_COLLISION, 30014, "Protective Stop: Collision detected by joint " _PH1_, "Make sure no
      objects are in the path of the robot and resume the program") \
00024     _D(PSTOP_POS_DISAGREE, 30015, "Protective stop: The robot was powered off last time due to a joint position
      disagreement.","(a) Verify that the robot position in the 3D graphics matches the real robot, to ensure that the encoders
      function before releasing the brakes. Stand back and monitor the robot performing its first program cycle as expected. (b)
      If the position is not correct, the robot must be repaired. In this case, click Power Off Robot. (c) If the position is correct,
      please tick the check box below the 3D graphics and click Robot Position Verified") \
00025     _D(TARGET_JOINT_SPEED_EXCEED, 30016, "Target joint speed exceed limits", "suggest...") \
00026     _D(TARGET_POS_SUDDEN_CHG, 30017, "Sudden change in target position", "suggest...") \
00027     _D(SUDDEN_STOP, 30018, "Sudden stop.","To abort a motion, use \"stopj\" or \"stopl\" script commands to
      generate a smooth deceleration before using \"wait\". Avoid aborting motions between waypoints with blend"") \
00028     _D(ROBOT_STOP_ABNORMAL, 30019, "Robot has not stopped in the allowed reaction and braking time",
      "suggest...") \
00029     _D(PROG_INVALID_SETP, 30020, "Robot program resulted in invalid setpoint.", "Please review waypoints in the
      program") \
00030     _D(BLEND_INVALID_SETP, 30021, "Blending failed and resulted in an invalid setpoint.", "Try changing the blend
      radius or contact technical support") \
00031     _D(APPROACH_SINGULARITY, 30022, "Robot approaching singularity – Acceleration threshold failed.","Review
      waypoints in the program, try using MoveJ instead of MoveL in the position close to singularity") \
00032     _D(TSPEED_UNMATCH_POS, 30023, "Target speed does not match target position", "suggest...") \
00033     _D(INCONSIS_TPOS_SPD, 30024, "Inconsistency between target position and speed", "suggest...") \
00034     _D(JOINT_TSPD_UNMATCH_POS, 30025, "Target joint speed does not match target joint position change – Joint "
      _PH1_ "", "suggest...") \
00035     _D(FIELDBUS_INPUT_DISCONN, 30026, "Fieldbus input disconnected.","Please check fieldbus connections (RTDE,
      ModBus, EtherNet/IP and Profinet) or disable the fieldbus in the installation. Check RTDE watchdog feature. Check if a
      URCap is using this feature.") \
00036     _D(OPMODE_CHANGED, 30027, "Operational mode changed: " _PH1_ "", "suggest...") \
00037     _D(NO_KIN_CALIB, 30028, "No Kinematic Calibration found (calibration.conf file is either corrupt or missing).","A
      new kinematics calibration may be needed if the robot needs to improve its kinematics, otherwise, ignore this message)") \
00038     _D(KIN_CALIB_UNMATCH_JOINT, 30029, "Kinematic Calibration for the robot does not match the joint(s).", "If
      moving a program from a different robot to this one, rekinematic calibrate the second robot to improve kinematics,
      otherwise ignore this message.") \
00039     _D(KIN_CALIB_UNMATCH_ROBOT, 30030, "Kinematic Calibration does not match the robot.","Please check if the
      serial number of the robot arm matches the Control Box") \
00040     _D(JOINT_OFFSET_CHANGED, 30031, "Large movement of the robot detected while it was powered off. The joints
      were moved while it was powered off, or the encoders do not function", "suggest...") \
00041     _D(OFFSET_CHANGE_HIGH, 30032, "Change in offset is too high", "suggest...") \
00042     _D(JOINT_SPEED_LIMIT, 30033, "Close to joint speed safety limit.", "Review program speed and acceleration") \
00043     _D(TOOL_SPEED_LIMIT, 30034, "Close to tool speed safety limit.", "Review program speed and acceleration") \
00044     _D(MOMENTUM_LIMIT, 30035, "Close to momentum safety limit.", "Review program speed and acceleration") \
00045     _D(ROBOT_MV_STOP, 30036, "Robot is moving when in Stop Mode", "suggest...") \
00046     _D(HAND_PROTECTION, 30037, "Hand protection: Tool is too close to the lower arm: " _PH1_ " meter.","(a)
      Check wrist position. (b) Verify mounting (c) Do a Complete rebooting sequence (d) Update software (e) Contact your
      local AUBO Robots service provider for assistance") \
00047     _D(WRONG_SAFETYMODE, 30038, "Wrong safety mode: " _PH1_, "suggest...") \
00048     _D(SAFETYMODE_CHANGED, 30039, "Safety mode changed: " _PH1_, "suggest...") \
```

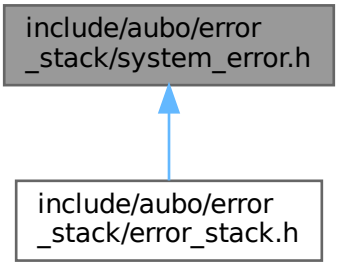
```
00049     _D(JOINT_ACC_LIMIT, 30040, "Close to joint acceleration safety limit", "suggest...") \
00050     _D(TOOL_ACC_LIMIT, 30041, "Close to tool acceleration safety limit", "suggest...") \
00051     _D(JOINT_TEMPERATURE_LIMIT, 30042, "Joint " _PH1_ " temperature too high(>" _PH2_ "℃)", "suggest...") \
00052     _D(CONTROL_BOX_TEMPERATURE_LIMIT, 30043, "Control box temperature too high(>" _PH1_ "℃)",
          "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, "SI0/SI1 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 foece 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     _D(WRIST_SINGULARITY_RISK, 30443,"Wrist singularity detected. Linear motion may cause excessive joint
          speed.Adjust robot posture to avoid J5 near 0° or 180°, or use joint motion instead of linear motion.","suggest...") \
00179     _D(PSTOP_PATH_OFFSET_OVER_LIMIT, 30444, "Protective Stop: the offset of the robot has exceeded the limit.",
          "suggest...") \
00180     _D(EXT_AXIS_SET_PARAM_FAILED_BUSY, 30445, "Set external axis parameter " _PH1_ " failed because the
          robot mode is in " _PH2_ ". Please power off the robot before changing this parameter.", "Power off the robot and try
          again.") \
00181     _D(WRONG_JOINT_POS_LIMIT, 30446, "Joint limit max pos less than min pos.", "suggest...") \
00182     _D(WRONG_JOINT_VEL_LIMIT, 30447, "Joint limit max vel is invalid.", "suggest...")
00183 // clang-format on
00184
00185 #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

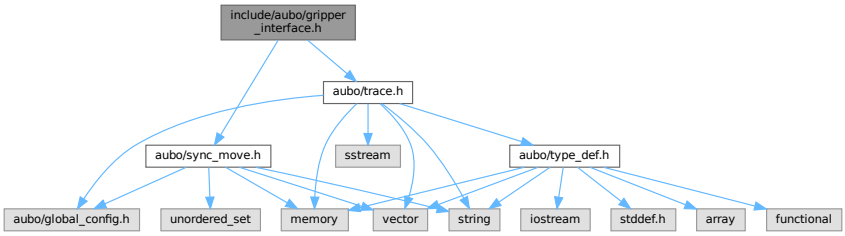
Go to the documentation of this file.

```
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.",       \
```

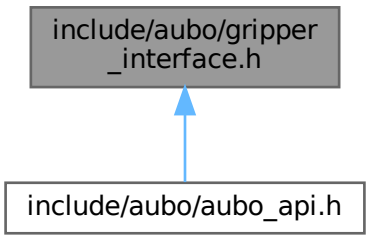
```
00017       "suggest...")                                                   \
00018   _D(MODBUS_SIGNAL_REMOVED, 11, "Modbus signal " _PH1_ " removed.",          \
00019       "suggest...")                                                   \
00020   _D(MODBUS_SIGNAL_VALUE_CHANGED, 12,                                        \
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 isVisibled
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    /**
```

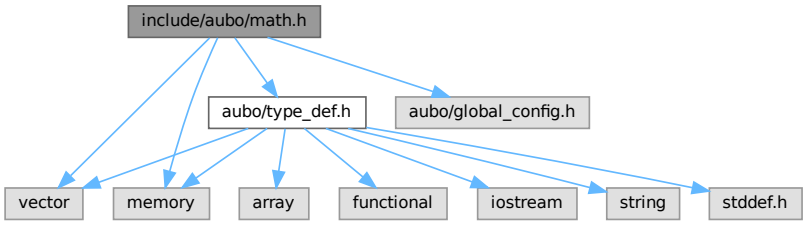
```
00191     * @ingroup GripperInterface
00192     * \chinese
00193     *
00194     *
00195     * @param name
00196     * @param position      m/Pa
00197     * @param velocity_percent      %   [0, 1]
00198     * @param force       N
00199     * @param angle       rad
00200     * @param r_velocity_percent       %   [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
```

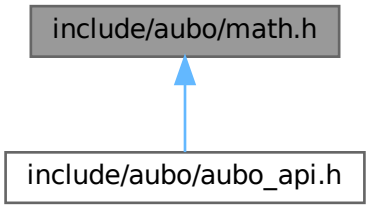
12.16 include/aubo/math.h File Reference

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     * \endengish
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_from_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     * 1.5699999989900404,-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     * 1.5699999989900404,-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      *
00497      * @param p1   1
00498      * @param p2   2
00499      * @return
00500      *
00501      * @par Lua
00502      * poseAngleDistance(p1: table, p2: table) -> number
00503      *
00504      * @par Lua
00505      * 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})
00506      *
00507      * \endchinese
00508      *
00509      * \english
00510      * Calculate axis-angle difference between two poses
00511      *
00512      * @param p1 pose 1
00513      * @param p2 pose 2
00514      * @return axis angle difference
00515      *
00516      * @par Lua
00517      * poseAngleDistance(p1: table, p2: table) -> number
00518      *
00519      * @par Lua
00520      * 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})
00521      *
00522      * \endenglish
00523      */
00524     double poseAngleDistance(const std::vector<double> &p1,
00525
00526                              const std::vector<double> &p2);
00527
00528     /**
00529      * @ingroup Math
00530      * \chinese
00531      *
00532      *
00533      * @param p1   1
00534      * @param p2   2
00535      * @param eps
00536      * @return    true    false
00537      *
00538      * @par Lua
00539      * poseEqual(p1: table, p2: table, eps: number) -> boolean
00540      *
00541      * @par Lua
00542      * 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)
00543      *
00544      * @par JSON-RPC
00545      * {"jsonrpc":"2.0","method":"Math.poseEqual","params":[[0.1, 0.3, 0.1,
00546      * 0.3142, 0.0, 1.571],[0.1, 0.3, 0.1, 0.3142, 0.0, 1.5711],0.01]],"id":1}
00547      *
00548      * @par JSON-RPC
00549      * {"id":1,"jsonrpc":"2.0","result":0.0}
00550      * \endchinese
00551      *
00552      * \english
00553      * Determine if two poses are equivalent
00554      *
00555      * @param p1 pose 1
00556      * @param p2 pose 2
00557      * @param eps error margin
00558      * @return true or false
00559      *
00560      * @par Lua
00561      * poseEqual(p1: table, p2: table, eps: number) -> boolean
00562      *
00563      * @par Lua
00564      * 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)
00565      *
00566      * @par JSON-RPC request example
00567      * {"jsonrpc":"2.0","method":"Math.poseEqual","params":[[0.1, 0.3, 0.1,
00568      * 0.3142, 0.0, 1.571],[0.1, 0.3, 0.1, 0.3142, 0.0, 1.5711],0.01],"id":1}
00569      *
00570      * @par JSON-RPC response example
00571      * {"id":1,"jsonrpc":"2.0","result":0.0}
00572      * \endenglish
00573      */
00574     bool poseEqual(const std::vector<double> &p1, 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 successfull
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-
      21,0.5932768162455785],0]}
00833     * \endchinese
00834     *
```

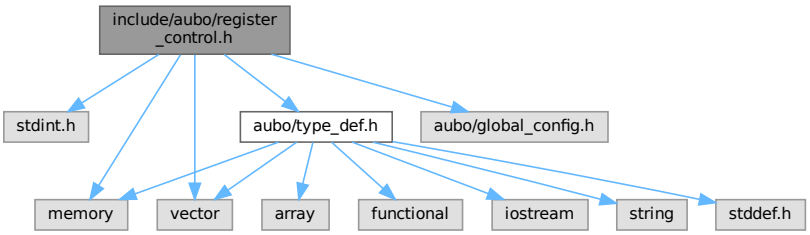
```
00835     * \english
00836     * Calibrate coordinate system with 3 points
00837     *
00838     * @param poses set of 3 poses
00839     * @param type type:\n
00840     *      0 - oxy origin, +x axis, xy plane (+y direction) \n
00841     *      1 - oxz origin, +x axis, xz plane (+z direction) \n
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    /**
00957     * @ingroup Math
00958     * \chinese
00959     * @brief
00960     * @param distances: N    , N >=3
00961     * @param position:
00962     * @param radius:        tcp
00963     * @param track_scale:    ,    (0, 1], 1
00964     * @return
00965     * \endchinese
00966     *
00967     * \english
00968     * @brief Calculate pose increment in tool coordinate system based on sensor data
00969     * @param distances: N distances, N >= 3
00970     * @param position: reference height to maintain from the trajectory
00971     * @param radius: effective radius from sensor center to tool TCP
00972     * @param track_scale: tracking ratio, range (0, 1], 1 means faster tracking
00973     * @return Pose increment in tool coordinate system
00974     * \endenglish
00975     */
00976    std::vector<double> getDeltaPoseBySensorDistance(
00977        const std::vector<double> &distances, double position, double radius,
00978        double track_scale);
00979
00980    /**
00981     * @ingroup Math
00982     * \chinese
00983     * @brief changeFTFrame:
00984     * @param pose_a_in_b: a      b
00985     * @param ft_in_a:    a       a
00986     * @return ft_in_b    b       b
00987     * \endchinese
00988     *
00989     * \english
00990     * @brief changeFTFrame: Transform the reference frame of force and torque
00991     * @param pose_a_in_b: pose of frame a in frame b
00992     * @param ft_in_a: force and torque applied at point a, described in frame a
00993     * @return ft_in_b, force and torque applied at point b, described in frame b
00994     * \endenglish
00995     */
```

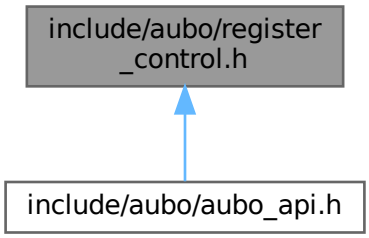
```
00996     std::vector<double> deltaPoseTrans(const std::vector<double> &pose_a_in_b,
00997                                        const std::vector<double> &ft_in_a);
00998
00999     /**
01000      * @ingroup Math
01001      * \chinese
01002      * @brief addDeltaPose:
01003      * @param pose_a_in_b:     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 initiallized.
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 enterd 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 initiallized
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 enterd 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     *
01165     * @param key
01166     * @return
01167     *
01168     * @par Lua
01169     * hasNamedVariable(key: string) -> boolean
01170     *
01171     * @par Lua
01172     * NamedVariable = hasNamedVariable("custom")
01173     *
01174     * @par JSON-RPC
01175     * {"jsonrpc":"2.0","method":"RegisterControl.hasNamedVariable","params":["custom"],"id":1}
01176     *
01177     * @par JSON-RPC
01178     * {"id":1,"jsonrpc":"2.0","result":false}
01179     *
01180     * \endchinese
01181     * \english
01182     * Whether the named variable exists
01183     *
01184     * @param key Variable name
01185     * @return
01186     *
01187     * @par Lua interface prototype
01188     * hasNamedVariable(key: string) -> boolean
01189     *
01190     * @par Lua example
01191     * NamedVariable = hasNamedVariable("custom")
01192     *
01193     * @par JSON-RPC request example
01194     * {"jsonrpc":"2.0","method":"RegisterControl.hasNamedVariable","params":["custom"],"id":1}
01195     *
01196     * @par JSON-RPC response example
01197     * {"id":1,"jsonrpc":"2.0","result":false}
01198     *
01199     * \endenglish
01200     */
01201    bool hasNamedVariable(const std::string &key);
01202
01203    /**
01204     * @ingroup RegisterControl
01205     * \chinese
01206     *
01207     *
01208     * @param key
01209     * @return
01210     *
01211     * @par Lua
01212     * getNamedVariableType(key: string) -> string
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     *   Modbus
02459     *
02460     *              16 Modbus
02461     *          "       "
02462     *
02463     *
02464     *
02465     * @param signal_name
02466     *                 modbusAddSignal  16 Modbus
02467     *     /
02468     * @param count                 1
02469     * -  1
02470     *
02471     * -  >1
02472     *         count
02473     *
02474     * @return    0
02475     *
02476     * @note
02477     * 1.      16 Modbus     signal_type 2/3
02478     * 2.                30
02479     * 3.                   count
02480     *
02481     * @par Python
02482     * addFixedModbusGroup(self: pyaubo_sdk.RegisterControl, arg0: str, arg1:
02483     * int) -> int
02484     *
02485     * @par Lua
02486     * addFixedModbusGroup(signal_name: string, count: number) -> nil
02487     *
02488     * @par Lua
02489     * --     "Modbus_0"   10
02490     * addFixedModbusGroup("Modbus_0", 10)
02491     *
02492     * @par JSON-RPC
02493     * {"jsonrpc":"2.0","method":"RegisterControl.addFixedModbusGroup","params":["Modbus_0",10],"id":1}
02494     *
02495     * @par JSON-RPC
02496     * {"id":1,"jsonrpc":"2.0","result":0}
02497     * \endchinese
02498     * \english
02499     * Create a fixed read group for the specified Modbus register signal to
02500     * optimize the batch reading efficiency of continuous addresses
02501     *
02502     * Function: Locate the address of the corresponding 16-bit Modbus register
02503     * according to the input start signal name, and create a "non-extensible
02504     * fixed group" based on the start address and continuous count. This group
02505     * will be managed independently, and will not be merged or expanded with
02506     * other normal groups. It is suitable for continuous register address
02507     * segments that require stable and atomic reading; After the fixed group is
02508     * created, the read operation for this address segment will be strictly
02509     * executed according to the group range, avoiding read range changes caused
02510     * by dynamic group expansion.
02511     *
02512     * @param signal_name
02513     * Start signal name (used as the reference signal for the fixed group, must
02514     * be a 16-bit Modbus register signal created via modbusAddSignal,
02515     * supporting holding register/input register types, not supporting digital
02516     * signals)
```

```
02517     * @param count Number of continuous signals (registers) contained in the
02518     * fixed group (greater than or equal to 1):
02519     * - Value = 1: Only include the single register address
02520     * corresponding to the start signal (cancel the fixed group containing this
02521     * signal);
02522     * - Value > 1: Continuous count register addresses starting
02523     * from the start signal address, ensure the addresses are continuous and do
02524     * not exceed the device support range;
02525     *
02526     * @return Returns 0 on success, returns an error code on failure.
02527     *
02528     * @note
02529     * 1. This interface only takes effect for 16-bit Modbus register signals
02530     * (signal_type is 2/3);
02531     * 2. The maximum continuous count of a single fixed group is limited by the
02532     * device (default maximum 30);
02533     * 3. Repeated creation of a fixed group for the same start signal will
02534     * overwrite the original configuration (the last count value shall
02535     * prevail).
02536     *
02537     * @par Python interface prototype
02538     * addFixedModbusGroup(self: pyaubo_sdk.RegisterControl, arg0: str, arg1:
02539     * int) -> int
02540     *
02541     * @par Lua interface prototype
02542     * addFixedModbusGroup(signal_name: string, count: number) -> nil
02543     *
02544     * @par Lua example
02545     * -- Create a fixed group containing 10 consecutive registers for the start
02546     * signal "Modbus_0" addFixedModbusGroup("Modbus_0", 10)
02547     *
02548     * @par JSON-RPC request example
02549     * {"jsonrpc":"2.0","method":"RegisterControl.addFixedModbusGroup","params":["Modbus_0",10],"id":1}
02550     *
02551     * @par JSON-RPC response example
02552     * {"id":1,"jsonrpc":"2.0","result":0}
02553     * \endenglish
02554     */
02555    int addFixedModbusGroup(const std::string &signal_name, int count);
02556
02557    /**
02558     * @ingroup RegisterControl
02559     * \chinese
02560     *
02561     *
02562     * @param signal_name
02563     * @return
02564     *
02565     * @par Python
02566     * modbusDeleteSignal(self: pyaubo_sdk.RegisterControl, arg0: str) -> int
02567     *
02568     * @par Lua
02569     * modbusDeleteSignal(signal_name: string) -> nil
02570     *
02571     * @par Lua
02572     * modbusDeleteSignal("Modbus_1")
02573     *
02574     * @par JSON-RPC
02575     * {"jsonrpc":"2.0","method":"RegisterControl.modbusDeleteSignal","params":["Modbus_1"],"id":1}
02576     *
02577     * @par JSON-RPC
02578     * {"id":1,"jsonrpc":"2.0","result":0}
02579     * \endchinese
02580     * \english
02581     * Deletes the signal identified by the supplied signal name.
02582     *
02583     * @param signal_name A string equal to the name of the signal that should
02584     * be deleted.
02585     * @return
02586     *
02587     * @par Python interface prototype
02588     * modbusDeleteSignal(self: pyaubo_sdk.RegisterControl, arg0: str) -> int
02589     *
02590     * @par Lua interface prototype
02591     * modbusDeleteSignal(signal_name: string) -> nil
02592     *
02593     * @par Lua example
02594     * modbusDeleteSignal("Modbus_1")
02595     *
02596     * @par JSON-RPC request example
02597     * {"jsonrpc":"2.0","method":"RegisterControl.modbusDeleteSignal","params":["Modbus_1"],"id":1}
02598     *
02599     * @par JSON-RPC response example
02600     * {"id":1,"jsonrpc":"2.0","result":0}
02601     * \endenglish
02602     */
02603    int modbusDeleteSignal(const std::string &signal_name);
```

```
02604
02605     /**
02606      * @ingroup RegisterControl
02607      * \chinese
02608      *    modbus
02609      *
02610      * @return
02611      *
02612      * @par JSON-RPC
02613      * {"jsonrpc":"2.0","method":"RegisterControl.modbusDeleteAllSignals","params":[],"id":1}
02614      *
02615      * @par JSON-RPC
02616      * {"id":1,"jsonrpc":"2.0","result":0}
02617      *
02618      * \endchinese
02619      * \english
02620      * Delete all modbus signals
02621      *
02622      * @return
02623      *
02624      * @par JSON-RPC request example
02625      * {"jsonrpc":"2.0","method":"RegisterControl.modbusDeleteAllSignals","params":[],"id":1}
02626      *
02627      * @par JSON-RPC response example
02628      * {"id":1,"jsonrpc":"2.0","result":0}
02629      *
02630      * \endenglish
02631      */
02632     int modbusDeleteAllSignals();
02633
02634     /**
02635      * @ingroup RegisterControl
02636      * \chinese
02637      *
02638      *
02639      * @param signal_name
02640      * @return      1 0
02641      *              -1
02642      *
02643      * @par Python
02644      * modbusGetSignalStatus(self: pyaubo_sdk.RegisterControl, arg0: str) -> int
02645      *
02646      * @par Lua
02647      * modbusGetSignalStatus(signal_name: string) -> number
02648      *
02649      * @par Lua
02650      * var0 = modbusGetSignalStatus("Modbus_0")
02651      *
02652      * @par JSON-RPC
02653      * {"jsonrpc":"2.0","method":"RegisterControl.modbusGetSignalStatus","params":["Modbus_0"],"id":1}
02654      *
02655      * @par JSON-RPC
02656      * {"id":1,"jsonrpc":"2.0","result":1}
02657      * \endchinese
02658      * \english
02659      * Reads the current value of a specific signal.
02660      *
02661      * @param signal_name A string equal to the name of the signal for which the
02662      * value should be gotten.
02663      * @return An integer or a boolean. For digital signals: 1 or 0. For
02664      * register signals: The register value expressed as an integer. If the
02665      * value is -1, it means the signal does not exist.
02666      *
02667      * @par Python interface prototype
02668      * modbusGetSignalStatus(self: pyaubo_sdk.RegisterControl, arg0: str) -> int
02669      *
02670      * @par Lua interface prototype
02671      * modbusGetSignalStatus(signal_name: string) -> number
02672      *
02673      * @par Lua example
02674      * var0 = modbusGetSignalStatus("Modbus_0")
02675      *
02676      * @par JSON-RPC request example
02677      * {"jsonrpc":"2.0","method":"RegisterControl.modbusGetSignalStatus","params":["Modbus_0"],"id":1}
02678      *
02679      * @par JSON-RPC response example
02680      * {"id":1,"jsonrpc":"2.0","result":1}
02681      * \endenglish
02682      */
02683     int modbusGetSignalStatus(const std::string &signal_name);
02684
02685     /**
02686      * @ingroup RegisterControl
02687      * \chinese
02688      *
02689      *
02690      * @return
```

```
02691     *
02692     * @par JSON-RPC
02693     * {"jsonrpc":"2.0","method":"RegisterControl.modbusGetSignalNames","params":[],"id":1}
02694     *
02695     * @par JSON-RPC
02696     * {"id":1,"jsonrpc":"2.0","result":["Modbus_0"]}
02697     *
02698     * \endchinese
02699     * \english
02700     * Get the collection of all signal names
02701     *
02702     * @return Collection of all signal names
02703     *
02704     * @par JSON-RPC request example
02705     * {"jsonrpc":"2.0","method":"RegisterControl.modbusGetSignalNames","params":[],"id":1}
02706     *
02707     * @par JSON-RPC response example
02708     * {"id":1,"jsonrpc":"2.0","result":["Modbus_0"]}
02709     *
02710     * \endenglish
02711     */
02712    std::vector<std::string> modbusGetSignalNames();
02713
02714    /**
02715     * @ingroup RegisterControl
02716     * \chinese
02717     *
02718     *
02719     * @return
02720     *
02721     * @par JSON-RPC
02722     * {"jsonrpc":"2.0","method":"RegisterControl.modbusGetSignalTypes","params":[],"id":1}
02723     *
02724     * @par JSON-RPC
02725     * {"id":1,"jsonrpc":"2.0","result":[1,0,2,3]}
02726     *
02727     * \endchinese
02728     * \english
02729     * Get the collection of all signal types
02730     *
02731     * @return Collection of all signal types
02732     *
02733     * @par JSON-RPC request example
02734     * {"jsonrpc":"2.0","method":"RegisterControl.modbusGetSignalTypes","params":[],"id":1}
02735     *
02736     * @par JSON-RPC response example
02737     * {"id":1,"jsonrpc":"2.0","result":[1,0,2,3]}
02738     *
02739     * \endenglish
02740     */
02741    std::vector<int> modbusGetSignalTypes();
02742
02743    /**
02744     * @ingroup RegisterControl
02745     * \chinese
02746     *
02747     *
02748     * @return
02749     *
02750     * @par JSON-RPC
02751     * {"jsonrpc":"2.0","method":"RegisterControl.modbusGetSignalValues","params":[],"id":1}
02752     *
02753     * @par JSON-RPC
02754     * {"id":1,"jsonrpc":"2.0","result":[1,1,88,33]}
02755     *
02756     * \endchinese
02757     * \english
02758     * Get the collection of all signal values
02759     *
02760     * @return Collection of all signal values
02761     *
02762     * @par JSON-RPC request example
02763     * {"jsonrpc":"2.0","method":"RegisterControl.modbusGetSignalValues","params":[],"id":1}
02764     *
02765     * @par JSON-RPC response example
02766     * {"id":1,"jsonrpc":"2.0","result":[1,1,88,33]}
02767     *
02768     * \endenglish
02769     */
02770    std::vector<int> modbusGetSignalValues();
02771
02772    /**
02773     * @ingroup RegisterControl
02774     * \chinese
02775     *          (0:  , :  )
02776     *
02777     * @return ModbusErrorNum
```

```
02778     *
02779     * @par JSON-RPC
02780     * {"jsonrpc":"2.0","method":"RegisterControl.modbusGetSignalErrors","params":[],"id":1}
02781     *
02782     * @par JSON-RPC
02783     * {"id":1,"jsonrpc":"2.0","result":[6,6,6,6]}
02784     *
02785     * \endchinese
02786     * \english
02787     * Get the error status of all signal requests (0: no error, others: error)
02788     * as a collection
02789     *
02790     * @return ModbusErrorNum
02791     *
02792     * @par JSON-RPC request example
02793     * {"jsonrpc":"2.0","method":"RegisterControl.modbusGetSignalErrors","params":[],"id":1}
02794     *
02795     * @par JSON-RPC response example
02796     * {"id":1,"jsonrpc":"2.0","result":[6,6,6,6]}
02797     *
02798     * \endenglish
02799     */
02800    std::vector<int> modbusGetSignalErrors();
02801
02802    /**
02803     * @ingroup RegisterControl
02804     * \chinese
02805     *          IP    Modbus
02806     *
02807     *
02808     *       Modbus
02809     *
02810     * @param device_info
02811     *     RTU     "serial_port,baud,parity,data_bit,stop_bit"
02812     * (1)serial_port          Linux  "/dev/ttyS0" "/dev/ttyUSB0"  Windows  "\.\COM10"
02813     * (2)baud          9600 19200 57600 115200
02814     * (3)parity        N     E     O
02815     * (4)data_bit          5 6 7 8
02816     * (5)stop_bit          1 2
02817     *
02818     *     TCP     "ip address,port"
02819     * (1)ip address      IP
02820     * (2)port
02821     * @param slave_number
02822     * @param function_code
02823     *
02824     * Modbus
02825     * MODBUS_FC_READ_COILS                0x01
02826     * MODBUS_FC_READ_DISCRETE_INPUTS      0x02
02827     * MODBUS_FC_READ_HOLDING_REGISTERS    0x03
02828     * MODBUS_FC_READ_INPUT_REGISTERS      0x04
02829     * MODBUS_FC_WRITE_SINGLE_COIL         0x05
02830     * MODBUS_FC_WRITE_SINGLE_REGISTER     0x06
02831     * MODBUS_FC_READ_EXCEPTION_STATUS     0x07
02832     * MODBUS_FC_WRITE_MULTIPLE_COILS      0x0F
02833     * MODBUS_FC_WRITE_MULTIPLE_REGISTERS  0x10
02834     * MODBUS_FC_REPORT_SLAVE_ID           0x11
02835     * MODBUS_FC_MASK_WRITE_REGISTER       0x16
02836     * MODBUS_FC_WRITE_AND_READ_REGISTERS  0x17
02837     *
02838     * @param data        0-255
02839     * @return
02840     *
02841     * @par Python
02842     * modbusSendCustomCommand(self: pyaubo_sdk.RegisterControl, arg0: str,
02843     * arg1: int, arg2: int, arg3: List[int]) -> int
02844     *
02845     * @par Lua
02846     * modbusSendCustomCommand(device_info: string, slave_number: number,
02847     * function_code: number, data: table) -> nil
02848     *
02849     * @par Lua
02850     * modbusSendCustomCommand("/dev/ttyRobotTool,115200,N,8,1", 1, 10,
02851     * {1,2,0,2,4,0,0,0,0}) -> nil
02852     *
02853     * @par JSON-RPC
02854     * {"json-
rpc":"2.0","method":"RegisterControl.modbusSendCustomCommand","params":["/dev/ttyRobotTool,115200,N,8,1",1,10,[1,2,0,2,4,0,0,0,0]],"id"
02855     *
02856     * @par JSON-RPC
02857     * {"id":1,"jsonrpc":"2.0","result":0}
02858     * \endchinese
02859     * \english
02860     * Sends a command specified by the user to the modbus unit located on the
02861     * specified IP address. Cannot be used to request data, since the response
02862     * will not be received. The user is responsible for supplying data which
02863     * is meaningful to the supplied function code. The builtin function takes
```

```
02864     * care of constructing the modbus frame, so the user should not be
02865     * concerned with the length of the command.
02866     *
02867     * @param device_info is rtu format.
02868     * eg,"serial_port,baud,parity,data_bit,stop_bit"
02869     * (1)The serial_port argument specifies the name of the serial port eg. On
02870     * Linux ,"/dev/ttyS0" or "/dev/ttyUSB0". On Windows, "\.\COM10".
02871     * (2)The baud argument specifies the baud rate of the communication, eg.
02872     * 9600, 19200, 57600, 115200, etc.
02873     * (3)parity:N for none,E for even,O for odd.
02874     * (4)data_bit:The data_bits argument specifies the number of bits of data,
02875     * the allowed values are 5, 6, 7 and 8.
02876     * (5)stop_bit:The stop_bits argument
02877     * specifies the bits of stop, the allowed values are 1 and 2.
02878     *
02879     * device_info is tcp format.eg,"ip address,port"
02880     * (1)The ip address parameter specifies the ip address of the server
02881     * (2)The port parameter specifies the port number that the server is
02882     * listening on.
02883     * @param slave_number An integer specifying the slave number to use for
02884     * the custom command.
02885     * @param function_code An integer specifying the function code for the
02886     * custom command.
02887     *
02888     * Modbus function codes
02889     * MODBUS_FC_READ_COILS                0x01
02890     * MODBUS_FC_READ_DISCRETE_INPUTS      0x02
02891     * MODBUS_FC_READ_HOLDING_REGISTERS    0x03
02892     * MODBUS_FC_READ_INPUT_REGISTERS      0x04
02893     * MODBUS_FC_WRITE_SINGLE_COIL         0x05
02894     * MODBUS_FC_WRITE_SINGLE_REGISTER     0x06
02895     * MODBUS_FC_READ_EXCEPTION_STATUS     0x07
02896     * MODBUS_FC_WRITE_MULTIPLE_COILS      0x0F
02897     * MODBUS_FC_WRITE_MULTIPLE_REGISTERS  0x10
02898     * MODBUS_FC_REPORT_SLAVE_ID           0x11
02899     * MODBUS_FC_MASK_WRITE_REGISTER       0x16
02900     * MODBUS_FC_WRITE_AND_READ_REGISTERS  0x17
02901     *
02902     * @param data An array of integers in which each entry must be a valid
02903     * byte (0-255) value.
02904     * @return
02905     *
02906     * @par Python interface prototype
02907     * modbusSendCustomCommand(self: pyaubo_sdk.RegisterControl, arg0: str,
02908     * arg1: int, arg2: int, arg3: List[int]) -> int
02909     *
02910     * @par Lua interface prototype
02911     * modbusSendCustomCommand(device_info: string, slave_number: number,
02912     * function_code: number, data: table) -> nil
02913     *
02914     * @par Lua example
02915     * modbusSendCustomCommand("/dev/ttyRobotTool,115200,N,8,1", 1, 10,
02916     * {1,2,0,2,4,0,0,0,0}) -> nil
02917     *
02918     * @par JSON-RPC request example
02919     * {"json-
rpc":"2.0","method":"RegisterControl.modbusSendCustomCommand","params":["/dev/ttyRobotTool,115200,N,8,1",1,10,[1,2,0,2,4,0,0,0,0]],"id"
02920     *
02921     * @par JSON-RPC response example
02922     * {"id":1,"jsonrpc":"2.0","result":0}
02923     * \endenglish
02924     */
02925    int modbusSendCustomCommand(const std::string &device_info,
02926                                int slave_number, int function_code,
02927                                const std::vector<uint8_t> &data);
02928
02929    /**
02930     * @ingroup RegisterControl
02931     * \chinese
02932     *          "default" "freedrive"
02933     *
02934     * @param robot_name
02935     * @param signal_name
02936     * @param action        "default" "freedrive"
02937     * @return
02938     *
02939     * @par Python
02940     * modbusSetDigitalInputAction(self: pyaubo_sdk.RegisterControl, arg0: str,
02941     * arg1: str, arg2: int)
02942     *
02943     * @par Lua
02944     * modbusSetDigitalInputAction(robot_name: string, signal_name: string,
02945     * action: number) -> nil
02946     *
02947     * @par Lua
02948     * modbusSetDigitalInputAction("rob1", "Modbus_0","Handguide")
02949     *
```

```
02950     * @par JSON-RPC
02951     * {"json-
rpc":"2.0","method":"RegisterControl.modbusSetDigitalInputAction","params":["rob1","Modbus_0","Handguide"],"id":1}
02952     *
02953     * @par JSON-RPC
02954     * {"id":1,"jsonrpc":"2.0","result":0}
02955     * \endchinese
02956     * \english
02957     * Sets the selected digital input signal to either a "default" or
02958     * "freedrive" action.
02959     *
02960     * @param robot_name A string identifying a robot name that connected robot
02961     * @param signal_name A string identifying a digital input signal that was
02962     * previously added.
02963     * @param action The type of action. The action can either be "default" or
02964     * "freedrive". (string)
02965     * @return
02966     *
02967     * @par Python interface prototype
02968     * modbusSetDigitalInputAction(self: pyaubo_sdk.RegisterControl, arg0: str,
02969     * arg1: str, arg2: int)
02970     *
02971     * @par Lua interface prototype
02972     * modbusSetDigitalInputAction(robot_name: string, signal_name: string,
02973     * action: number) -> nil
02974     *
02975     * @par Lua example
02976     * modbusSetDigitalInputAction("rob1", "Modbus_0","Handguide")
02977     *
02978     * @par JSON-RPC request example
02979     * {"json-
rpc":"2.0","method":"RegisterControl.modbusSetDigitalInputAction","params":["rob1","Modbus_0","Handguide"],"id":1}
02980     *
02981     * @par JSON-RPC response example
02982     * {"id":1,"jsonrpc":"2.0","result":0}
02983     * \endenglish
02984     */
02985    int modbusSetDigitalInputAction(const std::string &robot_name,
02986                                    const std::string &signal_name,
02987                                    StandardInputAction action);
02988
02989    /**
02990     * @ingroup RegisterControl
02991     * \chinese
02992     *   Modbus
02993     *
02994     * @param robot_name
02995     * @param signal_name
02996     * @param runstate
02997     * @return
02998     *
02999     * @par JSON-RPC
03000     *
{"jsonrpc":"2.0","method":"RegisterControl.modbusSetOutputRunstate","params":["rob1","Modbus_0","None"],"id":1}
03001     *
03002     * @par JSON-RPC
03003     * {"id":1,"jsonrpc":"2.0","result":0}
03004     *
03005     * \endchinese
03006     * \english
03007     * Set Modbus signal output action
03008     *
03009     * @param robot_name
03010     * @param signal_name
03011     * @param runstate
03012     * @return
03013     *
03014     * @par JSON-RPC request example
03015     *
{"jsonrpc":"2.0","method":"RegisterControl.modbusSetOutputRunstate","params":["rob1","Modbus_0","None"],"id":1}
03016     *
03017     * @par JSON-RPC response example
03018     * {"id":1,"jsonrpc":"2.0","result":0}
03019     *
03020     * \endenglish
03021     */
03022    int modbusSetOutputRunstate(const std::string &robot_name,
03023                                const std::string &signal_name,
03024                                StandardOutputRunState runstate);
03025
03026    /**
03027     * @ingroup RegisterControl
03028     * \chinese
03029     *   Modbus          / /
03030     *
03031     *                 /      Modbus
03032     *                Modbus
```

```
03033     *
03034     * @param robot_name             /
03035     * @param signal_name
03036     *     Modbus          modbusAddSignal
03037     * @param runstate
03038     * @param object_name
03039     * @param threshold
03040     * -              (rad)
03041     * -    /           (m)
03042     *
03043     * @return
03044     * - 0:
03045     * -  0:
03046     *
03047     * @par Python
03048     * modbusSetOutputRunstate1(self: pyaubo_sdk.RegisterControl, arg0: str,
03049     * arg1: str, arg2: str, arg3: str, arg4: float) -> int
03050     *
03051     * @par Lua
03052     * modbusSetOutputRunstate1(robot_name: string, signal_name: string,
03053     * runstate: string, object_name: string, threshold: number) -> nil
03054     *
03055     * @par JSON-RPC
03056     * {"json-
rpc":"2.0","method":"RegisterControl.modbusSetOutputRunstate1","params":["rob1","Modbus_0","WaypointArrived","P1",3],"id":1}
03057     *
03058     * @par JSON-RPC
03059     * {"id":1,"jsonrpc":"2.0","result":0}
03060     *
03061     * @note
03062     * 1.    object_name   object_type
03063     * 2. threshold      object_type
03064     *
03065     * \endchinese
03066     * \english
03067     * Set Modbus signal output action with optional condition checking based on
03068     * waypoint, plane or obstacle thresholds.
03069     *
03070     * Detailed description: Use this API to control a Modbus output signal when
03071     * a specified robot condition is met (for example, waypoint arrival or
03072     * distance to a plane/obstacle). Specify an object type, name and a
03073     * threshold value to enable conditional triggering. When the condition is
03074     * satisfied, the given Modbus signal will be set to the requested
03075     * runstate.
03076     *
03077     * @param robot_name The target robot name identifying which robot's
03078     *                   conditions to monitor.
03079     * @param signal_name The Modbus signal name to set (must be previously
03080     *                    added via modbusAddSignal).
03081     * @param runstate The desired output action.
03082     * @param object_name Unique identifier of the condition object
03083     * @param threshold Condition threshold: angle in radians for waypoints, or
03084     *                  distance in meters for planes/obstacles
03085     *
03086     * @return Returns 0 on success, non-zero error code on failure.
03087     *
03088     * @par JSON-RPC request example
03089     * {"json-
rpc":"2.0","method":"RegisterControl.modbusSetOutputRunstate1","params":["rob1","Modbus_0","WaypointArrived","P1",3],"id":1}
03090     *
03091     * @par JSON-RPC response example
03092     * {"id":1,"jsonrpc":"2.0","result":0}
03093     * \endenglish
03094     */
03095    int modbusSetOutputRunstate1(const std::string &robot_name,
03096                                 const std::string &signal_name,
03097                                 StandardOutputRunState runstate,
03098                                 const std::string &object_name,
03099                                 double threshold);
03100
03101    /**
03102     * @ingroup RegisterControl
03103     * \chinese
03104     *
03105     *
03106     * @param signal_name
03107     * @param value          0-65535
03108     * @return
03109     *
03110     * @par Python
03111     * modbusSetOutputSignal(self: pyaubo_sdk.RegisterControl, arg0: str, arg1:
03112     * int) -> int
03113     *
03114     * @par Lua
03115     * modbusSetOutputSignal(signal_name: string, value: number) -> nil
03116     *
03117     * @par Lua
```

```
03118     * modbusSetOutputSignal("Modbus_0",0)
03119     *
03120     * @par JSON-RPC
03121     * {"jsonrpc":"2.0","method":"RegisterControl.modbusSetOutputSignal","params":["Modbus_0",0],"id":1}
03122     *
03123     * @par JSON-RPC
03124     * {"id":1,"jsonrpc":"2.0","result":0}
03125     * \endchinese
03126     * \english
03127     * Sets the output register signal identified by the given name to the given
03128     * value.
03129     *
03130     * @param signal_name A string identifying an output register signal that in
03131     * advance has been added.
03132     * @param value An integer which must be a valid word (0-65535)
03133     * @return
03134     *
03135     * @par Python interface prototype
03136     * modbusSetOutputSignal(self: pyaubo_sdk.RegisterControl, arg0: str, arg1:
03137     * int) -> int
03138     *
03139     * @par Lua interface prototype
03140     * modbusSetOutputSignal(signal_name: string, value: number) -> nil
03141     *
03142     * @par Lua example
03143     * modbusSetOutputSignal("Modbus_0",0)
03144     *
03145     * @par JSON-RPC request example
03146     * {"jsonrpc":"2.0","method":"RegisterControl.modbusSetOutputSignal","params":["Modbus_0",0],"id":1}
03147     *
03148     * @par JSON-RPC response example
03149     * {"id":1,"jsonrpc":"2.0","result":0}
03150     * \endenglish
03151     */
03152    int modbusSetOutputSignal(const std::string &signal_name, uint16_t value);
03153
03154    /**
03155     * @ingroup RegisterControl
03156     * \chinese
03157     *
03158     *
03159     * @param signal_name
03160     * @param values          0-65535
03161     * @return
03162     *
03163     * @par Python
03164     * modbusSetOutputSignal1(self: pyaubo_sdk.RegisterControl, arg0: str, arg1:
03165     * list[int]) -> int
03166     *
03167     * @par Lua
03168     * modbusSetOutputSignal1(signal_name: string, value: table) -> nil
03169     *
03170     * @par Lua
03171     * modbusSetOutputSignal1("Modbus_0", {0, 1, 2})
03172     *
03173     * @par JSON-RPC
03174     * {"jsonrpc":"2.0","method":"RegisterControl.modbusSetOutputSignal1","params":["Modbus_0",[0,1,2]],"id":1}
03175     *
03176     * @par JSON-RPC
03177     * {"id":1,"jsonrpc":"2.0","result":0}
03178     * \endchinese
03179     * \english
03180     * Sets multiple consecutive output register signals starting from the given
03181     * name to the given values.
03182     *
03183     * @param signal_name A string identifying the starting output register
03184     * signal that in advance has been added.
03185     * @param values An array of integers where each element must be a valid
03186     * word (0-65535), corresponding to multiple consecutive signals
03187     * @return
03188     *
03189     * @par Python interface prototype
03190     * modbusSetOutputSignal1(self: pyaubo_sdk.RegisterControl, arg0: str, arg1:
03191     * list[int]) -> int
03192     *
03193     * @par Lua interface prototype
03194     * modbusSetOutputSignal1(signal_name: string, value: table) -> nil
03195     *
03196     * @par Lua example
03197     * modbusSetOutputSignal1("Modbus_0", {0, 1, 2})
03198     *
03199     * @par JSON-RPC request example
03200     * {"jsonrpc":"2.0","method":"RegisterControl.modbusSetOutputSignal1","params":["Modbus_0",[0,1,2]],"id":1}
03201     *
03202     * @par JSON-RPC response example
03203     * {"id":1,"jsonrpc":"2.0","result":0}
03204     * \endenglish
```

```
03205     */
03206     int modbusSetOutputSignal1(const std::string &signal_name,
03207                                const std::vector<uint16_t> &values);
03208
03209     /**
03210      * \chinese
03211      *
03212      *
03213      * @param signal_name
03214      * @param value          0-65535
03215      * @param timeout
03216      * @return
03217      *
03218      * @par Python
03219      * modbusSetOutputSignalWithTimeout(self: pyaubo_sdk.RegisterControl, arg0:
03220      * str, arg1: int, arg2: double) -> int
03221      *
03222      * @par Lua
03223      * modbusSetOutputSignalWithTimeout(signal_name: string, value: number,
03224      * timeout: number) -> nil
03225      *
03226      * @par Lua
03227      * modbusSetOutputSignalWithTimeout("Modbus_0",0,0.5)
03228      *
03229      * @par JSON-RPC
03230      *
     {"jsonrpc":"2.0","method":"RegisterControl.modbusSetOutputSignalWithTimeout","params":["Modbus_0",0,0.5],"id":1}
03231      *
03232      * @par JSON-RPC
03233      * {"id":1,"jsonrpc":"2.0","result":0}
03234      * \endchinese
03235      * \english
03236      * Sets the output register signal identified by the given name to the
03237      * given, with timeout value.
03238      *
03239      * @param signal_name A string identifying an output register signal that in
03240      * advance has been added.
03241      * @param value An integer which must be a valid word (0-65535)
03242      * @param timeout seconds
03243      * @return
03244      *
03245      * @par Python interface prototype
03246      * modbusSetOutputSignalWithTimeout(self: pyaubo_sdk.RegisterControl, arg0:
03247      * str, arg1: int, arg2: timeout) -> int
03248      *
03249      * @par Lua interface prototype
03250      * modbusSetOutputSignalWithTimeout(signal_name: string, value: number,
03251      * timeout: number) -> nil
03252      *
03253      * @par Lua example
03254      * modbusSetOutputSignalWithTimeout("Modbus_0",0,0.5)
03255      *
03256      * @par JSON-RPC request example
03257      *
     {"jsonrpc":"2.0","method":"RegisterControl.modbusSetOutputSignalWithTimeout","params":["Modbus_0",0,0.5],"id":1}
03258      *
03259      * @par JSON-RPC response example
03260      * {"id":1,"jsonrpc":"2.0","result":0}
03261      * \endenglish
03262      */
03263     int modbusSetOutputSignalWithTimeout(const std::string &signal_name,
03264                                          uint16_t value, double timeout);
03265
03266     /**
03267      * @ingroup RegisterControl
03268      * \chinese
03269      *  modbus    (      )
03270      *
03271      * @param signal_name:
03272      * @param value:          0-65535
03273      * @param duration:
03274      * @return
03275      *
03276      * @par Python
03277      * modbusSetOutputSignalPulse(self: pyaubo_sdk.RegisterControl, arg0: str,
03278      * arg1: int, arg2 double) -> int
03279      *
03280      * @par Lua
03281      * modbusSetOutputSignalPulse(signal_name: string, value: number, duration:
03282      * number) -> nil
03283      *
03284      * @par Lua
03285      * modbusSetOutputSignalPulse("Modbus_0",1,0.5)
03286      *
03287      * @par JSON-RPC
03288      * {"jsonrpc":"2.0","method":"RegisterControl.modbusSetOutputSignalPulse","params":["Modbus_0",1,0.5],"id":1}
03289      *
```

```
03290     * @par JSON-RPC
03291     * {”id”:1,”jsonrpc”:”2.0”,”result”:0}
03292     *
03293     * \endchinese
03294     * \english
03295     * Set modbus signal output pulse (only supports coil output type)
03296     *
03297     * @param signal_name: A string identifying an output register signal that
03298     * has been added in advance.
03299     * @param value: An integer which must be a valid word (0-65535)
03300     * @param duration: Duration of the signal, in seconds
03301     * @return
03302     *
03303     * @par Python interface prototype
03304     * modbusSetOutputSignalPulse(self: pyaubo_sdk.RegisterControl, arg0: str,
03305     * arg1: int, arg2: double) -> int
03306     *
03307     * @par Lua interface prototype
03308     * modbusSetOutputSignalPulse(signal_name: string, value: number, duration:
03309     * number) -> nil
03310     *
03311     * @par Lua example
03312     * modbusSetOutputSignalPulse(”Modbus_0”,1,0.5)
03313     *
03314     * @par JSON-RPC request example
03315     * {”jsonrpc”:”2.0”,”method”:”RegisterControl.modbusSetOutputSignalPulse”,”params”:[”Modbus_0”,1,0.5],”id”:1}
03316     *
03317     * @par JSON-RPC response example
03318     * {”id”:1,”jsonrpc”:”2.0”,”result”:0}
03319     *
03320     * \endenglish
03321     */
03322    int modbusSetOutputSignalPulse(const std::string &signal_name,
03323                                   uint16_t value, double duration);
03324
03325    /**
03326     * @ingroup RegisterControl
03327     * \chinese
03328     *     Modbus
03329     *
03330     * @param signal_name
03331     * @param update_frequency              0-125
03332     * @return
03333     *
03334     * @par Python
03335     * modbusSetSignalUpdateFrequency(self: pyaubo_sdk.RegisterControl, arg0:
03336     * str, arg1: int) -> int
03337     *
03338     * @par Lua
03339     * modbusSetSignalUpdateFrequency(signal_name: string, update_frequency:
03340     * number) -> nil
03341     *
03342     * @par Lua
03343     * modbusSetSignalUpdateFrequency(”Modbus_0”,1)
03344     *
03345     * @par JSON-RPC
03346     * {”jsonrpc”:”2.0”,”method”:”RegisterControl.modbusSetSignalUpdateFrequency”,”params”:[”Modbus_0”,1],”id”:1}
03347     *
03348     * @par JSON-RPC
03349     * {”id”:1,”jsonrpc”:”2.0”,”result”:0}
03350     * \endchinese
03351     * \english
03352     * Sets the frequency with which the robot will send requests to the Modbus
03353     * controller to either read or write the signal value.
03354     *
03355     * @param signal_name A string identifying an output digital signal that
03356     * in advance has been added.
03357     * @param update_frequency An integer in the range 0-125 specifying the
03358     * update frequency in Hz.
03359     * @return
03360     *
03361     * @par Python interface prototype
03362     * modbusSetSignalUpdateFrequency(self: pyaubo_sdk.RegisterControl, arg0:
03363     * str, arg1: int) -> int
03364     *
03365     * @par Lua interface prototype
03366     * modbusSetSignalUpdateFrequency(signal_name: string, update_frequency:
03367     * number) -> nil
03368     *
03369     * @par Lua example
03370     * modbusSetSignalUpdateFrequency(”Modbus_0”,1)
03371     *
03372     * @par JSON-RPC request example
03373     * {”jsonrpc”:”2.0”,”method”:”RegisterControl.modbusSetSignalUpdateFrequency”,”params”:[”Modbus_0”,1],”id”:1}
03374     *
03375     * @par JSON-RPC response example
03376     * {”id”:1,”jsonrpc”:”2.0”,”result”:0}
```

```
03377     * \endenglish
03378     */
03379    int modbusSetSignalUpdateFrequency(const std::string &signal_name,
03380                                       int update_frequency);
03381
03382    /**
03383     * @ingroup RegisterControl
03384     * \chinese
03385     *     modbus      0       -1
03386     *
03387     * @param signal_name
03388     * @return
03389     *
03390     * @par JSON-RPC
03391     * {"jsonrpc":"2.0","method":"RegisterControl.modbusGetSignalIndex","params":["Modbus_0"],"id":1}
03392     *
03393     * @par JSON-RPC
03394     * {"id":1,"jsonrpc":"2.0","result":0}
03395     *
03396     * \endchinese
03397     * \english
03398     * Get the index of the specified modbus signal, starting from 0. Returns -1
03399     * if it does not exist.
03400     *
03401     * @param signal_name
03402     * @return
03403     *
03404     * @par JSON-RPC request example
03405     * {"jsonrpc":"2.0","method":"RegisterControl.modbusGetSignalIndex","params":["Modbus_0"],"id":1}
03406     *
03407     * @par JSON-RPC response example
03408     * {"id":1,"jsonrpc":"2.0","result":0}
03409     *
03410     * \endenglish
03411     */
03412    int modbusGetSignalIndex(const std::string &signal_name);
03413
03414    /**
03415     * @ingroup RegisterControl
03416     * \chinese
03417     *     modbus
03418     *
03419     * @param signal_name
03420     * @return       ModbusErrorNum
03421     *
03422     * @par JSON-RPC
03423     * {"jsonrpc":"2.0","method":"RegisterControl.modbusGetSignalError","params":["Modbus_0"],"id":1}
03424     *
03425     * @par JSON-RPC
03426     * {"id":1,"jsonrpc":"2.0","result":6}
03427     * \endchinese
03428     * \english
03429     * Get the error status of the specified modbus signal
03430     *
03431     * @param signal_name
03432     * @return Returns error code ModbusErrorNum
03433     *
03434     * @par JSON-RPC request example
03435     * {"jsonrpc":"2.0","method":"RegisterControl.modbusGetSignalError","params":["Modbus_0"],"id":1}
03436     *
03437     * @par JSON-RPC response example
03438     * {"id":1,"jsonrpc":"2.0","result":6}
03439     * \endenglish
03440     */
03441    int modbusGetSignalError(const std::string &signal_name);
03442
03443    /**
03444     * @ingroup RegisterControl
03445     * \chinese
03446     *     modbus
03447     *
03448     * @param device_name
03449     *    TCP   "ip:port",    "127.0.0.1:502" \n
03450     *    RTU   "serial_port",    "/dev/ttyUSB0" \n
03451     *
03452     * @return
03453     * 0:
03454     * -1:
03455     * -2:
03456     *
03457     * @par JSON-RPC
03458     * {"jsonrpc":"2.0","method":"RegisterControl.getModbusDeviceStatus","params":["172.16.26.248:502"],"id":1}
03459     *
03460     * @par JSON-RPC
03461     * {"id":1,"jsonrpc":"2.0","result":0}
03462     *
03463     * \endchinese
```

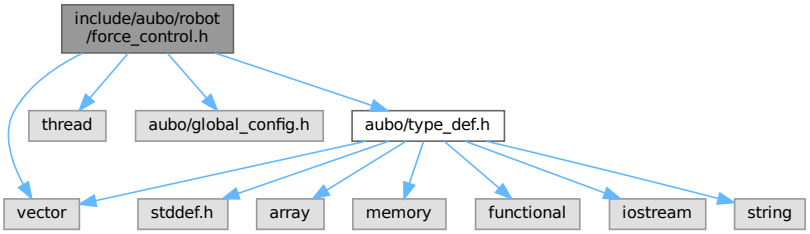
```
03464     * \english
03465     * Get the connection status of the specified modbus device
03466     *
03467     * @param device_name
03468     * Device name in TCP format: "ip:port", e.g. "127.0.0.1:502" \n
03469     * Device name in RTU format: "serial_port", e.g. "/dev/ttyUSB0" \n
03470     *
03471     * @return
03472     * 0: Device is connected
03473     * -1: Device does not exist
03474     * -2: Device is disconnected
03475     *
03476     * @par JSON-RPC request example
03477     * {"jsonrpc":"2.0","method":"RegisterControl.getModbusDeviceStatus","params":["172.16.26.248:502"],"id":1}
03478     *
03479     * @par JSON-RPC response example
03480     * {"id":1,"jsonrpc":"2.0","result":0}
03481     *
03482     * \endenglish
03483     */
03484    int getModbusDeviceStatus(const std::string &device_name);
03485
03486    /**
03487     * @ingroup RegisterControl
03488     * \chinese
03489     *    modbus
03490     *
03491     * @param encoder_id    0
03492     * @param signal_name modbus
03493     * @return
03494     * \endchinese
03495     * \english
03496     * Use a modbus register signal as an encoder
03497     *
03498     * @param encoder_id Must not be 0
03499     * @param signal_name Name of the modbus signal, must be of register type
03500     * @return
03501     * \endenglish
03502     */
03503    int addModbusEncoder(int encoder_id, int range_id,
03504                         const std::string &signal_name);
03505
03506    /**
03507     * @ingroup RegisterControl
03508     * \chinese
03509     *   Int32
03510     *
03511     * @param encoder_id    ID
03512     * @param range_id   ID
03513     * @param key
03514     * @return
03515     * \endchinese
03516     * \english
03517     * Add a virtual encoder for an Int32 register
03518     *
03519     * @param encoder_id Encoder ID
03520     * @param range_id Range ID
03521     * @param key Variable name
03522     * @return
03523     * \endenglish
03524     */
03525    int addInt32RegEncoder(int encoder_id, int range_id,
03526                           const std::string &key);
03527
03528    /**
03529     * @ingroup RegisterControl
03530     * \chinese
03531     *
03532     *
03533     * @param encoder_id
03534     * @return
03535     * \endchinese
03536     * \english
03537     * Delete virtual encoder
03538     *
03539     * @param encoder_id
03540     * @return
03541     * \endenglish
03542     */
03543    int deleteVirtualEncoder(int encoder_id);
03544
03545 protected:
03546    void *d_;
03547 };
03548 using RegisterControlPtr = std::shared_ptr<RegisterControl>;
03549
03550 } // namespace common_interface
```

```
03551 } // namespace arcs
03552
03553 #endif // AUBO_SDK_REGISTER_CONTROL_INTERFACE_H
```

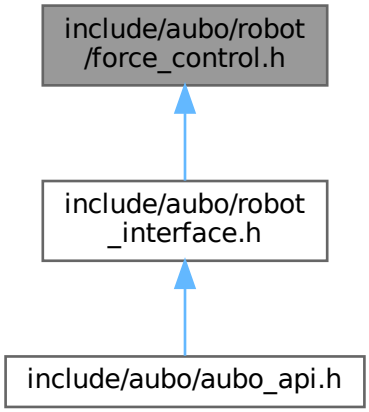
12.20 include/aubo/robot/force_control.h File Reference

```
#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 FCConditionWaitWhile    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, FCConditionWaitWhile 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,
          1], default 0
00419     * @param damp_scale: Parameter representing damping level, range [0, 1], default 0.5
00420     * @param stiff_scale: Parameter representing stiffness level, range [0, 1], default 0.5
00421     *
00422     * @return Return 0 if succeeded; return error code if failed
00423     * AUBO_BUSY
00424     * AUBO_BAD_STATE
00425     * -AUBO_INVL_ARGUMENT
00426     * -AUBO_BAD_STATE
00427     *
```

```
00428     * @throws arcs::common_interface::AuboException
00429     *
00430     * @par Python Function Prototype
00431     * setDynamicModel1(self: pyaubo_sdk.ForceControl, arg0: List[float],
00432     * arg1: List[float], arg2: List[float]) -> int
00433     *
00434     * @par Lua Function Prototype
00435     * setDynamicModel1(env_stiff: table, damp_scale: table, stiff_scale:
00436     * table) -> nil
00437     *
00438     * @par Lua example
00439     * 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})
00440     *
00441     * \endenglish
00442     */
00443    int setDynamicModel1(const std::vector<double> &env_stiff,
00444                         const std::vector<double> &damp_scale,
00445                         const std::vector<double> &stiff_scale);
00446
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},{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,
          1], default 0
00479     * @param damp_scale: Parameter representing damping level, range [0, 1], default 0.5
00480     * @param stiff_scale: Parameter representing stiffness level, range [0, 1], default 0.5
00481     *
00482     * @return Force control dynamics model MDK
00483     * AUBO_BUSY
00484     * AUBO_BAD_STATE
00485     * -AUBO_INVL_ARGUMENT
00486     * -AUBO_BAD_STATE
00487     *
00488     * @throws arcs::common_interface::AuboException
00489     *
00490     * @par Python Function Prototype
00491     * fcCalDynamicModel(self: pyaubo_sdk.ForceControl, arg0: List[float],
00492     * arg1: List[float], arg2: List[float]) -> Tuple[List[float], List[float], List[float]]
00493     *
00494     * @par Lua Function Prototype
00495     * fcCalDynamicModel(env_stiff: table, damp_scale: table, stiff_scale: table) -> table
00496     *
00497     * @par Lua example
00498     * 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})
00499     *
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     * @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,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 lateractivated 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     * @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              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
```

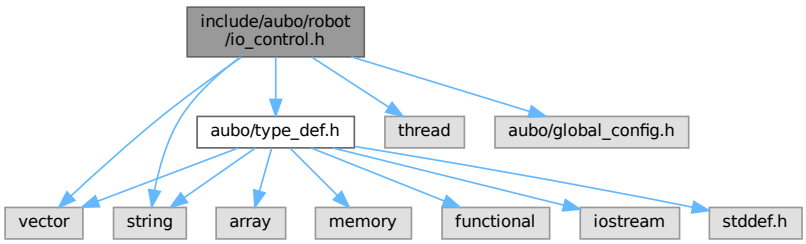
```
02504     */
02505     int softFloatDisable();
02506
02507     /**
02508      * @ingroup ForceControl
02509      * \chinese
02510      *
02511      *
02512      * @return
02513      *
02514      * @throws arcs::common_interface::AuboException
02515      *
02516      * @par JSON-RPC
02517      * {”jsonrpc”:”2.0”,”method”:”rob1.ForceControl.isSoftFloatEnabled”,”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
       contact is detected.
02586      * \endenglish
02587      */
02588     int toolContact(const std::vector<bool> &direction);
02589
```

```
02590     /**
02591      * @ingroup ForceControl
02592      * \chinese
02593      * @brief
02594      *
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 obtain the joint position data at that time.
02606      *
02607      * @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
```

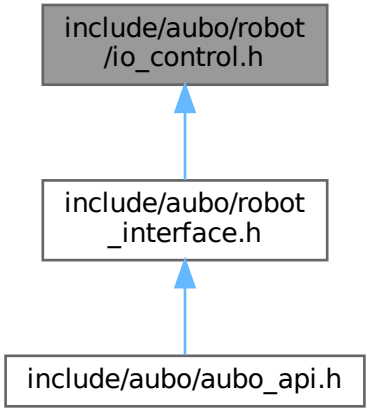
12.22 include/aubo/robot/io_control.h File Reference

IO

#include <vector>
#include <thread>
#include <string>
#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 #include <string>
00010
00011 #include <aubo/global_config.h>
00012 #include <aubo/type_def.h>
00013
00014 namespace arcs {
00015 namespace common_interface {
00016
00017 /**
00018  * @defgroup IoControl IoControl (IO     )
00019  * @ingroup RobotInterface
00020  * \chinese
00021  * IoControl                  IO
00022  *
00023  * 1.    IO    \n
00024  * 2.  IO       \n
00025  * 3.   IO    \n
00026  * 4.  IO
00027  *
00028  *          IO     IO \n
00029  *                 IO \n
00030  *            IO     IO \n
00031  * \endchinese
00032  * \english
00033  * The IoControl class provides a series of interfaces for configuring and
00034  * reading the robot's standard digital and analog IO, as well as setting output
00035  * states.
00036  *
00037  * 1. Get the number of various IOs \n
00038  * 2. Configure IO input/output functions \n
00039  * 3. Configuration of configurable IOs \n
00040  * 4. Set and read the input/output range of analog IOs
00041  *
00042  * Standard digital input/output: Standard IOs on the control cabinet IO panel
00043  * \n Tool digital input/output: Digital IOs exposed via the tool-end connector
00044  * \n Configurable input/output: Can be configured as safety IO or general
00045  * digital IO \n \endenglish
00046  */
00047 class ARCS_ABI_EXPORT IoControl
00048 {
00049 public:
00050     IoControl();
00051     virtual ~IoControl();
00052
00053     /**
00054      * @ingroup IoControl
00055      * \chinese
00056      *
00057      *
00058      * @return
00059      *
00060      * @throws arcs::common_interface::AuboException
00061      *
00062      * @par Python
00063      * getStandardDigitalInputNum(self: pyaubo_sdk.IoControl) -> int
00064      *
00065      * @par Lua
00066      * getStandardDigitalInputNum() -> number
00067      *
00068      * @par Lua
00069      * num = getStandardDigitalInputNum()
00070      *
00071      * @par JSON-RPC
00072      * {"jsonrpc":"2.0","method":"rob1.IoControl.getStandardDigitalInputNum","params":[],"id":1}
00073      *
00074      * @par JSON-RPC
00075      * {"id":1,"jsonrpc":"2.0","result":16}
00076      *
00077      * \endchinese
00078      * \english
00079      * Get the number of standard digital inputs.
00080      *
00081      * @return Number of standard digital inputs.
00082      *
```

```
00083     * @throws arcs::common_interface::AuboException
00084     *
00085     * @par Python function prototype
00086     * getStandardDigitalInputNum(self: pyaubo_sdk.IoControl) -> int
00087     *
00088     * @par Lua function prototype
00089     * getStandardDigitalInputNum() -> number
00090     *
00091     * @par Lua example
00092     * num = getStandardDigitalInputNum()
00093     *
00094     * @par JSON-RPC request example
00095     * {"jsonrpc":"2.0","method":"rob1.IoControl.getStandardDigitalInputNum","params":[],"id":1}
00096     *
00097     * @par JSON-RPC response example
00098     * {"id":1,"jsonrpc":"2.0","result":16}
00099     *
00100     * \endenglish
00101     */
00102    int getStandardDigitalInputNum();
00103
00104    /**
00105     * @ingroup IoControl
00106     * \chinese
00107     *      IO (        )
00108     *
00109     * @return     IO (        )
00110     *
00111     * @throws arcs::common_interface::AuboException
00112     *
00113     * @par Python
00114     * getToolDigitalInputNum(self: pyaubo_sdk.IoControl) -> int
00115     *
00116     * @par Lua
00117     * getToolDigitalInputNum() -> number
00118     *
00119     * @par Lua
00120     * num = getStandardDigitalInputNum()
00121     *
00122     * @par JSON-RPC
00123     * {"jsonrpc":"2.0","method":"rob1.IoControl.getToolDigitalInputNum","params":[],"id":1}
00124     *
00125     * @par JSON-RPC
00126     * {"id":1,"jsonrpc":"2.0","result":4}
00127     *
00128     * \endchinese
00129     * \english
00130     * Get the number of tool digital IOs (including digital inputs and
00131     * outputs).
00132     *
00133     * @return Number of tool digital IOs (including digital inputs and
00134     * outputs).
00135     *
00136     * @throws arcs::common_interface::AuboException
00137     *
00138     * @par Python function prototype
00139     * getToolDigitalInputNum(self: pyaubo_sdk.IoControl) -> int
00140     *
00141     * @par Lua function prototype
00142     * getToolDigitalInputNum() -> number
00143     *
00144     * @par Lua example
00145     * num = getStandardDigitalInputNum()
00146     *
00147     * @par JSON-RPC request example
00148     * {"jsonrpc":"2.0","method":"rob1.IoControl.getToolDigitalInputNum","params":[],"id":1}
00149     *
00150     * @par JSON-RPC response example
00151     * {"id":1,"jsonrpc":"2.0","result":4}
00152     *
00153     * \endenglish
00154     */
00155    int getToolDigitalInputNum();
00156
00157    /**
00158     * @ingroup IoControl
00159     * \chinese
00160     *
00161     *
00162     * @return
00163     *
00164     * @throws arcs::common_interface::AuboException
00165     *
00166     * @par Python
00167     * getConfigurableDigitalInputNum(self: pyaubo_sdk.IoControl) -> int
00168     *
00169     * @par Lua
```

```
00170     * getConfigurableDigitalInputNum() -> number
00171     *
00172     * @par Lua
00173     * num = getConfigurableDigitalInputNum()
00174     *
00175     * @par JSON-RPC
00176     * {"jsonrpc":"2.0","method":"rob1.IoControl.getConfigurableDigitalInputNum","params":[],"id":1}
00177     *
00178     * @par JSON-RPC
00179     * {"id":1,"jsonrpc":"2.0","result":16}
00180     *
00181     * \endchinese
00182     * \english
00183     * Get the number of configurable digital inputs.
00184     *
00185     * @return Number of configurable digital inputs.
00186     *
00187     * @throws arcs::common_interface::AuboException
00188     *
00189     * @par Python function prototype
00190     * getConfigurableDigitalInputNum(self: pyaubo_sdk.IoControl) -> int
00191     *
00192     * @par Lua function prototype
00193     * getConfigurableDigitalInputNum() -> number
00194     *
00195     * @par Lua example
00196     * num = getConfigurableDigitalInputNum()
00197     *
00198     * @par JSON-RPC request example
00199     * {"jsonrpc":"2.0","method":"rob1.IoControl.getConfigurableDigitalInputNum","params":[],"id":1}
00200     *
00201     * @par JSON-RPC response example
00202     * {"id":1,"jsonrpc":"2.0","result":16}
00203     *
00204     * \endenglish
00205     */
00206    int getConfigurableDigitalInputNum();
00207
00208    /**
00209     * @ingroup IoControl
00210     * \chinese
00211     *
00212     *
00213     * @return
00214     *
00215     * @throws arcs::common_interface::AuboException
00216     *
00217     * @par Python
00218     * getStandardDigitalOutputNum(self: pyaubo_sdk.IoControl) -> int
00219     *
00220     * @par Lua
00221     * getStandardDigitalOutputNum() -> number
00222     *
00223     * @par Lua
00224     * num = getStandardDigitalOutputNum()
00225     *
00226     * @par JSON-RPC
00227     * {"jsonrpc":"2.0","method":"rob1.IoControl.getStandardDigitalOutputNum","params":[],"id":1}
00228     *
00229     * @par JSON-RPC
00230     * {"id":1,"jsonrpc":"2.0","result":8}
00231     *
00232     * \endchinese
00233     * \english
00234     * Get the number of standard digital outputs.
00235     *
00236     * @return Number of standard digital outputs.
00237     *
00238     * @throws arcs::common_interface::AuboException
00239     *
00240     * @par Python function prototype
00241     * getStandardDigitalOutputNum(self: pyaubo_sdk.IoControl) -> int
00242     *
00243     * @par Lua function prototype
00244     * getStandardDigitalOutputNum() -> number
00245     *
00246     * @par Lua example
00247     * num = getStandardDigitalOutputNum()
00248     *
00249     * @par JSON-RPC request example
00250     * {"jsonrpc":"2.0","method":"rob1.IoControl.getStandardDigitalOutputNum","params":[],"id":1}
00251     *
00252     * @par JSON-RPC response example
00253     * {"id":1,"jsonrpc":"2.0","result":8}
00254     *
00255     * \endenglish
00256     */
```

```
00257     int getStandardDigitalOutputNum();
00258
00259     /**
00260      * @ingroup IoControl
00261      * \chinese
00262      *      IO (        )
00263      *
00264      * @return     IO (        )
00265      *
00266      * @throws arcs::common_interface::AuboException
00267      *
00268      * @par Python
00269      * getToolDigitalOutputNum(self: pyaubo_sdk.IoControl) -> int
00270      *
00271      * @par Lua
00272      * getToolDigitalOutputNum() -> number
00273      *
00274      * @par Lua
00275      * num = getToolDigitalOutputNum()
00276      *
00277      * @par JSON-RPC
00278      * {"jsonrpc":"2.0","method":"rob1.IoControl.getToolDigitalOutputNum","params":[],"id":1}
00279      *
00280      * @par JSON-RPC
00281      * {"id":1,"jsonrpc":"2.0","result":4}
00282      *
00283      * \endchinese
00284      * \english
00285      * Get the number of tool digital IOs (including digital inputs and
00286      * outputs).
00287      *
00288      * @return Number of tool digital IOs (including digital inputs and
00289      * outputs).
00290      *
00291      * @throws arcs::common_interface::AuboException
00292      *
00293      * @par Python function prototype
00294      * getToolDigitalOutputNum(self: pyaubo_sdk.IoControl) -> int
00295      *
00296      * @par Lua function prototype
00297      * getToolDigitalOutputNum() -> number
00298      *
00299      * @par Lua example
00300      * num = getToolDigitalOutputNum()
00301      *
00302      * @par JSON-RPC request example
00303      * {"jsonrpc":"2.0","method":"rob1.IoControl.getToolDigitalOutputNum","params":[],"id":1}
00304      *
00305      * @par JSON-RPC response example
00306      * {"id":1,"jsonrpc":"2.0","result":4}
00307      *
00308      * \endenglish
00309      */
00310     int getToolDigitalOutputNum();
00311
00312     /**
00313      * @ingroup IoControl
00314      * \chinese
00315      *        IO
00316      *
00317      *     IO    IO
00318      *
00319      * @param index:   IO       0
00320      *    0
00321      * @param input:    IO
00322      * input  true     IO
00323      * @return    0
00324      * AUBO_BUSY
00325      * AUBO_BAD_STATE
00326      * -AUBO_INVL_ARGUMENT
00327      * -AUBO_BAD_STATE
00328      *
00329      * @throws arcs::common_interface::AuboException
00330      *
00331      * @par Python
00332      * setToolIoInput(self: pyaubo_sdk.IoControl, arg0: int, arg1: bool) -> int
00333      *
00334      * @par Lua
00335      * setToolIoInput(index: number, input: boolean) -> nil
00336      *
00337      * @par Lua
00338      * setToolIoInput(0,true)
00339      *
00340      * @par JSON-RPC
00341      * {"jsonrpc":"2.0","method":"rob1.IoControl.setToolIoInput","params":[0,true],"id":1}
00342      *
00343      * @par JSON-RPC
```

```
00344     * {"id":1,"jsonrpc":"2.0","result":0}
00345     *
00346     * \endchinese
00347     * \english
00348     * Set the specified tool digital IO as input or output.
00349     *
00350     * Tool digital IOs are special and can be configured as input or output.
00351     *
00352     * @param index: Indicates the IO pin, starting from 0.
00353     * For example, 0 means the first pin.
00354     * @param input: Indicates whether the specified IO is input.
00355     * If input is true, set the IO as input; otherwise, set as output.
00356     * @return Returns 0 on success; error code on failure.
00357     * AUBO_BUSY
00358     * AUBO_BAD_STATE
00359     * -AUBO_INVL_ARGUMENT
00360     * -AUBO_BAD_STATE
00361     *
00362     * @throws arcs::common_interface::AuboException
00363     *
00364     * @par Python function prototype
00365     * setToolIoInput(self: pyaubo_sdk.IoControl, arg0: int, arg1: bool) -> int
00366     *
00367     * @par Lua function prototype
00368     * setToolIoInput(index: number, input: boolean) -> nil
00369     *
00370     * @par Lua example
00371     * setToolIoInput(0,true)
00372     *
00373     * @par JSON-RPC request example
00374     * {"jsonrpc":"2.0","method":"rob1.IoControl.setToolIoInput","params":[0,true],"id":1}
00375     *
00376     * @par JSON-RPC response example
00377     * {"id":1,"jsonrpc":"2.0","result":0}
00378     *
00379     * \endenglish
00380     */
00381    int setToolIoInput(int index, bool input);
00382
00383    /**
00384     * @ingroup IoControl
00385     * \chinese
00386     *        IO
00387     *
00388     * @param index:   IO       0
00389     *   0
00390     * @return    IO      true,    false
00391     *
00392     * @throws arcs::common_interface::AuboException
00393     *
00394     * @par Python
00395     * isToolIoInput(self: pyaubo_sdk.IoControl, arg0: int) -> bool
00396     *
00397     * @par Lua
00398     * isToolIoInput(index: number) -> boolean
00399     *
00400     * @par Lua
00401     * status = isToolIoInput(0)
00402     *
00403     * @par JSON-RPC
00404     * {"jsonrpc":"2.0","method":"rob1.IoControl.isToolIoInput","params":[0],"id":1}
00405     *
00406     * @par JSON-RPC
00407     * {"id":1,"jsonrpc":"2.0","result":true}
00408     *
00409     * \endchinese
00410     * \english
00411     * Determine whether the specified tool digital IO is configured as input.
00412     *
00413     * @param index: Indicates the IO pin, starting from 0.
00414     * For example, 0 means the first pin.
00415     * @return Returns true if the specified IO is input, otherwise false.
00416     *
00417     * @throws arcs::common_interface::AuboException
00418     *
00419     * @par Python function prototype
00420     * isToolIoInput(self: pyaubo_sdk.IoControl, arg0: int) -> bool
00421     *
00422     * @par Lua function prototype
00423     * isToolIoInput(index: number) -> boolean
00424     *
00425     * @par Lua example
00426     * status = isToolIoInput(0)
00427     *
00428     * @par JSON-RPC request example
00429     * {"jsonrpc":"2.0","method":"rob1.IoControl.isToolIoInput","params":[0],"id":1}
00430     *
```

```
00431     * @par JSON-RPC response example
00432     * {”id”:1,”jsonrpc”:”2.0”,”result”:true}
00433     *
00434     * \endenglish
00435     */
00436    bool isToolIoInput(int index);
00437
00438    /**
00439     * @ingroup IoControl
00440     * \chinese
00441     *
00442     *
00443     * @return
00444     *
00445     * @throws arcs::common_interface::AuboException
00446     *
00447     * @par Python
00448     * getConfigurableDigitalOutputNum(self: pyaubo_sdk.IoControl) -> int
00449     *
00450     * @par Lua
00451     * getConfigurableDigitalOutputNum() -> number
00452     *
00453     * @par Lua
00454     * num = getConfigurableDigitalOutputNum()
00455     *
00456     * @par JSON-RPC
00457     * {”jsonrpc”:”2.0”,”method”:”rob1.IoControl.getConfigurableDigitalOutputNum”,”params”:[],”id”:1}
00458     *
00459     * @par JSON-RPC
00460     * {”id”:1,”jsonrpc”:”2.0”,”result”:16}
00461     *
00462     * \endchinese
00463     * \english
00464     * Get the number of configurable digital outputs.
00465     *
00466     * @return Number of configurable digital outputs.
00467     *
00468     * @throws arcs::common_interface::AuboException
00469     *
00470     * @par Python function prototype
00471     * getConfigurableDigitalOutputNum(self: pyaubo_sdk.IoControl) -> int
00472     *
00473     * @par Lua function prototype
00474     * getConfigurableDigitalOutputNum() -> number
00475     *
00476     * @par Lua example
00477     * num = getConfigurableDigitalOutputNum()
00478     *
00479     * @par JSON-RPC request example
00480     * {”jsonrpc”:”2.0”,”method”:”rob1.IoControl.getConfigurableDigitalOutputNum”,”params”:[],”id”:1}
00481     *
00482     * @par JSON-RPC response example
00483     * {”id”:1,”jsonrpc”:”2.0”,”result”:16}
00484     *
00485     * \endenglish
00486     */
00487    int getConfigurableDigitalOutputNum();
00488
00489    /**
00490     * @ingroup IoControl
00491     * \chinese
00492     *
00493     *
00494     * @return
00495     *
00496     * @throws arcs::common_interface::AuboException
00497     *
00498     * @par Python
00499     * getStandardAnalogInputNum(self: pyaubo_sdk.IoControl) -> int
00500     *
00501     * @par Lua
00502     * getStandardAnalogInputNum() -> number
00503     *
00504     * @par Lua
00505     * num = getStandardAnalogInputNum()
00506     *
00507     * @par JSON-RPC
00508     * {”jsonrpc”:”2.0”,”method”:”rob1.IoControl.getStandardAnalogInputNum”,”params”:[],”id”:1}
00509     *
00510     * @par JSON-RPC
00511     * {”id”:1,”jsonrpc”:”2.0”,”result”:2}
00512     *
00513     * \endchinese
00514     * \english
00515     * Get the number of standard analog inputs.
00516     *
00517     * @return Number of standard analog inputs.
```

```
00518     *
00519     * @throws arcs::common_interface::AuboException
00520     *
00521     * @par Python function prototype
00522     * getStandardAnalogInputNum(self: pyaubo_sdk.IoControl) -> int
00523     *
00524     * @par Lua function prototype
00525     * getStandardAnalogInputNum() -> number
00526     *
00527     * @par Lua example
00528     * num = getStandardAnalogInputNum()
00529     *
00530     * @par JSON-RPC request example
00531     * {"jsonrpc":"2.0","method":"rob1.IoControl.getStandardAnalogInputNum","params":[],"id":1}
00532     *
00533     * @par JSON-RPC response example
00534     * {"id":1,"jsonrpc":"2.0","result":2}
00535     *
00536     * \endenglish
00537     */
00538    int getStandardAnalogInputNum();
00539
00540    /**
00541     * @ingroup IoControl
00542     * \chinese
00543     *
00544     *
00545     * @return
00546     *
00547     * @throws arcs::common_interface::AuboException
00548     *
00549     * @par Python
00550     * getToolAnalogInputNum(self: pyaubo_sdk.IoControl) -> int
00551     *
00552     * @par Lua
00553     * getToolAnalogInputNum() -> number
00554     *
00555     * @par Lua
00556     * num = getToolAnalogInputNum()
00557     *
00558     * @par JSON-RPC
00559     * {"jsonrpc":"2.0","method":"rob1.IoControl.getToolAnalogInputNum","params":[],"id":1}
00560     *
00561     * @par JSON-RPC
00562     * {"id":1,"jsonrpc":"2.0","result":2}
00563     *
00564     * \endchinese
00565     * \english
00566     * Get the number of tool analog inputs.
00567     *
00568     * @return Number of tool analog inputs.
00569     *
00570     * @throws arcs::common_interface::AuboException
00571     *
00572     * @par Python function prototype
00573     * getToolAnalogInputNum(self: pyaubo_sdk.IoControl) -> int
00574     *
00575     * @par Lua function prototype
00576     * getToolAnalogInputNum() -> number
00577     *
00578     * @par Lua example
00579     * num = getToolAnalogInputNum()
00580     *
00581     * @par JSON-RPC request example
00582     * {"jsonrpc":"2.0","method":"rob1.IoControl.getToolAnalogInputNum","params":[],"id":1}
00583     *
00584     * @par JSON-RPC response example
00585     * {"id":1,"jsonrpc":"2.0","result":2}
00586     *
00587     * \endenglish
00588     */
00589    int getToolAnalogInputNum();
00590
00591    /**
00592     * @ingroup IoControl
00593     * \chinese
00594     *
00595     *
00596     * @return
00597     *
00598     * @throws arcs::common_interface::AuboException
00599     *
00600     * @par Python
00601     * getStandardAnalogOutputNum(self: pyaubo_sdk.IoControl) -> int
00602     *
00603     * @par Lua
00604     * getStandardAnalogOutputNum() -> number
```

```
00605     *
00606     * @par Lua
00607     * num = getStandardAnalogOutputNum()
00608     *
00609     * @par JSON-RPC
00610     * {"jsonrpc":"2.0","method":"rob1.IoControl.getStandardAnalogOutputNum","params":[],"id":1}
00611     *
00612     * @par JSON-RPC
00613     * {"id":1,"jsonrpc":"2.0","result":2}
00614     *
00615     * \endchinese
00616     * \english
00617     * Get the number of standard analog outputs.
00618     *
00619     * @return Number of standard analog outputs.
00620     *
00621     * @throws arcs::common_interface::AuboException
00622     *
00623     * @par Python function prototype
00624     * getStandardAnalogOutputNum(self: pyaubo_sdk.IoControl) -> int
00625     *
00626     * @par Lua function prototype
00627     * getStandardAnalogOutputNum() -> number
00628     *
00629     * @par Lua example
00630     * num = getStandardAnalogOutputNum()
00631     *
00632     * @par JSON-RPC request example
00633     * {"jsonrpc":"2.0","method":"rob1.IoControl.getStandardAnalogOutputNum","params":[],"id":1}
00634     *
00635     * @par JSON-RPC response example
00636     * {"id":1,"jsonrpc":"2.0","result":2}
00637     *
00638     * \endenglish
00639     */
00640    int getStandardAnalogOutputNum();
00641
00642    /**
00643     * @ingroup IoControl
00644     * \chinese
00645     *
00646     *
00647     * @return
00648     *
00649     * @throws arcs::common_interface::AuboException
00650     *
00651     * @par Python
00652     * getToolAnalogOutputNum(self: pyaubo_sdk.IoControl) -> int
00653     *
00654     * @par Lua
00655     * getToolAnalogOutputNum() -> number
00656     *
00657     * @par Lua
00658     * num = getToolAnalogOutputNum()
00659     *
00660     * @par JSON-RPC
00661     * {"jsonrpc":"2.0","method":"rob1.IoControl.getToolAnalogOutputNum","params":[],"id":1}
00662     *
00663     * @par JSON-RPC
00664     * {"id":1,"jsonrpc":"2.0","result":0}
00665     *
00666     * \endchinese
00667     * \english
00668     * Get the number of tool analog outputs.
00669     *
00670     * @return Number of tool analog outputs.
00671     *
00672     * @throws arcs::common_interface::AuboException
00673     *
00674     * @par Python function prototype
00675     * getToolAnalogOutputNum(self: pyaubo_sdk.IoControl) -> int
00676     *
00677     * @par Lua function prototype
00678     * getToolAnalogOutputNum() -> number
00679     *
00680     * @par Lua example
00681     * num = getToolAnalogOutputNum()
00682     *
00683     * @par JSON-RPC request example
00684     * {"jsonrpc":"2.0","method":"rob1.IoControl.getToolAnalogOutputNum","params":[],"id":1}
00685     *
00686     * @par JSON-RPC response example
00687     * {"id":1,"jsonrpc":"2.0","result":0}
00688     *
00689     * \endenglish
00690     */
00691    int getToolAnalogOutputNum();
```

```
00692
00693     /**
00694      * @ingroup IoControl
00695      * \chinese
00696      *
00697      *
00698      * @note
00699      *
00700      *
00701      * @return    0
00702      * AUBO_BUSY
00703      * AUBO_BAD_STATE
00704      * -AUBO_BAD_STATE
00705      *
00706      * @throws arcs::common_interface::AuboException
00707      *
00708      * @par Python
00709      * setDigitalInputActionDefault(self: pyaubo_sdk.IoControl) -> int
00710      *
00711      * @par Lua
00712      * setDigitalInputActionDefault() -> nil
00713      *
00714      * @par Lua
00715      * setDigitalInputActionDefault()
00716      *
00717      * @par JSON-RPC
00718      * {"jsonrpc":"2.0","method":"rob1.IoControl.setDigitalInputActionDefault","params":[],"id":1}
00719      *
00720      * @par JSON-RPC
00721      * {"id":1,"jsonrpc":"2.0","result":0}
00722      *
00723      * \endchinese
00724      * \english
00725      * Set all digital input actions to no trigger.
00726      *
00727      * @note
00728      * When the input action is set to no trigger, setting the digital input
00729      * value to high will not trigger any robot action.
00730      *
00731      * @return Returns 0 on success; error code on failure.
00732      * AUBO_BUSY
00733      * AUBO_BAD_STATE
00734      * -AUBO_BAD_STATE
00735      *
00736      * @throws arcs::common_interface::AuboException
00737      *
00738      * @par Python function prototype
00739      * setDigitalInputActionDefault(self: pyaubo_sdk.IoControl) -> int
00740      *
00741      * @par Lua function prototype
00742      * setDigitalInputActionDefault() -> nil
00743      *
00744      * @par Lua example
00745      * setDigitalInputActionDefault()
00746      *
00747      * @par JSON-RPC request example
00748      * {"jsonrpc":"2.0","method":"rob1.IoControl.setDigitalInputActionDefault","params":[],"id":1}
00749      *
00750      * @par JSON-RPC response example
00751      * {"id":1,"jsonrpc":"2.0","result":0}
00752      *
00753      * \endenglish
00754      */
00755     int setDigitalInputActionDefault();
00756
00757     /**
00758      * @ingroup IoControl
00759      * \chinese
00760      *
00761      *
00762      * @note
00763      *          (StandardInputAction::Default)
00764      *                  \n
00765      *                          \n
00766      *      DI0        (StandardInputAction::Handguide)
00767      *     DI0
00768      *   DI0
00769      *
00770      * @param index:   IO        0
00771      *    0
00772      * @param action:
00773      *
00774      * @return    0
00775      * AUBO_REQUEST_IGNORE
00776      * AUBO_BUSY
00777      * AUBO_BAD_STATE
00778      * -AUBO_INVL_ARGUMENT
```

```
00779     * -AUBO_BAD_STATE
00780     *
00781     * @throws arcs::common_interface::AuboException
00782     *
00783     * @par Python
00784     * setStandardDigitalInputAction(self: pyaubo_sdk.IoControl, arg0: int,
00785     * arg1: arcs::common_interface::StandardInputAction) -> int
00786     *
00787     * @par Lua
00788     * setStandardDigitalInputAction(index: number, action: number) -> nil
00789     *
00790     * @par Lua
00791     * setStandardDigitalInputAction(0,1)
00792     *
00793     * @par JSON-RPC
00794     * {"jsonrpc":"2.0","method":"rob1.IoControl.setStandardDigitalInputAction","params":[0,"Handguide"],"id":1}
00795     *
00796     * @par JSON-RPC
00797     * {"id":1,"jsonrpc":"2.0","result":0}
00798     *
00799     * \endchinese
00800     * \english
00801     * Set the trigger action for standard digital input.
00802     *
00803     * @note
00804     * When the input is set to no trigger action
00805     * (StandardInputAction::Default), setting the digital input value to high
00806     * will not trigger any robot action.\n When a trigger action is set,
00807     * setting the digital input value to high will trigger the corresponding
00808     * robot action.\n For example, if DI0 is set to trigger Handguide
00809     * (StandardInputAction::Handguide), setting DI0 to high will enable
00810     * hand-guiding mode. Setting DI0 to low will exit hand-guiding mode.
00811     *
00812     * @param index: Indicates the IO pin, starting from 0.
00813     * For example, 0 means the first pin.
00814     * @param action: Trigger action
00815     *
00816     * @return Returns 0 on success; error code on failure.
00817     * AUBO_REQUEST_IGNORE
00818     * AUBO_BUSY
00819     * AUBO_BAD_STATE
00820     * -AUBO_INVL_ARGUMENT
00821     * -AUBO_BAD_STATE
00822     *
00823     * @throws arcs::common_interface::AuboException
00824     *
00825     * @par Python function prototype
00826     * setStandardDigitalInputAction(self: pyaubo_sdk.IoControl, arg0: int,
00827     * arg1: arcs::common_interface::StandardInputAction) -> int
00828     *
00829     * @par Lua function prototype
00830     * setStandardDigitalInputAction(index: number, action: number) -> nil
00831     *
00832     * @par Lua example
00833     * setStandardDigitalInputAction(0,1)
00834     *
00835     * @par JSON-RPC request example
00836     * {"jsonrpc":"2.0","method":"rob1.IoControl.setStandardDigitalInputAction","params":[0,"Handguide"],"id":1}
00837     *
00838     * @par JSON-RPC response example
00839     * {"id":1,"jsonrpc":"2.0","result":0}
00840     *
00841     * \endenglish
00842     */
00843    int setStandardDigitalInputAction(int index, StandardInputAction action);
00844
00845    /**
00846     * @ingroup IoControl
00847     * \chinese
00848     *
00849     *
00850     * @note
00851     *          (StandardInputAction::Default)
00852     *                      \n
00853     *                              \n
00854     *      TOOL_IO[0]              (StandardInputAction::Handguide)
00855     *     TOOL_IO[0]
00856     *    TOOL_IO[0]
00857     *
00858     * @param index:   IO       0
00859     *    0
00860     * @param action:
00861     *
00862     * @return     0
00863     * AUBO_REQUEST_IGNORE
00864     * AUBO_BUSY
00865     * AUBO_BAD_STATE
```

```
00866     * -AUBO_INVL_ARGUMENT
00867     * -AUBO_BAD_STATE
00868     *
00869     * @throws arcs::common_interface::AuboException
00870     *
00871     * @par Python
00872     * setToolDigitalInputAction(self: pyaubo_sdk.IoControl, arg0: int, arg1:
00873     * arcs::common_interface::StandardInputAction) -> int
00874     *
00875     * @par Lua
00876     * setToolDigitalInputAction(index: number, action: number) -> nil
00877     *
00878     * @par Lua
00879     * setToolDigitalInputAction(0,1)
00880     *
00881     * @par JSON-RPC
00882     * {"jsonrpc":"2.0","method":"rob1.IoControl.setToolDigitalInputAction","params":[0,"Handguide"],"id":1}
00883     *
00884     * @par JSON-RPC
00885     * {"id":1,"jsonrpc":"2.0","result":0}
00886     *
00887     * \endchinese
00888     * \english
00889     * Set the trigger action for tool digital input.
00890     *
00891     * @note
00892     * When the input is set to no trigger action
00893     * (StandardInputAction::Default), setting the tool digital input value to
00894     * high will not trigger any robot action.\n When a trigger action is set,
00895     * setting the tool digital input value to high will trigger the
00896     * corresponding robot action.\n For example, if TOOL_IO[0] is set as input
00897     * and its trigger action is Handguide (StandardInputAction::Handguide),
00898     * setting TOOL_IO[0] to high will enable hand-guiding mode.
00899     * Setting TOOL_IO[0] to low will exit hand-guiding mode.
00900     *
00901     * @param index: Indicates the IO pin, starting from 0.
00902     * For example, 0 means the first pin.
00903     * @param action: Trigger action
00904     *
00905     * @return Returns 0 on success; error code on failure.
00906     * AUBO_REQUEST_IGNORE
00907     * AUBO_BUSY
00908     * AUBO_BAD_STATE
00909     * -AUBO_INVL_ARGUMENT
00910     * -AUBO_BAD_STATE
00911     *
00912     * @throws arcs::common_interface::AuboException
00913     *
00914     * @par Python function prototype
00915     * setToolDigitalInputAction(self: pyaubo_sdk.IoControl, arg0: int, arg1:
00916     * arcs::common_interface::StandardInputAction) -> int
00917     *
00918     * @par Lua function prototype
00919     * setToolDigitalInputAction(index: number, action: number) -> nil
00920     *
00921     * @par Lua example
00922     * setToolDigitalInputAction(0,1)
00923     *
00924     * @par JSON-RPC request example
00925     * {"jsonrpc":"2.0","method":"rob1.IoControl.setToolDigitalInputAction","params":[0,"Handguide"],"id":1}
00926     *
00927     * @par JSON-RPC response example
00928     * {"id":1,"jsonrpc":"2.0","result":0}
00929     *
00930     * \endenglish
00931     */
00932    int setToolDigitalInputAction(int index, StandardInputAction action);
00933
00934    /**
00935     * @ingroup IoControl
00936     * \chinese
00937     *
00938     *
00939     * @param index:   IO       0
00940     *   0
00941     * @param action:
00942     *
00943     * @return    0
00944     * AUBO_REQUEST_IGNORE
00945     * AUBO_BUSY
00946     * AUBO_BAD_STATE
00947     * -AUBO_INVL_ARGUMENT
00948     * -AUBO_BAD_STATE
00949     *
00950     * @throws arcs::common_interface::AuboException
00951     *
00952     * @note
```

```
00953     * SafetyInputAction::Unassigned
00954     *
00955     * @par Python
00956     * setConfigurableDigitalInputAction(self: pyaubo_sdk.IoControl, arg0: int,
00957     * arg1: arcs::common_interface::StandardInputAction) -> int
00958     *
00959     * @par Lua
00960     * setConfigurableDigitalInputAction(index: number, action: number) -> nil
00961     *
00962     * @par Lua
00963     * setConfigurableDigitalInputAction(0,1)
00964     *
00965     * @par JSON-RPC
00966     * {"jsonrpc":"2.0","method":"rob1.IoControl.setConfigurableDigitalInputAction","params":[0,"Handguide"],"id":1}
00967     *
00968     * @par JSON-RPC
00969     * {"id":1,"jsonrpc":"2.0","result":0}
00970     *
00971     * \endchinese
00972     * \english
00973     * Set the trigger action for configurable digital input.
00974     *
00975     * @param index: Indicates the IO pin, starting from 0.
00976     * For example, 0 means the first pin.
00977     * @param action: Trigger action
00978     *
00979     * @return Returns 0 on success; error code on failure.
00980     * AUBO_REQUEST_IGNORE
00981     * AUBO_BUSY
00982     * AUBO_BAD_STATE
00983     * -AUBO_INVL_ARGUMENT
00984     * -AUBO_BAD_STATE
00985     *
00986     * @throws arcs::common_interface::AuboException
00987     *
00988     * @note This function takes effect only when the safety input action of the
00989     * configurable input is set to SafetyInputAction::Unassigned.
00990     *
00991     * @par Python function prototype
00992     * setConfigurableDigitalInputAction(self: pyaubo_sdk.IoControl, arg0: int,
00993     * arg1: arcs::common_interface::StandardInputAction) -> int
00994     *
00995     * @par Lua function prototype
00996     * setConfigurableDigitalInputAction(index: number, action: number) -> nil
00997     *
00998     * @par Lua example
00999     * setConfigurableDigitalInputAction(0,1)
01000     *
01001     * @par JSON-RPC request example
01002     * {"jsonrpc":"2.0","method":"rob1.IoControl.setConfigurableDigitalInputAction","params":[0,"Handguide"],"id":1}
01003     *
01004     * @par JSON-RPC response example
01005     * {"id":1,"jsonrpc":"2.0","result":0}
01006     *
01007     * \endenglish
01008     */
01009    int setConfigurableDigitalInputAction(int index,
01010                                          StandardInputAction action);
01011
01012    /**
01013     * @ingroup IoControl
01014     * \chinese
01015     *
01016     *
01017     * @param index:   IO       0
01018     *   0
01019     * @return
01020     *
01021     * @par Python
01022     * getStandardDigitalInputAction(self: pyaubo_sdk.IoControl, arg0: int) ->
01023     * arcs::common_interface::StandardInputAction
01024     *
01025     * @par Lua
01026     * getStandardDigitalInputAction(index: number) -> number
01027     *
01028     * @par Lua
01029     * num = getStandardDigitalInputAction(0)
01030     *
01031     * @par JSON-RPC
01032     * {"jsonrpc":"2.0","method":"rob1.IoControl.getStandardDigitalInputAction","params":[0],"id":1}
01033     *
01034     * @par JSON-RPC
01035     * {"id":1,"jsonrpc":"2.0","result":"Default"}
01036     *
01037     * \endchinese
01038     * \english
01039     * Get the trigger action for standard digital input.
```

```
01040     *
01041     * @param index: Indicates the IO pin, starting from 0.
01042     * For example, 0 means the first pin.
01043     * @return Standard digital input trigger action
01044     *
01045     * @par Python function prototype
01046     * getStandardDigitalInputAction(self: pyaubo_sdk.IoControl, arg0: int) ->
01047     * arcs::common_interface::StandardInputAction
01048     *
01049     * @par Lua function prototype
01050     * getStandardDigitalInputAction(index: number) -> number
01051     *
01052     * @par Lua example
01053     * num = getStandardDigitalInputAction(0)
01054     *
01055     * @par JSON-RPC request example
01056     * {"jsonrpc":"2.0","method":"rob1.IoControl.getStandardDigitalInputAction","params":[0],"id":1}
01057     *
01058     * @par JSON-RPC response example
01059     * {"id":1,"jsonrpc":"2.0","result":"Default"}
01060     *
01061     * \endenglish
01062     */
01063    StandardInputAction getStandardDigitalInputAction(int index);
01064
01065    /**
01066     * @ingroup IoControl
01067     * \chinese
01068     *
01069     *
01070     * @param index:   IO       0
01071     *   0
01072     * @return
01073     *
01074     * @par Python
01075     * getToolDigitalInputAction(self: pyaubo_sdk.IoControl, arg0: int) ->
01076     * arcs::common_interface::StandardInputAction
01077     *
01078     * @par Lua
01079     * getToolDigitalInputAction(index: number) -> number
01080     *
01081     * @par Lua
01082     * getToolDigitalInputAction(0)
01083     *
01084     * @par JSON-RPC
01085     * {"jsonrpc":"2.0","method":"rob1.IoControl.getToolDigitalInputAction","params":[0],"id":1}
01086     *
01087     * @par JSON-RPC
01088     * {"id":1,"jsonrpc":"2.0","result":"Default"}
01089     *
01090     * \endchinese
01091     * \english
01092     * Get the trigger action for tool digital input.
01093     *
01094     * @param index: Indicates the IO pin, starting from 0.
01095     * For example, 0 means the first pin.
01096     * @return Tool digital input trigger action
01097     *
01098     * @par Python function prototype
01099     * getToolDigitalInputAction(self: pyaubo_sdk.IoControl, arg0: int) ->
01100     * arcs::common_interface::StandardInputAction
01101     *
01102     * @par Lua function prototype
01103     * getToolDigitalInputAction(index: number) -> number
01104     *
01105     * @par Lua example
01106     * getToolDigitalInputAction(0)
01107     *
01108     * @par JSON-RPC request example
01109     * {"jsonrpc":"2.0","method":"rob1.IoControl.getToolDigitalInputAction","params":[0],"id":1}
01110     *
01111     * @par JSON-RPC response example
01112     * {"id":1,"jsonrpc":"2.0","result":"Default"}
01113     *
01114     * \endenglish
01115     */
01116    StandardInputAction getToolDigitalInputAction(int index);
01117
01118    /**
01119     * @ingroup IoControl
01120     * \chinese
01121     *
01122     *
01123     * @param index:   IO       0
01124     *   0
01125     * @return
01126     *
```

```
01127     * @par Python
01128     * getConfigurableDigitalInputAction(self: pyaubo_sdk.IoControl, arg0: int)
01129     * -> arcs::common_interface::StandardInputAction
01130     *
01131     * @par Lua
01132     * getConfigurableDigitalInputAction(index: number) -> number
01133     *
01134     * @par Lua
01135     * num = getConfigurableDigitalInputAction(0)
01136     *
01137     * @par JSON-RPC
01138     * {"jsonrpc":"2.0","method":"rob1.IoControl.getConfigurableDigitalInputAction","params":[0],"id":1}
01139     *
01140     * @par JSON-RPC
01141     * {"id":1,"jsonrpc":"2.0","result":"Default"}
01142     *
01143     * \endchinese
01144     * \english
01145     * Get the trigger action for configurable digital input.
01146     *
01147     * @param index: Indicates the IO pin, starting from 0.
01148     * For example, 0 means the first pin.
01149     * @return Returns the input trigger action.
01150     *
01151     * @par Python function prototype
01152     * getConfigurableDigitalInputAction(self: pyaubo_sdk.IoControl, arg0: int)
01153     * -> arcs::common_interface::StandardInputAction
01154     *
01155     * @par Lua function prototype
01156     * getConfigurableDigitalInputAction(index: number) -> number
01157     *
01158     * @par Lua example
01159     * num = getConfigurableDigitalInputAction(0)
01160     *
01161     * @par JSON-RPC request example
01162     * {"jsonrpc":"2.0","method":"rob1.IoControl.getConfigurableDigitalInputAction","params":[0],"id":1}
01163     *
01164     * @par JSON-RPC response example
01165     * {"id":1,"jsonrpc":"2.0","result":"Default"}
01166     *
01167     * \endenglish
01168     */
01169    StandardInputAction getConfigurableDigitalInputAction(int index);
01170
01171    /**
01172     * @ingroup IoControl
01173     * \chinese
01174     *
01175     *
01176     * @note
01177     *        (StandardOutputRunState::None)
01178     *         \n
01179     *                        \n
01180     *     DO0            (StandardOutputRunState::Handguiding)
01181     *        DO0
01182     *        DO0
01183     *
01184     * @return    0
01185     * AUBO_BUSY
01186     * AUBO_BAD_STATE
01187     * -AUBO_BAD_STATE
01188     *
01189     * @throws arcs::common_interface::AuboException
01190     *
01191     * @par Python
01192     * setDigitalOutputRunstateDefault(self: pyaubo_sdk.IoControl) -> int
01193     *
01194     * @par Lua
01195     * setDigitalOutputRunstateDefault() -> nil
01196     *
01197     * @par Lua
01198     * setDigitalOutputRunstateDefault()
01199     *
01200     * @par JSON-RPC
01201     * {"jsonrpc":"2.0","method":"rob1.IoControl.setDigitalOutputRunstateDefault","params":[],"id":1}
01202     *
01203     * @par JSON-RPC
01204     * {"id":1,"jsonrpc":"2.0","result":0}
01205     *
01206     * \endchinese
01207     * \english
01208     * Set all digital output runstates to None.
01209     *
01210     * @note
01211     * When the output runstate is set to None (StandardOutputRunState::None),
01212     * users can set the digital output value.\n
01213     * When the output runstate is set, users cannot set the digital output
```

```
01214     * value, and the controller will set it automatically.\n For example, when
01215     * DO0's output runstate is set to indicate hand-guiding
01216     * (StandardOutputRunState::Handguiding), the robot enters hand-guiding mode
01217     * and DO0 will automatically become high. When the robot exits
01218     * hand-guiding, DO0 will automatically become low.
01219     *
01220     * @return Returns 0 on success; error code on failure.
01221     * AUBO_BUSY
01222     * AUBO_BAD_STATE
01223     * -AUBO_BAD_STATE
01224     *
01225     * @throws arcs::common_interface::AuboException
01226     *
01227     * @par Python function prototype
01228     * setDigitalOutputRunstateDefault(self: pyaubo_sdk.IoControl) -> int
01229     *
01230     * @par Lua function prototype
01231     * setDigitalOutputRunstateDefault() -> nil
01232     *
01233     * @par Lua example
01234     * setDigitalOutputRunstateDefault()
01235     *
01236     * @par JSON-RPC request example
01237     * {"jsonrpc":"2.0","method":"rob1.IoControl.setDigitalOutputRunstateDefault","params":[],"id":1}
01238     *
01239     * @par JSON-RPC response example
01240     * {"id":1,"jsonrpc":"2.0","result":0}
01241     *
01242     * \endenglish
01243     */
01244    int setDigitalOutputRunstateDefault();
01245
01246    /**
01247     * @ingroup IoControl
01248     * \chinese
01249     *
01250     *
01251     * @note
01252     *        (StandardOutputRunState::None)
01253     *         \n
01254     *                        \n
01255     *     DO0            (StandardOutputRunState::Handguiding)
01256     *        DO0
01257     *        DO0
01258     *
01259     * @param index:   IO       0
01260     *   0
01261     * @param runstate:
01262     *
01263     * @return    0
01264     * AUBO_REQUEST_IGNORE
01265     * AUBO_BUSY
01266     * AUBO_BAD_STATE
01267     * -AUBO_INVL_ARGUMENT
01268     * -AUBO_BAD_STATE
01269     *
01270     * @throws arcs::common_interface::AuboException
01271     *
01272     * @par Python
01273     * setStandardDigitalOutputRunstate(self: pyaubo_sdk.IoControl, arg0: int,
01274     * arg1: arcs::common_interface::StandardOutputRunState) -> int
01275     *
01276     * @par Lua
01277     * setStandardDigitalOutputRunstate(index: number, runstate: number) -> nil
01278     *
01279     * @par Lua
01280     * setStandardDigitalOutputRunstate(0,1)
01281     *
01282     * @par JSON-RPC
01283     * {"jsonrpc":"2.0","method":"rob1.IoControl.setStandardDigitalOutputRunstate","params":[0,"PowerOn"],"id":1}
01284     *
01285     * @par JSON-RPC
01286     * {"id":1,"jsonrpc":"2.0","result":0}
01287     *
01288     * \endchinese
01289     * \english
01290     * Set the runstate for standard digital output.
01291     *
01292     * @note
01293     * When the output runstate is set to None (StandardOutputRunState::None),
01294     * users can set the digital output value.\n
01295     * When the output runstate is set, users cannot set the digital output
01296     * value, and the controller will set it automatically.\n For example, when
01297     * DO0's output runstate is set to indicate hand-guiding
01298     * (StandardOutputRunState::Handguiding), the robot enters hand-guiding mode
01299     * and DO0 will automatically become high. When the robot exits
01300     * hand-guiding, DO0 will automatically become low.
```

```
01301     *
01302     * @param index: Indicates the IO pin, starting from 0.
01303     * For example, 0 means the first pin.
01304     * @param runstate: Output runstate selection
01305     *
01306     * @return Returns 0 on success; error code on failure.
01307     * AUBO_REQUEST_IGNORE
01308     * AUBO_BUSY
01309     * AUBO_BAD_STATE
01310     * -AUBO_INVL_ARGUMENT
01311     * -AUBO_BAD_STATE
01312     *
01313     * @throws arcs::common_interface::AuboException
01314     *
01315     * @par Python function prototype
01316     * setStandardDigitalOutputRunstate(self: pyaubo_sdk.IoControl, arg0: int,
01317     * arg1: arcs::common_interface::StandardOutputRunState) -> int
01318     *
01319     * @par Lua function prototype
01320     * setStandardDigitalOutputRunstate(index: number, runstate: number) -> nil
01321     *
01322     * @par Lua example
01323     * setStandardDigitalOutputRunstate(0,1)
01324     *
01325     * @par JSON-RPC request example
01326     * {"jsonrpc":"2.0","method":"rob1.IoControl.setStandardDigitalOutputRunstate","params":[0,"PowerOn"],"id":1}
01327     *
01328     * @par JSON-RPC response example
01329     * {"id":1,"jsonrpc":"2.0","result":0}
01330     *
01331     * \endenglish
01332     */
01333    int setStandardDigitalOutputRunstate(int index,
01334                                         StandardOutputRunState runstate);
01335
01336    /**
01337     * @ingroup IoControl
01338     * \chinese
01339     *
01340     *
01341     * @note
01342     *       (StandardOutputRunState::None)
01343     *        \n
01344     *                    \n
01345     *     TOOL_IO[0]              (StandardOutputRunState::Handguiding)
01346     *       TOOL_IO[0]
01347     *       TOOL_IO[0]
01348     *
01349     * @param index:  IO       0
01350     *   0
01351     * @param runstate:
01352     *
01353     * @return    0
01354     * AUBO_REQUEST_IGNORE
01355     * AUBO_BUSY
01356     * AUBO_BAD_STATE
01357     * -AUBO_INVL_ARGUMENT
01358     * -AUBO_BAD_STATE
01359     *
01360     * @throws arcs::common_interface::AuboException
01361     *
01362     * @par Python
01363     * setToolDigitalOutputRunstate(self: pyaubo_sdk.IoControl, arg0: int, arg1:
01364     * arcs::common_interface::StandardOutputRunState) -> int
01365     *
01366     * @par Lua
01367     * setToolDigitalOutputRunstate(index: number, runstate: number) -> nil
01368     *
01369     * @par Lua
01370     * setToolDigitalOutputRunstate(0,1)
01371     *
01372     * @par JSON-RPC
01373     * {"jsonrpc":"2.0","method":"rob1.IoControl.setToolDigitalOutputRunstate","params":[0,"None"],"id":1}
01374     *
01375     * @par JSON-RPC
01376     * {"id":1,"jsonrpc":"2.0","result":0}
01377     *
01378     * \endchinese
01379     * \english
01380     * Set the runstate for tool digital output.
01381     *
01382     * @note
01383     * When the output runstate is set to None (StandardOutputRunState::None),
01384     * users can set the digital output value.\n
01385     * When the output runstate is set, users cannot set the digital output
01386     * value, and the controller will set it automatically.\n For example, when
01387     * TOOL_IO[0] is set as output and its runstate is set to indicate
```

```
01388     * hand-guiding (StandardOutputRunState::Handguiding), the robot enters
01389     * hand-guiding mode and TOOL_IO[0] will automatically become high. When the
01390     * robot exits hand-guiding, TOOL_IO[0] will automatically become low.
01391     *
01392     * @param index: Indicates the IO pin, starting from 0.
01393     * For example, 0 means the first pin.
01394     * @param runstate: Output runstate selection
01395     *
01396     * @return Returns 0 on success; error code on failure.
01397     * AUBO_REQUEST_IGNORE
01398     * AUBO_BUSY
01399     * AUBO_BAD_STATE
01400     * -AUBO_INVL_ARGUMENT
01401     * -AUBO_BAD_STATE
01402     *
01403     * @throws arcs::common_interface::AuboException
01404     *
01405     * @par Python function prototype
01406     * setToolDigitalOutputRunstate(self: pyaubo_sdk.IoControl, arg0: int, arg1:
01407     * arcs::common_interface::StandardOutputRunState) -> int
01408     *
01409     * @par Lua function prototype
01410     * setToolDigitalOutputRunstate(index: number, runstate: number) -> nil
01411     *
01412     * @par Lua example
01413     * setToolDigitalOutputRunstate(0,1)
01414     *
01415     * @par JSON-RPC request example
01416     * {"jsonrpc":"2.0","method":"rob1.IoControl.setToolDigitalOutputRunstate","params":[0,"None"],"id":1}
01417     *
01418     * @par JSON-RPC response example
01419     * {"id":1,"jsonrpc":"2.0","result":0}
01420     *
01421     * \endenglish
01422     */
01423    int setToolDigitalOutputRunstate(int index,
01424                                     StandardOutputRunState runstate);
01425
01426    /**
01427     * @ingroup IoControl
01428     * \chinese
01429     *
01430     *
01431     * @param index:   IO       0
01432     *   0
01433     * @param runstate:
01434     *
01435     * @return    0
01436     * AUBO_REQUEST_IGNORE
01437     * AUBO_BUSY
01438     * AUBO_BAD_STATE
01439     * -AUBO_INVL_ARGUMENT
01440     * -AUBO_BAD_STATE
01441     *
01442     * @throws arcs::common_interface::AuboException
01443     *
01444     * @par Python
01445     * setConfigurableDigitalOutputRunstate(self: pyaubo_sdk.IoControl, arg0:
01446     * int, arg1: arcs::common_interface::StandardOutputRunState) -> int
01447     *
01448     * @par Lua
01449     * setConfigurableDigitalOutputRunstate(index: number, runstate: number) ->
01450     * nil
01451     *
01452     * @par Lua
01453     * setConfigurableDigitalOutputRunstate(0,1)
01454     *
01455     * @par JSON-RPC
01456     * {"jsonrpc":"2.0","method":"rob1.IoControl.setConfigurableDigitalOutputRunstate","params":[0,"None"],"id":1}
01457     *
01458     * @par JSON-RPC
01459     * {"id":1,"jsonrpc":"2.0","result":0}
01460     *
01461     * \endchinese
01462     * \english
01463     * Set the runstate for configurable digital output.
01464     *
01465     * @param index: Indicates the IO pin, starting from 0.
01466     * For example, 0 means the first pin.
01467     * @param runstate: Output runstate selection
01468     *
01469     * @return Returns 0 on success; error code on failure.
01470     * AUBO_REQUEST_IGNORE
01471     * AUBO_BUSY
01472     * AUBO_BAD_STATE
01473     * -AUBO_INVL_ARGUMENT
01474     * -AUBO_BAD_STATE
```

```
01475     *
01476     * @throws arcs::common_interface::AuboException
01477     *
01478     * @par Python function prototype
01479     * setConfigurableDigitalOutputRunstate(self: pyaubo_sdk.IoControl, arg0:
01480     * int, arg1: arcs::common_interface::StandardOutputRunState) -> int
01481     *
01482     * @par Lua function prototype
01483     * setConfigurableDigitalOutputRunstate(index: number, runstate: number) ->
01484     * nil
01485     *
01486     * @par Lua example
01487     * setConfigurableDigitalOutputRunstate(0,1)
01488     *
01489     * @par JSON-RPC request example
01490     * {"jsonrpc":"2.0","method":"rob1.IoControl.setConfigurableDigitalOutputRunstate","params":[0,"None"],"id":1}
01491     *
01492     * @par JSON-RPC response example
01493     * {"id":1,"jsonrpc":"2.0","result":0}
01494     *
01495     * \endenglish
01496     */
01497    int setConfigurableDigitalOutputRunstate(int index,
01498                                             StandardOutputRunState runstate);
01499
01500    /**
01501     * @ingroup IoControl
01502     * \chinese
01503     *              / /
01504     *
01505     *                  /
01506     *
01507     *
01508     * @param io_type
01509     *      0=     1=     2=
01510     * @param index          0
01511     * @param runstate
01512     * @param object_name
01513     * @param threshold
01514     *              (rad)   /          (m)
01515     *
01516     * @return       0      0
01517     *
01518     * @par Python
01519     * setDigitalOutputRunstate(self: pyaubo_sdk.IoControl, arg0: int, arg1:
01520     * int, arg2: arcs::common_interface::StandardOutputRunState, arg3: str="",
01521     * arg4: float=0.0) -> int
01522     *
01523     * @par Lua
01524     * setDigitalOutputRunstate(io_type: number, index: number, runstate:
01525     * number, object_name: string="", threshold: number=0.0) -> nil
01526     *
01527     * @par JSON-RPC
01528     *
    {"jsonrpc":"2.0","method":"rob1.IoControl.setDigitalOutputRunstate","params":[1,0,"WaypointArrived","P1",3],"id":1}
01529     *
01530     * @par JSON-RPC
01531     * {"id":1,"jsonrpc":"2.0","result":0}
01532     * \endchinese
01533     * \english
01534     * Set digital output runstate with optional condition checking based on
01535     * waypoint, plane or obstacle thresholds.
01536     *
01537     * Use this API to control a digital output when a specified condition is
01538     * met (for example, waypoint arrival or distance to a plane/obstacle).
01539     * Specify an object type, name and a threshold value to enable conditional
01540     * triggering. When the condition is satisfied, the given digital output
01541     * will be set to the requested runstate.
01542     *
01543     * @param io_type Numeric IO type (e.g. 0=standard, 1=tool, 2=configurable)
01544     * @param index IO pin index starting from 0
01545     * @param runstate The desired output action
01546     * @param object_name Unique identifier of the condition object (optional)
01547     * @param threshold Condition threshold: angle in radians for waypoints,
01548     * or distance in meters for planes/obstacles (optional)
01549     *
01550     * @return Returns 0 on success, non-zero error code on failure.
01551     * \endenglish
01552     */
01553    int setDigitalOutputRunstate(int io_type, int index,
01554                                 StandardOutputRunState runstate,
01555                                 const std::string &object_name,
01556                                 double threshold);
01557
01558    /**
01559     * @ingroup IoControl
01560     * \chinese
```

```
01561     *
01562     *
01563     * @param index:  IO       0
01564     *   0
01565     * @return
01566     *
01567     * @throws arcs::common_interface::AuboException
01568     *
01569     * @par Python
01570     * getStandardDigitalOutputRunstate(self: pyaubo_sdk.IoControl, arg0: int)
01571     * -> arcs::common_interface::StandardOutputRunState
01572     *
01573     * @par Lua
01574     * getStandardDigitalOutputRunstate(index: number) -> number
01575     *
01576     * @par Lua
01577     * num = getStandardDigitalOutputRunstate(0)
01578     *
01579     * @par JSON-RPC
01580     * {"jsonrpc":"2.0","method":"rob1.IoControl.getStandardDigitalOutputRunstate","params":[0],"id":1}
01581     *
01582     * @par JSON-RPC
01583     * {"id":1,"jsonrpc":"2.0","result":"None"}
01584     *
01585     * \endchinese
01586     * \english
01587     * Get the runstate for standard digital output.
01588     *
01589     * @param index: Indicates the IO pin, starting from 0.
01590     * For example, 0 means the first pin.
01591     * @return Output runstate selection
01592     *
01593     * @throws arcs::common_interface::AuboException
01594     *
01595     * @par Python function prototype
01596     * getStandardDigitalOutputRunstate(self: pyaubo_sdk.IoControl, arg0: int)
01597     * -> arcs::common_interface::StandardOutputRunState
01598     *
01599     * @par Lua function prototype
01600     * getStandardDigitalOutputRunstate(index: number) -> number
01601     *
01602     * @par Lua example
01603     * num = getStandardDigitalOutputRunstate(0)
01604     *
01605     * @par JSON-RPC request example
01606     * {"jsonrpc":"2.0","method":"rob1.IoControl.getStandardDigitalOutputRunstate","params":[0],"id":1}
01607     *
01608     * @par JSON-RPC response example
01609     * {"id":1,"jsonrpc":"2.0","result":"None"}
01610     *
01611     * \endenglish
01612     */
01613    StandardOutputRunState getStandardDigitalOutputRunstate(int index);
01614
01615    /**
01616     * @ingroup IoControl
01617     * \chinese
01618     *
01619     *
01620     * @param index:  IO       0
01621     *   0
01622     * @return
01623     *
01624     * @throws arcs::common_interface::AuboException
01625     *
01626     * @par Python
01627     * getToolDigitalOutputRunstate(self: pyaubo_sdk.IoControl, arg0: int) ->
01628     * arcs::common_interface::StandardOutputRunState
01629     *
01630     * @par Lua
01631     * getToolDigitalOutputRunstate(index: number) -> number
01632     *
01633     * @par Lua
01634     * num = getToolDigitalOutputRunstate(0)
01635     *
01636     * @par JSON-RPC
01637     * {"jsonrpc":"2.0","method":"rob1.IoControl.getToolDigitalOutputRunstate","params":[0],"id":1}
01638     *
01639     * @par JSON-RPC
01640     * {"id":1,"jsonrpc":"2.0","result":"None"}
01641     *
01642     * \endchinese
01643     * \english
01644     * Get the runstate for tool digital output.
01645     *
01646     * @param index: Indicates the IO pin, starting from 0.
01647     * For example, 0 means the first pin.
```

```
01648     * @return Output runstate selection
01649     *
01650     * @throws arcs::common_interface::AuboException
01651     *
01652     * @par Python function prototype
01653     * getToolDigitalOutputRunstate(self: pyaubo_sdk.IoControl, arg0: int) ->
01654     * arcs::common_interface::StandardOutputRunState
01655     *
01656     * @par Lua function prototype
01657     * getToolDigitalOutputRunstate(index: number) -> number
01658     *
01659     * @par Lua example
01660     * num = getToolDigitalOutputRunstate(0)
01661     *
01662     * @par JSON-RPC request example
01663     * {"jsonrpc":"2.0","method":"rob1.IoControl.getToolDigitalOutputRunstate","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 getToolDigitalOutputRunstate(int index);
01671
01672    /**
01673     * @ingroup IoControl
01674     * \chinese
01675     *
01676     *
01677     * @param index:   IO       0
01678     *   0
01679     * @return
01680     *
01681     * @throws arcs::common_interface::AuboException
01682     *
01683     * @par Python
01684     * getConfigurableDigitalOutputRunstate(self: pyaubo_sdk.IoControl, arg0:
01685     * int)
01686     * -> arcs::common_interface::StandardOutputRunState
01687     *
01688     * @par Lua
01689     * getConfigurableDigitalOutputRunstate(index: number) -> number
01690     *
01691     * @par Lua
01692     * num = getConfigurableDigitalOutputRunstate(0)
01693     *
01694     * @par JSON-RPC
01695     * {"jsonrpc":"2.0","method":"rob1.IoControl.getConfigurableDigitalOutputRunstate","params":[0],"id":1}
01696     *
01697     * @par JSON-RPC
01698     * {"id":1,"jsonrpc":"2.0","result":"None"}
01699     *
01700     * \endchinese
01701     * \english
01702     * Get the runstate for configurable digital output.
01703     *
01704     * @param index: Indicates the IO pin, starting from 0.
01705     * For example, 0 means the first pin.
01706     * @return Output runstate selection
01707     *
01708     * @throws arcs::common_interface::AuboException
01709     *
01710     * @par Python function prototype
01711     * getConfigurableDigitalOutputRunstate(self: pyaubo_sdk.IoControl, arg0:
01712     * int)
01713     * -> arcs::common_interface::StandardOutputRunState
01714     *
01715     * @par Lua function prototype
01716     * getConfigurableDigitalOutputRunstate(index: number) -> number
01717     *
01718     * @par Lua example
01719     * num = getConfigurableDigitalOutputRunstate(0)
01720     *
01721     * @par JSON-RPC request example
01722     * {"jsonrpc":"2.0","method":"rob1.IoControl.getConfigurableDigitalOutputRunstate","params":[0],"id":1}
01723     *
01724     * @par JSON-RPC response example
01725     * {"id":1,"jsonrpc":"2.0","result":"None"}
01726     *
01727     * \endenglish
01728     */
01729    StandardOutputRunState getConfigurableDigitalOutputRunstate(int index);
01730
01731    /**
01732     * @ingroup IoControl
01733     * \chinese
01734     *
```

```
01735     *
01736     *
01737     *          WaypointArrived   ,
01738     *    setDigitalOutputRunstate
01739     *
01740     * @param io_type
01741     *      0=      1=      2=
01742     * @param index          0
01743     * @return      `StandardOutputRunState‘
01744     *
01745     * @throws arcs::common_interface::AuboException
01746     *
01747     * @par Python
01748     * getDigitalOutputRunstate(self: pyaubo_sdk.IoControl, arg0: int, arg1:
01749     * int) -> arcs::common_interface::StandardOutputRunState
01750     *
01751     * @par Lua
01752     * getDigitalOutputRunstate(io_type: number, index: number) -> number
01753     *
01754     * @par JSON-RPC
01755     * {”jsonrpc”:”2.0”,”method”:”rob1.IoControl.getDigitalOutputRunstate”,”params”:[1,0],”id”:1}
01756     *
01757     * @par JSON-RPC
01758     * {”id”:1,”jsonrpc”:”2.0”,”result”:”None”}
01759     * \endchinese
01760     * \english
01761     * Get the current runstate of a digital output of the specified type.
01762     *
01763     * Use this API to query the runstate selection (for example None,
01764     * Handguiding, WaypointArrived) of different digital output types
01765     * (standard, tool, configurable).
01766     *
01767     * @param io_type Numeric IO type (0=standard, 1=tool, 2=configurable)
01768     * @param index IO pin index starting from 0
01769     * @return Returns the corresponding `StandardOutputRunState‘ enum value
01770     *
01771     * @throws arcs::common_interface::AuboException
01772     * \endenglish
01773     */
01774    StandardOutputRunState getDigitalOutputRunstate(int io_type, int index);
01775
01776    /**
01777     * @ingroup IoControl
01778     * \chinese
01779     *     DIO Cross Connection
01780     *
01781     *    digital/bool    DIO       DO
01782     * DI      actor         result    actor_inverts
01783     * actor      bool             IO
01784     *
01785     * result_type    actor_types           result
01786     *    actor            DO
01787     * - 0      IO result       DO actor       DI
01788     * - 1      IO result       DO actor       DI
01789     * - 2       IO result       DO actor       DI
01790     * - 3    DO       DO             result
01791     *         DO    actor       DO
01792     *
01793     * operator_type             0=FWD 1=NOT 2=AND
01794     * 3=OR FWD/NOT      1    actor AND/OR     2-5    actor
01795     * actor_types actor_pins actor_inverts
01796     *
01797     *
01798     * - Cross Connection       300
01799     * -    id    32
01800     * -    DO       20
01801     * -
01802     * -          result      DO      DO
01803     * -    DO actor               DO
01804     * -    DO pin/index           DO
01805     * -    Cross Connection       DO       runstate
01806     *    set/pulse    DO         DO      Cross Connection
01807     *
01808     * @param id      ID
01809     * @param operator_type
01810     * @param result_type                 DO
01811     * @param result_pin      pin/index    DO       index
01812     * @param actor_types actor           DI     DO
01813     * @param actor_pins actor    pin/index       DO       index
01814     * @param actor_inverts actor
01815     * @return    0
01816     *
01817     * @throws arcs::common_interface::AuboException
01818     *
01819     * @par Python
01820     * addCrossConnection(self: pyaubo_sdk.IoControl, arg0: str, arg1: int,
01821     * arg2: int, arg3: int, arg4: List[int], arg5: List[int],
```

```
01822     * arg6: List[bool]) -> int
01823     *
01824     * @par Lua
01825     * addCrossConnection(id: string, operator_type: number,
01826     * result_type: number, result_pin: number, actor_types: table,
01827     * actor_pins: table, actor_inverts: table) -> number
01828     *
01829     * @par Lua
01830     * addCrossConnection("start_permit", 2, 0, 0, {0, 0}, {3, 4},
01831     *                    {false, true})
01832     *
01833     * @par JSON-RPC
01834     * {"json-
      rpc":"2.0","method":"rob1.IoControl.addCrossConnection","params":["start_permit",2,0,0,[0,0],[3,4],[false,true]],"id":1}
01835     *
01836     * @par JSON-RPC  DO
01837     *       VirtualDO[0] = DI[0] AND NOT DI[1]
01838     * DO[0] = VirtualDO[0] OR DI[2]
01839     * {"jsonrpc":"2.0","method":"rob1.IoControl.clearCrossConnections","params":[],"id":10}
01840     * {"json-
      rpc":"2.0","method":"rob1.IoControl.addCrossConnection","params":["v0_di0_and_not_di1",2,3,0,[0,0],[0,1],[false,true]],"id":11}
01841     * {"json-
      rpc":"2.0","method":"rob1.IoControl.addCrossConnection","params":["v0_or_di2_to_do0",3,0,0,[3,0],[0,2],[false,false]],"id":12}
01842     *
01843     * @par JSON-RPC
01844     * {"id":1,"jsonrpc":"2.0","result":0}
01845     *
01846     * \endchinese
01847     * \english
01848     * Add a DIO Cross Connection logic rule.
01849     *
01850     * Supports digital/bool DIO signals and virtual DO signals. The rule takes
01851     * effect immediately after it is added successfully. DI signals may only be
01852     * read as actors and cannot be written as results. actor_inverts tells the
01853     * controller whether to invert each actor's raw bool value before
01854     * evaluating the rule; it does not change the hardware IO direction.
01855     *
01856     * result_type and actor_types use the same numeric type set. As a result,
01857     * the value denotes the output side; as an actor, it denotes the input side
01858     * or a readable virtual DO:
01859     * - 0: Standard digital IO. result means standard DO, actor means standard
01860     *      DI.
01861     * - 1: Tool digital IO. result means tool DO, actor means tool DI.
01862     * - 2: Configurable digital IO. result means configurable DO, actor means
01863     *      configurable DI.
01864     * - 3: Virtual DO. It does not occupy a real DO pin and can be used as an
01865     *      intermediate result for cascaded rules. As a result it writes the
01866     *      virtual DO; as an actor it reads the virtual DO.
01867     *
01868     * operator_type is interpreted by the controller implementation; the
01869     * recommended mapping is 0=FWD, 1=NOT, 2=AND, 3=OR. FWD/NOT require exactly
01870     * one actor, and AND/OR require 2 to 5 actors. actor_types, actor_pins, and
01871     * actor_inverts must have the same length.
01872     *
01873     * Limits:
01874     * - Up to 300 Cross Connection rules are allowed.
01875     * - A rule id can contain up to 32 characters and must be unique.
01876     * - Virtual DO cascade depth is limited to 20.
01877     * - Cyclic dependencies are not allowed.
01878     * - Multiple rules cannot write the same result, including real DO and
01879     *   virtual DO results.
01880     * - A virtual DO actor must be produced by an earlier rule; reading an
01881     *   undefined virtual DO should fail.
01882     * - A virtual DO pin/index only needs to be non-negative and does not
01883     *   occupy a real DO pin.
01884     * - A real DO written by Cross Connection should not also have a runstate
01885     *   configured. Later manual set/pulse calls to that DO should be rejected
01886     *   with a message that names the owning Cross Connection.
01887     *
01888     * @param id Unique rule ID
01889     * @param operator_type Logic operator type
01890     * @param result_type Result signal type; must be writable output type or
01891     * virtual DO
01892     * @param result_pin Result signal pin/index; virtual DO uses virtual signal
01893     * index
01894     * @param actor_types Actor signal type list; may read DI or virtual DO
01895     * @param actor_pins Actor signal pin/index list; virtual DO uses virtual
01896     * signal index
01897     * @param actor_inverts Actor polarity list
01898     * @return Returns 0 on success, non-zero error code on failure.
01899     *
01900     * @throws arcs::common_interface::AuboException
01901     *
01902     * @par Python function prototype
01903     * addCrossConnection(self: pyaubo_sdk.IoControl, arg0: str, arg1: int,
01904     * arg2: int, arg3: int, arg4: List[int], arg5: List[int],
01905     * arg6: List[bool]) -> int
```

```
01906     *
01907     * @par Lua function prototype
01908     * addCrossConnection(id: string, operator_type: number,
01909     * result_type: number, result_pin: number, actor_types: table,
01910     * actor_pins: table, actor_inverts: table) -> number
01911     *
01912     * @par Lua example
01913     * addCrossConnection("start_permit", 2, 0, 0, {0, 0}, {3, 4},
01914     *                    {false, true})
01915     *
01916     * @par JSON-RPC request example
01917     * {"json-
      rpc":"2.0","method":"rob1.IoControl.addCrossConnection","params":["start_permit",2,0,0,[0,0],[3,4],[false,true]],"id":1}
01918     *
01919     * @par JSON-RPC virtual DO cascade example
01920     * First generate the intermediate signal VirtualDO[0] = DI[0] AND NOT
01921     * DI[1], then drive DO[0] = VirtualDO[0] OR DI[2]:
01922     * {"jsonrpc":"2.0","method":"rob1.IoControl.clearCrossConnections","params":[],"id":10}
01923     * {"json-
      rpc":"2.0","method":"rob1.IoControl.addCrossConnection","params":["v0_di0_and_not_di1",2,3,0,[0,0],[0,1],[false,true]],"id":11}
01924     * {"json-
      rpc":"2.0","method":"rob1.IoControl.addCrossConnection","params":["v0_or_di2_to_do0",3,0,0,[3,0],[0,2],[false,false]],"id":12}
01925     *
01926     * @par JSON-RPC response example
01927     * {"id":1,"jsonrpc":"2.0","result":0}
01928     *
01929     * \endenglish
01930     */
01931    int addCrossConnection(const std::string &id, int operator_type,
01932                           int result_type, int result_pin,
01933                           const std::vector<int> &actor_types,
01934                           const std::vector<int> &actor_pins,
01935                           const std::vector<bool> &actor_inverts);
01936
01937    /**
01938     * @ingroup IoControl
01939     * \chinese
01940     *     DIO Cross Connection
01941     *
01942     * @param id     ID
01943     * @return    0
01944     *
01945     * @throws arcs::common_interface::AuboException
01946     *
01947     * @par Python
01948     * deleteCrossConnection(self: pyaubo_sdk.IoControl, arg0: str) -> int
01949     *
01950     * @par Lua
01951     * deleteCrossConnection(id: string) -> number
01952     *
01953     * @par JSON-RPC
01954     * {"jsonrpc":"2.0","method":"rob1.IoControl.deleteCrossConnection","params":["start_permit"],"id":1}
01955     *
01956     * @par JSON-RPC
01957     * {"id":1,"jsonrpc":"2.0","result":0}
01958     *
01959     * \endchinese
01960     * \english
01961     * Delete a DIO Cross Connection logic rule.
01962     *
01963     * @param id Unique rule ID
01964     * @return Returns 0 on success, non-zero error code on failure.
01965     *
01966     * @throws arcs::common_interface::AuboException
01967     *
01968     * @par Python function prototype
01969     * deleteCrossConnection(self: pyaubo_sdk.IoControl, arg0: str) -> int
01970     *
01971     * @par Lua function prototype
01972     * deleteCrossConnection(id: string) -> number
01973     *
01974     * @par JSON-RPC request example
01975     * {"jsonrpc":"2.0","method":"rob1.IoControl.deleteCrossConnection","params":["start_permit"],"id":1}
01976     *
01977     * @par JSON-RPC response example
01978     * {"id":1,"jsonrpc":"2.0","result":0}
01979     *
01980     * \endenglish
01981     */
01982    int deleteCrossConnection(const std::string &id);
01983
01984    /**
01985     * @ingroup IoControl
01986     * \chinese
01987     *     DIO Cross Connection
01988     *
01989     *
```

```
01990     *
01991     * @return    0
01992     *
01993     * @throws arcs::common_interface::AuboException
01994     *
01995     * @par Python
01996     * clearCrossConnections(self: pyaubo_sdk.IoControl) -> int
01997     *
01998     * @par Lua
01999     * clearCrossConnections() -> number
02000     *
02001     * @par JSON-RPC
02002     * {"jsonrpc":"2.0","method":"rob1.IoControl.clearCrossConnections","params":[],"id":1}
02003     *
02004     * @par JSON-RPC
02005     * {"id":1,"jsonrpc":"2.0","result":0}
02006     *
02007     * \endchinese
02008     * \english
02009     * Clear all DIO Cross Connection logic rules.
02010     *
02011     * This only clears rules. It does not clear or reset real digital output
02012     * values.
02013     *
02014     * @return Returns 0 on success, non-zero error code on failure.
02015     *
02016     * @throws arcs::common_interface::AuboException
02017     *
02018     * @par Python function prototype
02019     * clearCrossConnections(self: pyaubo_sdk.IoControl) -> int
02020     *
02021     * @par Lua function prototype
02022     * clearCrossConnections() -> number
02023     *
02024     * @par JSON-RPC request example
02025     * {"jsonrpc":"2.0","method":"rob1.IoControl.clearCrossConnections","params":[],"id":1}
02026     *
02027     * @par JSON-RPC response example
02028     * {"id":1,"jsonrpc":"2.0","result":0}
02029     *
02030     * \endenglish
02031     */
02032    int clearCrossConnections();
02033
02034    /**
02035     * @ingroup IoControl
02036     * \chinese
02037     *
02038     *
02039     * @note
02040     *        (StandardOutputRunState::None)
02041     *         \n
02042     *                        \n
02043     *     AO0             (StandardOutputRunState::Handguiding)
02044     *        AO0
02045     *        AO0     0
02046     *
02047     * @param index:   IO       0
02048     *   0
02049     * @param runstate:
02050     * @return    0
02051     * AUBO_REQUEST_IGNORE
02052     * AUBO_BUSY
02053     * AUBO_BAD_STATE
02054     * -AUBO_INVL_ARGUMENT
02055     * -AUBO_BAD_STATE
02056     *
02057     * @throws arcs::common_interface::AuboException
02058     *
02059     * @par Python
02060     * setStandardAnalogOutputRunstate(self: pyaubo_sdk.IoControl, arg0: int,
02061     * arg1: arcs::common_interface::StandardOutputRunState) -> int
02062     *
02063     * @par Lua
02064     * setStandardAnalogOutputRunstate(index: number, runstate: number) -> nil
02065     *
02066     * @par Lua
02067     * setStandardAnalogOutputRunstate(0,6)
02068     *
02069     * @par JSON-RPC
02070     * {"jsonrpc":"2.0","method":"rob1.IoControl.setStandardAnalogOutputRunstate","params":[0,"None"],"id":1}
02071     *
02072     * @par JSON-RPC
02073     * {"id":1,"jsonrpc":"2.0","result":0}
02074     *
02075     * \endchinese
02076     * \english
```

```
02077     * Set the runstate for standard analog output.
02078     *
02079     * @note
02080     * When the output runstate is set to None (StandardOutputRunState::None),
02081     * users can set the analog output value.\n
02082     * When the output runstate is set, users cannot set the analog output
02083     * value, and the controller will set it automatically.\n For example, when
02084     * AO0's output runstate is set to indicate hand-guiding
02085     * (StandardOutputRunState::Handguiding), the robot enters hand-guiding mode
02086     * and AO0 will automatically become the maximum value. When the robot exits
02087     * hand-guiding, AO0 will automatically become 0.
02088     *
02089     * @param index: Indicates the IO pin, starting from 0.
02090     * For example, 0 means the first pin.
02091     * @param runstate: Output runstate selection
02092     * @return Returns 0 on success; error code on failure.
02093     * AUBO_REQUEST_IGNORE
02094     * AUBO_BUSY
02095     * AUBO_BAD_STATE
02096     * -AUBO_INVL_ARGUMENT
02097     * -AUBO_BAD_STATE
02098     *
02099     * @throws arcs::common_interface::AuboException
02100     *
02101     * @par Python function prototype
02102     * setStandardAnalogOutputRunstate(self: pyaubo_sdk.IoControl, arg0: int,
02103     * arg1: arcs::common_interface::StandardOutputRunState) -> int
02104     *
02105     * @par Lua function prototype
02106     * setStandardAnalogOutputRunstate(index: number, runstate: number) -> nil
02107     *
02108     * @par Lua example
02109     * setStandardAnalogOutputRunstate(0,6)
02110     *
02111     * @par JSON-RPC request example
02112     * {"jsonrpc":"2.0","method":"rob1.IoControl.setStandardAnalogOutputRunstate","params":[0,"None"],"id":1}
02113     *
02114     * @par JSON-RPC response example
02115     * {"id":1,"jsonrpc":"2.0","result":0}
02116     *
02117     * \endenglish
02118     */
02119    int setStandardAnalogOutputRunstate(int index,
02120                                        StandardOutputRunState runstate);
02121
02122    /**
02123     * @ingroup IoControl
02124     * \chinese
02125     *
02126     *
02127     * @param index:   IO       0
02128     *   0
02129     * @param runstate:
02130     * @return    0
02131     * AUBO_REQUEST_IGNORE
02132     * AUBO_BUSY
02133     * AUBO_BAD_STATE
02134     * -AUBO_INVL_ARGUMENT
02135     * -AUBO_BAD_STATE
02136     *
02137     * @throws arcs::common_interface::AuboException
02138     *
02139     * @par Python
02140     * setToolAnalogOutputRunstate(self: pyaubo_sdk.IoControl, arg0: int, arg1:
02141     * arcs::common_interface::StandardOutputRunState) -> int
02142     *
02143     * @par Lua
02144     * setToolAnalogOutputRunstate(index: number, runstate: number) -> nil
02145     *
02146     * @par Lua
02147     * setToolAnalogOutputRunstate(0,1)
02148     *
02149     * @par JSON-RPC
02150     * {"jsonrpc":"2.0","method":"rob1.IoControl.setToolAnalogOutputRunstate","params":[0,"None"],"id":1}
02151     *
02152     * @par JSON-RPC
02153     * {"id":1,"jsonrpc":"2.0","result":0}
02154     *
02155     * \endchinese
02156     * \english
02157     * Set the runstate for tool analog output.
02158     *
02159     * @param index: Indicates the IO pin, starting from 0.
02160     * For example, 0 means the first pin.
02161     * @param runstate: Output runstate selection
02162     * @return Returns 0 on success; error code on failure.
02163     * AUBO_REQUEST_IGNORE
```

```
02164     * AUBO_BUSY
02165     * AUBO_BAD_STATE
02166     * -AUBO_INVL_ARGUMENT
02167     * -AUBO_BAD_STATE
02168     *
02169     * @throws arcs::common_interface::AuboException
02170     *
02171     * @par Python function prototype
02172     * setToolAnalogOutputRunstate(self: pyaubo_sdk.IoControl, arg0: int, arg1:
02173     * arcs::common_interface::StandardOutputRunState) -> int
02174     *
02175     * @par Lua function prototype
02176     * setToolAnalogOutputRunstate(index: number, runstate: number) -> nil
02177     *
02178     * @par Lua example
02179     * setToolAnalogOutputRunstate(0,1)
02180     *
02181     * @par JSON-RPC request example
02182     * {"jsonrpc":"2.0","method":"rob1.IoControl.setToolAnalogOutputRunstate","params":[0,"None"],"id":1}
02183     *
02184     * @par JSON-RPC response example
02185     * {"id":1,"jsonrpc":"2.0","result":0}
02186     *
02187     * \endenglish
02188     */
02189    int setToolAnalogOutputRunstate(int index, StandardOutputRunState runstate);
02190
02191    /**
02192     * @ingroup IoControl
02193     * \chinese
02194     *
02195     *
02196     * @param index:   IO       0
02197     *   0
02198     * @return
02199     *
02200     * @throws arcs::common_interface::AuboException
02201     *
02202     * @par Python
02203     * getStandardAnalogOutputRunstate(self: pyaubo_sdk.IoControl, arg0: int) ->
02204     * arcs::common_interface::StandardOutputRunState
02205     *
02206     * @par Lua
02207     * getStandardAnalogOutputRunstate(index: number) -> number
02208     *
02209     * @par Lua
02210     * num = getStandardAnalogOutputRunstate(0)
02211     *
02212     * @par JSON-RPC
02213     * {"jsonrpc":"2.0","method":"rob1.IoControl.getStandardAnalogOutputRunstate","params":[0],"id":1}
02214     *
02215     * @par JSON-RPC
02216     * {"id":1,"jsonrpc":"2.0","result":"None"}
02217     *
02218     * \endchinese
02219     * \english
02220     * Get the runstate for standard analog output.
02221     *
02222     * @param index: Indicates the IO pin, starting from 0.
02223     * For example, 0 means the first pin.
02224     * @return Standard analog output runstate selection
02225     *
02226     * @throws arcs::common_interface::AuboException
02227     *
02228     * @par Python function prototype
02229     * getStandardAnalogOutputRunstate(self: pyaubo_sdk.IoControl, arg0: int) ->
02230     * arcs::common_interface::StandardOutputRunState
02231     *
02232     * @par Lua function prototype
02233     * getStandardAnalogOutputRunstate(index: number) -> number
02234     *
02235     * @par Lua example
02236     * num = getStandardAnalogOutputRunstate(0)
02237     *
02238     * @par JSON-RPC request example
02239     * {"jsonrpc":"2.0","method":"rob1.IoControl.getStandardAnalogOutputRunstate","params":[0],"id":1}
02240     *
02241     * @par JSON-RPC response example
02242     * {"id":1,"jsonrpc":"2.0","result":"None"}
02243     *
02244     * \endenglish
02245     */
02246    StandardOutputRunState getStandardAnalogOutputRunstate(int index);
02247
02248    /**
02249     * @ingroup IoControl
02250     * \chinese
```

```
02251     *
02252     *
02253     * @param index:   IO       0
02254     *   0
02255     * @return
02256     *
02257     * @throws arcs::common_interface::AuboException
02258     *
02259     * @par Python
02260     * getToolAnalogOutputRunstate(self: pyaubo_sdk.IoControl, arg0: int) ->
02261     * arcs::common_interface::StandardOutputRunState
02262     *
02263     * @par Lua
02264     * getToolAnalogOutputRunstate(index: number) -> number
02265     *
02266     * @par Lua
02267     * num = getToolAnalogOutputRunstate(0)
02268     *
02269     * @par JSON-RPC
02270     * {"jsonrpc":"2.0","method":"rob1.IoControl.getToolAnalogOutputRunstate","params":[0],"id":1}
02271     *
02272     * @par JSON-RPC
02273     * {"id":1,"jsonrpc":"2.0","result":"None"}
02274     *
02275     * \endchinese
02276     * \english
02277     * Get the runstate for tool analog output.
02278     *
02279     * @param index: Indicates the IO pin, starting from 0.
02280     * For example, 0 means the first pin.
02281     * @return Tool analog output runstate selection
02282     *
02283     * @throws arcs::common_interface::AuboException
02284     *
02285     * @par Python function prototype
02286     * getToolAnalogOutputRunstate(self: pyaubo_sdk.IoControl, arg0: int) ->
02287     * arcs::common_interface::StandardOutputRunState
02288     *
02289     * @par Lua function prototype
02290     * getToolAnalogOutputRunstate(index: number) -> number
02291     *
02292     * @par Lua example
02293     * num = getToolAnalogOutputRunstate(0)
02294     *
02295     * @par JSON-RPC request example
02296     * {"jsonrpc":"2.0","method":"rob1.IoControl.getToolAnalogOutputRunstate","params":[0],"id":1}
02297     *
02298     * @par JSON-RPC response example
02299     * {"id":1,"jsonrpc":"2.0","result":"None"}
02300     *
02301     * \endenglish
02302     */
02303    StandardOutputRunState getToolAnalogOutputRunstate(int index);
02304
02305    /**
02306     * @ingroup IoControl
02307     * \chinese
02308     *
02309     *
02310     * @return    0
02311     * AUBO_BUSY
02312     * AUBO_BAD_STATE
02313     * -AUBO_BAD_STATE
02314     *
02315     * @throws arcs::common_interface::AuboException
02316     *
02317     * @par Python
02318     * setDigitalOutputAfterEStopDefault(self: pyaubo_sdk.IoControl) -> int
02319     *
02320     * @par Lua
02321     * setDigitalOutputAfterEStopDefault() -> nil
02322     *
02323     * @par Lua
02324     * setDigitalOutputAfterEStopDefault()
02325     *
02326     * @par JSON-RPC
02327     * {"jsonrpc":"2.0","method":"rob1.IoControl.setDigitalOutputAfterEStopDefault","params":[],"id":1}
02328     *
02329     * @par JSON-RPC
02330     * {"id":1,"jsonrpc":"2.0","result":0}
02331     *
02332     * \endchinese
02333     * \english
02334     * Set all digital output states after emergency stop to default(no change)
02335     *
02336     * @return Returns 0 on success; error code on failure.
02337     * AUBO_BUSY
```

```
02338     * AUBO_BAD_STATE
02339     * -AUBO_BAD_STATE
02340     *
02341     * @throws arcs::common_interface::AuboException
02342     *
02343     * @par Python function prototype
02344     * setDigitalOutputAfterEStopDefault(self: pyaubo_sdk.IoControl) -> int
02345     *
02346     * @par Lua function prototype
02347     * setDigitalOutputAfterEStopDefault() -> nil
02348     *
02349     * @par Lua example
02350     * setDigitalOutputAfterEStopDefault()
02351     *
02352     * @par JSON-RPC request example
02353     * {"jsonrpc":"2.0","method":"rob1.IoControl.setDigitalOutputAfterEStopDefault","params":[],"id":1}
02354     *
02355     * @par JSON-RPC response example
02356     * {"id":1,"jsonrpc":"2.0","result":0}
02357     *
02358     * \endenglish
02359     */
02360    int setDigitalOutputAfterEStopDefault();
02361
02362    /**
02363     * @ingroup IoControl
02364     * \chinese
02365     *
02366     *
02367     * @return    0
02368     * AUBO_BUSY
02369     * AUBO_BAD_STATE
02370     * -AUBO_BAD_STATE
02371     *
02372     * @throws arcs::common_interface::AuboException
02373     *
02374     * @par Python
02375     * setAnalogOutputAfterEStopDefault(self: pyaubo_sdk.IoControl) -> int
02376     *
02377     * @par Lua
02378     * setAnalogOutputAfterEStopDefault() -> nil
02379     *
02380     * @par Lua
02381     * setAnalogOutputAfterEStopDefault()
02382     *
02383     * @par JSON-RPC
02384     * {"jsonrpc":"2.0","method":"rob1.IoControl.setAnalogOutputAfterEStopDefault","params":[],"id":1}
02385     *
02386     * @par JSON-RPC
02387     * {"id":1,"jsonrpc":"2.0","result":0}
02388     *
02389     * \endchinese
02390     * \english
02391     * Set all analog output states after emergency stop to default(no change)
02392     *
02393     * @return Returns 0 on success; error code on failure.
02394     * AUBO_BUSY
02395     * AUBO_BAD_STATE
02396     * -AUBO_BAD_STATE
02397     *
02398     * @throws arcs::common_interface::AuboException
02399     *
02400     * @par Python function prototype
02401     * setAnalogOutputAfterEStopDefault(self: pyaubo_sdk.IoControl) -> int
02402     *
02403     * @par Lua function prototype
02404     * setAnalogOutputAfterEStopDefault() -> nil
02405     *
02406     * @par Lua example
02407     * setAnalogOutputAfterEStopDefault()
02408     *
02409     * @par JSON-RPC request example
02410     * {"jsonrpc":"2.0","method":"rob1.IoControl.setAnalogOutputAfterEStopDefault","params":[],"id":1}
02411     *
02412     * @par JSON-RPC response example
02413     * {"id":1,"jsonrpc":"2.0","result":0}
02414     *
02415     * \endenglish
02416     */
02417    int setAnalogOutputAfterEStopDefault();
02418
02419    /**
02420     * @ingroup IoControl
02421     * \chinese
02422     *
02423     *
02424     * @param index:    IO
```

```
02425     * @param value:
02426     * @return    0
02427     * AUBO_REQUEST_IGNORE
02428     * AUBO_BUSY
02429     * AUBO_BAD_STATE
02430     * -AUBO_INVL_ARGUMENT
02431     * -AUBO_BAD_STATE
02432     *
02433     * @throws arcs::common_interface::AuboException
02434     *
02435     * @par Python
02436     * setStandardDigitalOutputAfterEStop(self: pyaubo_sdk.IoControl, arg0: int,
02437     * arg1: bool) -> int
02438     *
02439     * @par Lua
02440     * setStandardDigitalOutputAfterEStop(index: number, value: boolean) -> nil
02441     *
02442     * @par Lua
02443     * setStandardDigitalOutputAfterEStop(0,true)
02444     *
02445     * @par JSON-RPC
02446     * {"jsonrpc":"2.0","method":"rob1.IoControl.setStandardDigitalOutputAfterEStop","params":[0,true],"id":1}
02447     *
02448     * @par JSON-RPC
02449     * {"id":1,"jsonrpc":"2.0","result":0}
02450     *
02451     * \endchinese
02452     * \english
02453     * Set the value of a standard digital output after emergency stop
02454     *
02455     * @param index: Indicates the IO pin.
02456     * @param value: Output value.
02457     * @return Returns 0 on success; error code on failure.
02458     * AUBO_REQUEST_IGNORE
02459     * AUBO_BUSY
02460     * AUBO_BAD_STATE
02461     * -AUBO_INVL_ARGUMENT
02462     * -AUBO_BAD_STATE
02463     *
02464     * @throws arcs::common_interface::AuboException
02465     *
02466     * @par Python function prototype
02467     * setStandardDigitalOutputAfterEStop(self: pyaubo_sdk.IoControl, arg0: int,
02468     * arg1: bool) -> int
02469     *
02470     * @par Lua function prototype
02471     * setStandardDigitalOutputAfterEStop(index: number, value: boolean) -> nil
02472     *
02473     * @par Lua example
02474     * setStandardDigitalOutputAfterEStop(0,true)
02475     *
02476     * @par JSON-RPC request example
02477     * {"jsonrpc":"2.0","method":"rob1.IoControl.setStandardDigitalOutputAfterEStop","params":[0,true],"id":1}
02478     *
02479     * @par JSON-RPC response example
02480     * {"id":1,"jsonrpc":"2.0","result":0}
02481     *
02482     * \endenglish
02483     */
02484    int setStandardDigitalOutputAfterEStop(int index, bool value);
02485
02486    /**
02487     * @ingroup IoControl
02488     * \chinese
02489     *
02490     *
02491     * @param index:    IO
02492     * @param value:
02493     * @return    0
02494     * AUBO_REQUEST_IGNORE
02495     * AUBO_BUSY
02496     * AUBO_BAD_STATE
02497     * -AUBO_INVL_ARGUMENT
02498     * -AUBO_BAD_STATE
02499     *
02500     * @throws arcs::common_interface::AuboException
02501     *
02502     * @par Python
02503     * setConfigurableDigitalOutputAfterEStop(self: pyaubo_sdk.IoControl, arg0:
02504     * int, arg1: bool) -> int
02505     *
02506     * @par Lua
02507     * setConfigurableDigitalOutputAfterEStop(index: number, value: boolean) ->
02508     * nil
02509     *
02510     * @par Lua
02511     * setConfigurableDigitalOutputAfterEStop(0,true)
```

```
02512     *
02513     * @par JSON-RPC
02514     * {"jsonrpc":"2.0","method":"rob1.IoControl.setConfigurableDigitalOutputAfterEStop","params":[0,true],"id":1}
02515     *
02516     * @par JSON-RPC
02517     * {"id":1,"jsonrpc":"2.0","result":0}
02518     *
02519     * \endchinese
02520     * \english
02521     * Set the value of a configurable digital output after emergency stop
02522     *
02523     * @param index: Indicates the IO pin.
02524     * @param value: Output value.
02525     * @return Returns 0 on success; error code on failure.
02526     * AUBO_REQUEST_IGNORE
02527     * AUBO_BUSY
02528     * AUBO_BAD_STATE
02529     * -AUBO_INVL_ARGUMENT
02530     * -AUBO_BAD_STATE
02531     *
02532     * @throws arcs::common_interface::AuboException
02533     *
02534     * @par Python function prototype
02535     * setConfigurableDigitalOutputAfterEStop(self: pyaubo_sdk.IoControl, arg0:
02536     * int, arg1: bool) -> int
02537     *
02538     * @par Lua function prototype
02539     * setConfigurableDigitalOutputAfterEStop(index: number, value: boolean) ->
02540     * nil
02541     *
02542     * @par Lua example
02543     * setConfigurableDigitalOutputAfterEStop(0,true)
02544     *
02545     * @par JSON-RPC request example
02546     * {"jsonrpc":"2.0","method":"rob1.IoControl.setConfigurableDigitalOutputAfterEStop","params":[0,true],"id":1}
02547     *
02548     * @par JSON-RPC response example
02549     * {"id":1,"jsonrpc":"2.0","result":0}
02550     *
02551     * \endenglish
02552     */
02553    int setConfigurableDigitalOutputAfterEStop(int index, bool value);
02554
02555    /**
02556     * @ingroup IoControl
02557     * \chinese
02558     *
02559     *
02560     * @param index:   IO       0
02561     *   0
02562     * @param value:
02563     *
02564     * @return    0
02565     * AUBO_REQUEST_IGNORE
02566     * AUBO_BUSY
02567     * AUBO_BAD_STATE
02568     * -AUBO_INVL_ARGUMENT
02569     * -AUBO_BAD_STATE
02570     *
02571     * @throws arcs::common_interface::AuboException
02572     *
02573     * @par Python
02574     * setStandardAnalogOutputAfterEStop(self: pyaubo_sdk.IoControl, arg0: int,
02575     * arg1: float) -> int
02576     *
02577     * @par Lua
02578     * setStandardAnalogOutputAfterEStop(index: number, value: number) -> nil
02579     *
02580     * @par Lua
02581     * setStandardAnalogOutputAfterEStop(0,5.4)
02582     *
02583     * \endchinese
02584     * \english
02585     * Set the value of standard analog output after emergency stop
02586     *
02587     * @param index: Indicates the IO pin, starting from 0.
02588     * For example, 0 means the first pin.
02589     * @param value: Output value.
02590     *
02591     * @return Returns 0 on success; error code on failure.
02592     * AUBO_REQUEST_IGNORE
02593     * AUBO_BUSY
02594     * AUBO_BAD_STATE
02595     * -AUBO_INVL_ARGUMENT
02596     * -AUBO_BAD_STATE
02597     *
02598     * @throws arcs::common_interface::AuboException
```

```
02599     *
02600     * @par Python function prototype
02601     * setStandardAnalogOutputAfterEStop(self: pyaubo_sdk.IoControl, arg0: int,
02602     * arg1: float) -> int
02603     *
02604     * @par Lua function prototype
02605     * setStandardAnalogOutputAfterEStop(index: number, value: number) -> nil
02606     *
02607     * @par Lua example
02608     * setStandardAnalogOutputAfterEStop(0,5.4)
02609     *
02610     * \endenglish
02611     */
02612    int setStandardAnalogOutputAfterEStop(int index, double value);
02613
02614    /**
02615     * @ingroup IoControl
02616     * \chinese
02617     *
02618     *
02619     * @param index:   IO       0
02620     *   0
02621     * @param domain:
02622     *
02623     * @return    0
02624     * AUBO_REQUEST_IGNORE
02625     * AUBO_BUSY
02626     * AUBO_BAD_STATE
02627     * -AUBO_INVL_ARGUMENT
02628     * -AUBO_BAD_STATE
02629     *
02630     * @throws arcs::common_interface::AuboException
02631     *
02632     * @par Python
02633     * setStandardAnalogInputDomain(self: pyaubo_sdk.IoControl, arg0: int, arg1:
02634     * int) -> int
02635     *
02636     * @par Lua
02637     * setStandardAnalogInputDomain(index: number, domain: number) -> nil
02638     *
02639     * @par Lua
02640     * setStandardAnalogInputDomain(0,1)
02641     *
02642     * @par JSON-RPC
02643     * {"jsonrpc":"2.0","method":"rob1.IoControl.setStandardAnalogInputDomain","params":[0,8],"id":1}
02644     *
02645     * @par JSON-RPC
02646     * {"id":1,"jsonrpc":"2.0","result":0}
02647     *
02648     * \endchinese
02649     * \english
02650     * Set the range of standard analog input.
02651     *
02652     * @param index: Indicates the IO pin, starting from 0.
02653     * For example, 0 means the first pin.
02654     * @param domain: Input range
02655     *
02656     * @return Returns 0 on success; error code on failure.
02657     * AUBO_REQUEST_IGNORE
02658     * AUBO_BUSY
02659     * AUBO_BAD_STATE
02660     * -AUBO_INVL_ARGUMENT
02661     * -AUBO_BAD_STATE
02662     *
02663     * @throws arcs::common_interface::AuboException
02664     *
02665     * @par Python function prototype
02666     * setStandardAnalogInputDomain(self: pyaubo_sdk.IoControl, arg0: int, arg1:
02667     * int) -> int
02668     *
02669     * @par Lua function prototype
02670     * setStandardAnalogInputDomain(index: number, domain: number) -> nil
02671     *
02672     * @par Lua example
02673     * setStandardAnalogInputDomain(0,1)
02674     *
02675     * @par JSON-RPC request example
02676     * {"jsonrpc":"2.0","method":"rob1.IoControl.setStandardAnalogInputDomain","params":[0,8],"id":1}
02677     *
02678     * @par JSON-RPC response example
02679     * {"id":1,"jsonrpc":"2.0","result":0}
02680     *
02681     * \endenglish
02682     */
02683    int setStandardAnalogInputDomain(int index, int domain);
02684
02685    /**
```

```
02686     * @ingroup IoControl
02687     * \chinese
02688     *
02689     *
02690     * @param index:   IO       0
02691     *   0
02692     * @param domain:
02693     * @return    0
02694     * AUBO_REQUEST_IGNORE
02695     * AUBO_BUSY
02696     * AUBO_BAD_STATE
02697     * -AUBO_INVL_ARGUMENT
02698     * -AUBO_BAD_STATE
02699     *
02700     * @throws arcs::common_interface::AuboException
02701     *
02702     * @par Python
02703     * setToolAnalogInputDomain(self: pyaubo_sdk.IoControl, arg0: int, arg1:
02704     * int) -> int
02705     *
02706     * @par Lua
02707     * setToolAnalogInputDomain(index: number, domain: number) -> nil
02708     *
02709     * @par Lua
02710     * setToolAnalogInputDomain(0,1)
02711     *
02712     * @par JSON-RPC
02713     * {"jsonrpc":"2.0","method":"rob1.IoControl.setToolAnalogInputDomain","params":[0,8],"id":1}
02714     *
02715     * @par JSON-RPC
02716     * {"id":1,"jsonrpc":"2.0","result":0}
02717     *
02718     * \endchinese
02719     * \english
02720     * Set the range of tool analog input.
02721     *
02722     * @param index: Indicates the IO pin, starting from 0.
02723     * For example, 0 means the first pin.
02724     * @param domain: Input range
02725     * @return Returns 0 on success; error code on failure.
02726     * AUBO_REQUEST_IGNORE
02727     * AUBO_BUSY
02728     * AUBO_BAD_STATE
02729     * -AUBO_INVL_ARGUMENT
02730     * -AUBO_BAD_STATE
02731     *
02732     * @throws arcs::common_interface::AuboException
02733     *
02734     * @par Python function prototype
02735     * setToolAnalogInputDomain(self: pyaubo_sdk.IoControl, arg0: int, arg1:
02736     * int) -> int
02737     *
02738     * @par Lua function prototype
02739     * setToolAnalogInputDomain(index: number, domain: number) -> nil
02740     *
02741     * @par Lua example
02742     * setToolAnalogInputDomain(0,1)
02743     *
02744     * @par JSON-RPC request example
02745     * {"jsonrpc":"2.0","method":"rob1.IoControl.setToolAnalogInputDomain","params":[0,8],"id":1}
02746     *
02747     * @par JSON-RPC response example
02748     * {"id":1,"jsonrpc":"2.0","result":0}
02749     *
02750     * \endenglish
02751     */
02752    int setToolAnalogInputDomain(int index, int domain);
02753
02754    /**
02755     * @ingroup IoControl
02756     * \chinese
02757     *
02758     *
02759     * @param index:   IO       0
02760     *   0
02761     * @return
02762     *
02763     * @throws arcs::common_interface::AuboException
02764     *
02765     * @par Python
02766     * getStandardAnalogInputDomain(self: pyaubo_sdk.IoControl, arg0: int) ->
02767     * int
02768     *
02769     * @par Lua
02770     * getStandardAnalogInputDomain(index: number) -> number
02771     *
02772     * @par Lua
```

```
02773     * num = getStandardAnalogInputDomain(0)
02774     *
02775     * @par JSON-RPC
02776     * {”jsonrpc”:”2.0”,”method”:”rob1.IoControl.getStandardAnalogInputDomain”,”params”:[0],”id”:1}
02777     *
02778     * @par JSON-RPC
02779     * {”id”:1,”jsonrpc”:”2.0”,”result”:0}
02780     *
02781     * \endchinese
02782     * \english
02783     * Get the domain of standard analog input.
02784     *
02785     * @param index: Indicates the IO pin, starting from 0.
02786     * For example, 0 means the first pin.
02787     * @return Standard analog input domain
02788     *
02789     * @throws arcs::common_interface::AuboException
02790     *
02791     * @par Python function prototype
02792     * getStandardAnalogInputDomain(self: pyaubo_sdk.IoControl, arg0: int) ->
02793     * int
02794     *
02795     * @par Lua function prototype
02796     * getStandardAnalogInputDomain(index: number) -> number
02797     *
02798     * @par Lua example
02799     * num = getStandardAnalogInputDomain(0)
02800     *
02801     * @par JSON-RPC request example
02802     * {”jsonrpc”:”2.0”,”method”:”rob1.IoControl.getStandardAnalogInputDomain”,”params”:[0],”id”:1}
02803     *
02804     * @par JSON-RPC response example
02805     * {”id”:1,”jsonrpc”:”2.0”,”result”:0}
02806     *
02807     * \endenglish
02808     */
02809    int getStandardAnalogInputDomain(int index);
02810
02811    /**
02812     * @ingroup IoControl
02813     * \chinese
02814     *
02815     *
02816     * @param index:   IO       0
02817     *   0
02818     * @return
02819     *
02820     * @throws arcs::common_interface::AuboException
02821     *
02822     * @par Python
02823     * getToolAnalogInputDomain(self: pyaubo_sdk.IoControl, arg0: int) -> int
02824     *
02825     * @par Lua
02826     * getToolAnalogInputDomain(index: number) -> number
02827     *
02828     * @par Lua
02829     * num = getToolAnalogInputDomain(0)
02830     *
02831     * @par JSON-RPC
02832     * {”jsonrpc”:”2.0”,”method”:”rob1.IoControl.getToolAnalogInputDomain”,”params”:[0],”id”:1}
02833     *
02834     * @par JSON-RPC
02835     * {”id”:1,”jsonrpc”:”2.0”,”result”:10}
02836     *
02837     * \endchinese
02838     * \english
02839     * Get the domain of tool analog input.
02840     *
02841     * @param index: Indicates the IO pin, starting from 0.
02842     * For example, 0 means the first pin.
02843     * @return Tool analog input domain
02844     *
02845     * @throws arcs::common_interface::AuboException
02846     *
02847     * @par Python function prototype
02848     * getToolAnalogInputDomain(self: pyaubo_sdk.IoControl, arg0: int) -> int
02849     *
02850     * @par Lua function prototype
02851     * getToolAnalogInputDomain(index: number) -> number
02852     *
02853     * @par Lua example
02854     * num = getToolAnalogInputDomain(0)
02855     *
02856     * @par JSON-RPC request example
02857     * {”jsonrpc”:”2.0”,”method”:”rob1.IoControl.getToolAnalogInputDomain”,”params”:[0],”id”:1}
02858     *
02859     * @par JSON-RPC response example
```

```
02860     * {"id":1,"jsonrpc":"2.0","result":10}
02861     *
02862     * \endenglish
02863     */
02864    int getToolAnalogInputDomain(int index);
02865
02866    /**
02867     * @ingroup IoControl
02868     * \chinese
02869     *
02870     *
02871     * @param index:   IO       0
02872     *   0
02873     * @param domain:
02874     *
02875     * @return    0
02876     * AUBO_REQUEST_IGNORE
02877     * AUBO_BUSY
02878     * AUBO_BAD_STATE
02879     * -AUBO_INVL_ARGUMENT
02880     * -AUBO_BAD_STATE
02881     *
02882     * @throws arcs::common_interface::AuboException
02883     *
02884     * @par Python
02885     * setStandardAnalogOutputDomain(self: pyaubo_sdk.IoControl, arg0: int,
02886     * arg1: int) -> int
02887     *
02888     * @par Lua
02889     * setStandardAnalogOutputDomain(index: number, domain: number) -> nil
02890     *
02891     * @par Lua
02892     * setStandardAnalogOutputDomain(0,1)
02893     *
02894     * @par JSON-RPC
02895     * {"jsonrpc":"2.0","method":"rob1.IoControl.setStandardAnalogOutputDomain","params":[0,8],"id":1}
02896     *
02897     * @par JSON-RPC
02898     * {"id":1,"jsonrpc":"2.0","result":0}
02899     *
02900     * \endchinese
02901     * \english
02902     * Set the range of standard analog output.
02903     *
02904     * @param index: Indicates the IO pin, starting from 0.
02905     * For example, 0 means the first pin.
02906     * @param domain: Output range
02907     *
02908     * @return Returns 0 on success; error code on failure.
02909     * AUBO_REQUEST_IGNORE
02910     * AUBO_BUSY
02911     * AUBO_BAD_STATE
02912     * -AUBO_INVL_ARGUMENT
02913     * -AUBO_BAD_STATE
02914     *
02915     * @throws arcs::common_interface::AuboException
02916     *
02917     * @par Python function prototype
02918     * setStandardAnalogOutputDomain(self: pyaubo_sdk.IoControl, arg0: int,
02919     * arg1: int) -> int
02920     *
02921     * @par Lua function prototype
02922     * setStandardAnalogOutputDomain(index: number, domain: number) -> nil
02923     *
02924     * @par Lua example
02925     * setStandardAnalogOutputDomain(0,1)
02926     *
02927     * @par JSON-RPC request example
02928     * {"jsonrpc":"2.0","method":"rob1.IoControl.setStandardAnalogOutputDomain","params":[0,8],"id":1}
02929     *
02930     * @par JSON-RPC response example
02931     * {"id":1,"jsonrpc":"2.0","result":0}
02932     *
02933     * \endenglish
02934     */
02935    int setStandardAnalogOutputDomain(int index, int domain);
02936
02937    /**
02938     * @ingroup IoControl
02939     * \chinese
02940     *
02941     *
02942     * @param index:   IO       0
02943     *   0
02944     * @param domain:
02945     *
02946     * @return    0
```

```
02947     * AUBO_REQUEST_IGNORE
02948     * AUBO_BUSY
02949     * AUBO_BAD_STATE
02950     * -AUBO_INVL_ARGUMENT
02951     * -AUBO_BAD_STATE
02952     *
02953     * @throws arcs::common_interface::AuboException
02954     *
02955     * @par Python
02956     * setToolAnalogOutputDomain(self: pyaubo_sdk.IoControl, arg0: int, arg1:
02957     * int) -> int
02958     *
02959     * @par Lua
02960     * setToolAnalogOutputDomain(index: number, domain: number) -> nil
02961     *
02962     * @par Lua
02963     * setToolAnalogOutputDomain(0,1)
02964     *
02965     * \endchinese
02966     * \english
02967     * Set the range of tool analog output.
02968     *
02969     * @param index: Indicates the IO pin, starting from 0.
02970     * For example, 0 means the first pin.
02971     * @param domain: Output range
02972     *
02973     * @return Returns 0 on success; error code on failure.
02974     * AUBO_REQUEST_IGNORE
02975     * AUBO_BUSY
02976     * AUBO_BAD_STATE
02977     * -AUBO_INVL_ARGUMENT
02978     * -AUBO_BAD_STATE
02979     *
02980     * @throws arcs::common_interface::AuboException
02981     *
02982     * @par Python function prototype
02983     * setToolAnalogOutputDomain(self: pyaubo_sdk.IoControl, arg0: int, arg1:
02984     * int) -> int
02985     *
02986     * @par Lua function prototype
02987     * setToolAnalogOutputDomain(index: number, domain: number) -> nil
02988     *
02989     * @par Lua example
02990     * setToolAnalogOutputDomain(0,1)
02991     *
02992     * \endenglish
02993     */
02994    int setToolAnalogOutputDomain(int index, int domain);
02995
02996    /**
02997     * @ingroup IoControl
02998     * \chinese
02999     *
03000     *
03001     * @param index:   IO       0
03002     *   0
03003     * @return
03004     *
03005     * @throws arcs::common_interface::AuboException
03006     *
03007     * @par Python
03008     * getStandardAnalogOutputDomain(self: pyaubo_sdk.IoControl, arg0: int) ->
03009     * int
03010     *
03011     * @par Lua
03012     * getStandardAnalogOutputDomain(index: number) -> number
03013     *
03014     * @par Lua
03015     * num = getStandardAnalogOutputDomain(0)
03016     *
03017     * @par JSON-RPC
03018     * {"jsonrpc":"2.0","method":"rob1.IoControl.getStandardAnalogOutputDomain","params":[0],"id":1}
03019     *
03020     * @par JSON-RPC
03021     * {"id":1,"jsonrpc":"2.0","result":0}
03022     *
03023     * \endchinese
03024     * \english
03025     * Get the domain of standard analog output.
03026     *
03027     * @param index: Indicates the IO pin, starting from 0.
03028     * For example, 0 means the first pin.
03029     * @return Standard analog output domain
03030     *
03031     * @throws arcs::common_interface::AuboException
03032     *
03033     * @par Python function prototype
```

```
03034     * getStandardAnalogOutputDomain(self: pyaubo_sdk.IoControl, arg0: int) ->
03035     * int
03036     *
03037     * @par Lua function prototype
03038     * getStandardAnalogOutputDomain(index: number) -> number
03039     *
03040     * @par Lua example
03041     * num = getStandardAnalogOutputDomain(0)
03042     *
03043     * @par JSON-RPC request example
03044     * {"jsonrpc":"2.0","method":"rob1.IoControl.getStandardAnalogOutputDomain","params":[0],"id":1}
03045     *
03046     * @par JSON-RPC response example
03047     * {"id":1,"jsonrpc":"2.0","result":0}
03048     *
03049     * \endenglish
03050     */
03051    int getStandardAnalogOutputDomain(int index);
03052
03053    /**
03054     * @ingroup IoControl
03055     * \chinese
03056     *
03057     *
03058     * @param index:   IO       0
03059     *   0
03060     * @return
03061     *
03062     * @throws arcs::common_interface::AuboException
03063     *
03064     * @par Python
03065     * getToolAnalogOutputDomain(self: pyaubo_sdk.IoControl, arg0: int) -> int
03066     *
03067     * @par Lua
03068     * getToolAnalogOutputDomain(index: number) -> number
03069     *
03070     * @par Lua
03071     * num = getToolAnalogOutputDomain(0)
03072     *
03073     * @par JSON-RPC
03074     * {"jsonrpc":"2.0","method":"rob1.IoControl.getToolAnalogOutputDomain","params":[0],"id":1}
03075     *
03076     * @par JSON-RPC
03077     * {"id":1,"jsonrpc":"2.0","result":0}
03078     *
03079     * \endchinese
03080     * \english
03081     * Get the domain of tool analog output.
03082     *
03083     * @param index: Indicates the IO pin, starting from 0.
03084     * For example, 0 means the first pin.
03085     * @return Tool analog output domain
03086     *
03087     * @throws arcs::common_interface::AuboException
03088     *
03089     * @par Python function prototype
03090     * getToolAnalogOutputDomain(self: pyaubo_sdk.IoControl, arg0: int) -> int
03091     *
03092     * @par Lua function prototype
03093     * getToolAnalogOutputDomain(index: number) -> number
03094     *
03095     * @par Lua example
03096     * num = getToolAnalogOutputDomain(0)
03097     *
03098     * @par JSON-RPC request example
03099     * {"jsonrpc":"2.0","method":"rob1.IoControl.getToolAnalogOutputDomain","params":[0],"id":1}
03100     *
03101     * @par JSON-RPC response example
03102     * {"id":1,"jsonrpc":"2.0","result":0}
03103     *
03104     * \endenglish
03105     */
03106    int getToolAnalogOutputDomain(int index);
03107
03108    /**
03109     * @ingroup IoControl
03110     * \chinese
03111     *       ( V)
03112     *
03113     * @param domain:              0 12 24 \n
03114     *  0  0V, 12  12V, 24  24V
03115     *
03116     * @return    0;
03117     * AUBO_REQUEST_IGNORE
03118     * AUBO_BUSY
03119     * AUBO_BAD_STATE
03120     * -AUBO_INVL_ARGUMENT
```

```
03121     * -AUBO_BAD_STATE
03122     *
03123     * @throws arcs::common_interface::AuboException
03124     *
03125     * @par Python
03126     * setToolVoltageOutputDomain(self: pyaubo_sdk.IoControl, arg0: int) -> int
03127     *
03128     * @par Lua
03129     * setToolVoltageOutputDomain(domain: number) -> nil
03130     *
03131     * @par Lua
03132     * setToolVoltageOutputDomain(24)
03133     *
03134     * @par JSON-RPC
03135     * {"jsonrpc":"2.0","method":"rob1.IoControl.setToolVoltageOutputDomain","params":[24],"id":1}
03136     *
03137     * @par JSON-RPC
03138     * {"id":1,"jsonrpc":"2.0","result":0}
03139     *
03140     * \endchinese
03141     * \english
03142     * Set the tool voltage output value (unit: V)
03143     *
03144     * @param domain: Tool voltage output value, can be 0, 12, or 24.\n
03145     *  0 means 0V, 12 means 12V, 24 means 24V.
03146     *
03147     * @return Returns 0 on success; error code on failure.
03148     * AUBO_REQUEST_IGNORE
03149     * AUBO_BUSY
03150     * AUBO_BAD_STATE
03151     * -AUBO_INVL_ARGUMENT
03152     * -AUBO_BAD_STATE
03153     *
03154     * @throws arcs::common_interface::AuboException
03155     *
03156     * @par Python function prototype
03157     * setToolVoltageOutputDomain(self: pyaubo_sdk.IoControl, arg0: int) -> int
03158     *
03159     * @par Lua function prototype
03160     * setToolVoltageOutputDomain(domain: number) -> nil
03161     *
03162     * @par Lua example
03163     * setToolVoltageOutputDomain(24)
03164     *
03165     * @par JSON-RPC request example
03166     * {"jsonrpc":"2.0","method":"rob1.IoControl.setToolVoltageOutputDomain","params":[24],"id":1}
03167     *
03168     * @par JSON-RPC response example
03169     * {"id":1,"jsonrpc":"2.0","result":0}
03170     *
03171     * \endenglish
03172     */
03173    int setToolVoltageOutputDomain(int domain);
03174
03175    /**
03176     * @ingroup IoControl
03177     * \chinese
03178     *       ( V)
03179     *
03180     * @return       ( V)
03181     *
03182     * @throws arcs::common_interface::AuboException
03183     *
03184     * @par Python
03185     * getToolVoltageOutputDomain(self: pyaubo_sdk.IoControl) -> int
03186     *
03187     * @par Lua
03188     * getToolVoltageOutputDomain() -> number
03189     *
03190     * @par Lua
03191     * num = getToolVoltageOutputDomain()
03192     *
03193     * @par JSON-RPC
03194     * {"jsonrpc":"2.0","method":"rob1.IoControl.getToolVoltageOutputDomain","params":[],"id":1}
03195     *
03196     * @par JSON-RPC
03197     * {"id":1,"jsonrpc":"2.0","result":0}
03198     *
03199     * \endchinese
03200     * \english
03201     * Get the tool voltage output value (unit: V)
03202     *
03203     * @return Tool voltage output value (unit: V)
03204     *
03205     * @throws arcs::common_interface::AuboException
03206     *
03207     * @par Python function prototype
```

```
03208     * getToolVoltageOutputDomain(self: pyaubo_sdk.IoControl) -> int
03209     *
03210     * @par Lua function prototype
03211     * getToolVoltageOutputDomain() -> number
03212     *
03213     * @par Lua example
03214     * num = getToolVoltageOutputDomain()
03215     *
03216     * @par JSON-RPC request example
03217     * {"jsonrpc":"2.0","method":"rob1.IoControl.getToolVoltageOutputDomain","params":[],"id":1}
03218     *
03219     * @par JSON-RPC response example
03220     * {"id":1,"jsonrpc":"2.0","result":0}
03221     *
03222     * \endenglish
03223     */
03224    int getToolVoltageOutputDomain();
03225
03226    /**
03227     * @ingroup IoControl
03228     * \chinese
03229     *
03230     *
03231     * @param index:    IO
03232     * @param value:
03233     * @return    0
03234     * AUBO_REQUEST_IGNORE
03235     * AUBO_BUSY
03236     * AUBO_BAD_STATE
03237     * -AUBO_INVL_ARGUMENT
03238     * -AUBO_BAD_STATE
03239     *
03240     * @throws arcs::common_interface::AuboException
03241     *
03242     * @par Python
03243     * setStandardDigitalOutput(self: pyaubo_sdk.IoControl, arg0: int, arg1:
03244     * bool) -> int
03245     *
03246     * @par Lua
03247     * setStandardDigitalOutput(index: number, value: boolean) -> nil
03248     *
03249     * @par Lua
03250     * setStandardDigitalOutput(0,true)
03251     *
03252     * @par JSON-RPC
03253     * {"jsonrpc":"2.0","method":"rob1.IoControl.setStandardDigitalOutput","params":[0,true],"id":1}
03254     *
03255     * @par JSON-RPC
03256     * {"id":1,"jsonrpc":"2.0","result":0}
03257     *
03258     * \endchinese
03259     * \english
03260     * Set the value of a standard digital output.
03261     *
03262     * @param index: Indicates the IO pin.
03263     * @param value: Output value.
03264     * @return Returns 0 on success; error code on failure.
03265     * AUBO_REQUEST_IGNORE
03266     * AUBO_BUSY
03267     * AUBO_BAD_STATE
03268     * -AUBO_INVL_ARGUMENT
03269     * -AUBO_BAD_STATE
03270     *
03271     * @throws arcs::common_interface::AuboException
03272     *
03273     * @par Python function prototype
03274     * setStandardDigitalOutput(self: pyaubo_sdk.IoControl, arg0: int, arg1:
03275     * bool) -> int
03276     *
03277     * @par Lua function prototype
03278     * setStandardDigitalOutput(index: number, value: boolean) -> nil
03279     *
03280     * @par Lua example
03281     * setStandardDigitalOutput(0,true)
03282     *
03283     * @par JSON-RPC request example
03284     * {"jsonrpc":"2.0","method":"rob1.IoControl.setStandardDigitalOutput","params":[0,true],"id":1}
03285     *
03286     * @par JSON-RPC response example
03287     * {"id":1,"jsonrpc":"2.0","result":0}
03288     *
03289     * \endenglish
03290     */
03291    int setStandardDigitalOutput(int index, bool value);
03292
03293    /**
03294     * @ingroup IoControl
```

```
03295     * \chinese
03296     *
03297     *
03298     *
03299     * `indexes[0]‘    bit0 ‘indexes[1]‘    bit1
03300     *
03301     * @param name:
03302     * @param io_type: IO   0  Standard 1  Tool 2  Configurable
03303     * @param indexes:
03304     *
03305     * @return    0
03306     *
03307     * @throws arcs::common_interface::AuboException
03308     *
03309     * @par Python
03310     * addGroupOutput(self: pyaubo_sdk.IoControl, arg0: str, arg1: int, arg2:
03311     * List[int]) -> int
03312     *
03313     * @par Lua
03314     * addGroupOutput(name: string, io_type: number, indexes: table) -> nil
03315     *
03316     * @par Lua
03317     * addGroupOutput(”go_state”, 0, {4,5,6,7})
03318     *
03319     * @par JSON-RPC
03320     * {”jsonrpc”:”2.0”,”method”:”rob1.IoControl.addGroupOutput”,”params”:[”go_state”,0,[4,5,6,7]],”id”:1}
03321     *
03322     * @par JSON-RPC
03323     * {”id”:1,”jsonrpc”:”2.0”,”result”:0}
03324     *
03325     * \endchinese
03326     * \english
03327     * Register a grouped digital output.
03328     *
03329     * Registers a named digital output group. `indexes[0]‘ maps to bit0,
03330     * `indexes[1]‘ maps to bit1, and so on.
03331     *
03332     * @param name: Group name.
03333     * @param io_type: IO type. 0 means Standard, 1 means Tool, and 2 means
03334     * Configurable.
03335     * @param indexes: Index list of digital outputs in the group.
03336     *
03337     * @return Returns 0 on success; error code on failure.
03338     *
03339     * @throws arcs::common_interface::AuboException
03340     *
03341     * @par Python function prototype
03342     * addGroupOutput(self: pyaubo_sdk.IoControl, arg0: str, arg1: int, arg2:
03343     * List[int]) -> int
03344     *
03345     * @par Lua function prototype
03346     * addGroupOutput(name: string, io_type: number, indexes: table) -> nil
03347     *
03348     * @par Lua example
03349     * addGroupOutput(”go_state”, 0, {4,5,6,7})
03350     *
03351     * @par JSON-RPC request example
03352     * {”jsonrpc”:”2.0”,”method”:”rob1.IoControl.addGroupOutput”,”params”:[”go_state”,0,[4,5,6,7]],”id”:1}
03353     *
03354     * @par JSON-RPC response example
03355     * {”id”:1,”jsonrpc”:”2.0”,”result”:0}
03356     *
03357     * \endenglish
03358     */
03359    int addGroupOutput(const std::string &name, int io_type,
03360                       const std::vector<int> &indexes);
03361
03362    /**
03363     * @ingroup IoControl
03364     * \chinese
03365     *
03366     *
03367     * @param name:
03368     *
03369     * @return    0
03370     *
03371     * @throws arcs::common_interface::AuboException
03372     *
03373     * @par Python
03374     * deleteGroupOutput(self: pyaubo_sdk.IoControl, arg0: str) -> int
03375     *
03376     * @par Lua
03377     * deleteGroupOutput(name: string) -> nil
03378     *
03379     * @par JSON-RPC
03380     * {”jsonrpc”:”2.0”,”method”:”rob1.IoControl.deleteGroupOutput”,”params”:[”go_state”],”id”:1}
03381     *
```

```
03382     * @par JSON-RPC
03383     * {”id”:1,”jsonrpc”:”2.0”,”result”:0}
03384     *
03385     * \endchinese
03386     * \english
03387     * Delete a grouped digital output.
03388     *
03389     * @param name: Group name.
03390     *
03391     * @return Returns 0 on success; error code on failure.
03392     *
03393     * @throws arcs::common_interface::AuboException
03394     *
03395     * @par Python function prototype
03396     * deleteGroupOutput(self: pyaubo_sdk.IoControl, arg0: str) -> int
03397     *
03398     * @par Lua function prototype
03399     * deleteGroupOutput(name: string) -> nil
03400     *
03401     * @par JSON-RPC request example
03402     * {”jsonrpc”:”2.0”,”method”:”rob1.IoControl.deleteGroupOutput”,”params”:[”go_state”],”id”:1}
03403     *
03404     * @par JSON-RPC response example
03405     * {”id”:1,”jsonrpc”:”2.0”,”result”:0}
03406     *
03407     * \endenglish
03408     */
03409    int deleteGroupOutput(const std::string &name);
03410
03411    /**
03412     * @ingroup IoControl
03413     * \chinese
03414     *
03415     *
03416     *          `value‘
03417     *
03418     * @param name:
03419     * @param value:
03420     *
03421     * @return    0
03422     * AUBO_REQUEST_IGNORE
03423     * AUBO_BUSY
03424     * AUBO_BAD_STATE
03425     * -AUBO_INVL_ARGUMENT
03426     * -AUBO_BAD_STATE
03427     *
03428     * @throws arcs::common_interface::AuboException
03429     *
03430     * @par Python
03431     * setGroupOutput(self: pyaubo_sdk.IoControl, arg0: str, arg1: int) -> int
03432     *
03433     * @par Lua
03434     * setGroupOutput(name: string, value: number) -> nil
03435     *
03436     * @par Lua
03437     * setGroupOutput(”go_state”, 13)
03438     *
03439     * @par JSON-RPC
03440     * {”jsonrpc”:”2.0”,”method”:”rob1.IoControl.setGroupOutput”,”params”:[”go_state”,13],”id”:1}
03441     *
03442     * @par JSON-RPC
03443     * {”id”:1,”jsonrpc”:”2.0”,”result”:0}
03444     *
03445     * \endchinese
03446     * \english
03447     * Set the grouped digital output value.
03448     *
03449     * Looks up the named grouped digital output and writes `value‘ to the
03450     * group’s members bit by bit.
03451     *
03452     * @param name: Group name.
03453     * @param value: Integer value to write to the grouped digital output.
03454     *
03455     * @return Returns 0 on success; error code on failure.
03456     * AUBO_REQUEST_IGNORE
03457     * AUBO_BUSY
03458     * AUBO_BAD_STATE
03459     * -AUBO_INVL_ARGUMENT
03460     * -AUBO_BAD_STATE
03461     *
03462     * @throws arcs::common_interface::AuboException
03463     *
03464     * @par Python function prototype
03465     * setGroupOutput(self: pyaubo_sdk.IoControl, arg0: str, arg1: int) -> int
03466     *
03467     * @par Lua function prototype
03468     * setGroupOutput(name: string, value: number) -> nil
```

```
03469     *
03470     * @par Lua example
03471     * setGroupOutput("go_state", 13)
03472     *
03473     * @par JSON-RPC request example
03474     * {"jsonrpc":"2.0","method":"rob1.IoControl.setGroupOutput","params":["go_state",13],"id":1}
03475     *
03476     * @par JSON-RPC response example
03477     * {"id":1,"jsonrpc":"2.0","result":0}
03478     *
03479     * \endenglish
03480     */
03481    int setGroupOutput(const std::string &name, uint32_t value);
03482
03483    /**
03484     * @ingroup IoControl
03485     * \chinese
03486     *
03487     *
03488     * @param index
03489     * @param value
03490     * @param duration
03491     *
03492     * @return    0
03493     * AUBO_REQUEST_IGNORE
03494     * AUBO_BUSY
03495     * AUBO_BAD_STATE
03496     * -AUBO_INVL_ARGUMENT
03497     * -AUBO_BAD_STATE
03498     *
03499     * @throws arcs::common_interface::AuboException
03500     *
03501     * @par Python
03502     * setStandardDigitalOutputPulse(self: pyaubo_sdk.IoControl, arg0: int,
03503     * arg1: bool, arg2: float) -> int
03504     *
03505     * @par Lua
03506     * setStandardDigitalOutputPulse(index: number, value: boolean, duration:
03507     * number) -> nil
03508     *
03509     * @par Lua
03510     * setStandardDigitalOutputPulse(0,true,2.5)
03511     *
03512     * @par JSON-RPC
03513     * {"jsonrpc":"2.0","method":"rob1.IoControl.setStandardDigitalOutputPulse","params":[0,true,0.5],"id":1}
03514     *
03515     * @par JSON-RPC
03516     * {"id":1,"jsonrpc":"2.0","result":0}
03517     *
03518     * \endchinese
03519     * \english
03520     * Set digital output pulse.
03521     *
03522     * @param index
03523     * @param value
03524     * @param duration
03525     *
03526     * @return Returns 0 on success; error code on failure.
03527     * AUBO_REQUEST_IGNORE
03528     * AUBO_BUSY
03529     * AUBO_BAD_STATE
03530     * -AUBO_INVL_ARGUMENT
03531     * -AUBO_BAD_STATE
03532     *
03533     * @throws arcs::common_interface::AuboException
03534     *
03535     * @par Python function prototype
03536     * setStandardDigitalOutputPulse(self: pyaubo_sdk.IoControl, arg0: int,
03537     * arg1: bool, arg2: float) -> int
03538     *
03539     * @par Lua function prototype
03540     * setStandardDigitalOutputPulse(index: number, value: boolean, duration:
03541     * number) -> nil
03542     *
03543     * @par Lua example
03544     * setStandardDigitalOutputPulse(0,true,2.5)
03545     *
03546     * @par JSON-RPC request example
03547     * {"jsonrpc":"2.0","method":"rob1.IoControl.setStandardDigitalOutputPulse","params":[0,true,0.5],"id":1}
03548     *
03549     * @par JSON-RPC response example
03550     * {"id":1,"jsonrpc":"2.0","result":0}
03551     *
03552     * \endenglish
03553     */
03554    int setStandardDigitalOutputPulse(int index, bool value, double duration);
03555
```

```
03556     /**
03557      * @ingroup IoControl
03558      * \chinese
03559      *
03560      *
03561      * @param index:   IO       0
03562      *   0
03563      * @param value:
03564      *
03565      * @return    0
03566      * AUBO_REQUEST_IGNORE
03567      * AUBO_BUSY
03568      * AUBO_BAD_STATE
03569      * -AUBO_INVL_ARGUMENT
03570      * -AUBO_BAD_STATE
03571      *
03572      * @throws arcs::common_interface::AuboException
03573      *
03574      * @par Python
03575      * setToolDigitalOutput(self: pyaubo_sdk.IoControl, arg0: int, arg1: bool)
03576      * -> int
03577      *
03578      * @par Lua
03579      * setToolDigitalOutput(index: number, value: boolean) -> nil
03580      *
03581      * @par Lua
03582      * setToolDigitalOutput(0,true)
03583      *
03584      * @par JSON-RPC
03585      * {"jsonrpc":"2.0","method":"rob1.IoControl.setToolDigitalOutput","params":[0,true],"id":1}
03586      *
03587      * @par JSON-RPC
03588      * {"id":1,"jsonrpc":"2.0","result":0}
03589      *
03590      * \endchinese
03591      * \english
03592      * Set the value of tool digital output.
03593      *
03594      * @param index: Indicates the IO pin, starting from 0.
03595      * For example, 0 means the first pin.
03596      * @param value: Output value.
03597      *
03598      * @return Returns 0 on success; error code on failure.
03599      * AUBO_REQUEST_IGNORE
03600      * AUBO_BUSY
03601      * AUBO_BAD_STATE
03602      * -AUBO_INVL_ARGUMENT
03603      * -AUBO_BAD_STATE
03604      *
03605      * @throws arcs::common_interface::AuboException
03606      *
03607      * @par Python function prototype
03608      * setToolDigitalOutput(self: pyaubo_sdk.IoControl, arg0: int, arg1: bool)
03609      * -> int
03610      *
03611      * @par Lua function prototype
03612      * setToolDigitalOutput(index: number, value: boolean) -> nil
03613      *
03614      * @par Lua example
03615      * setToolDigitalOutput(0,true)
03616      *
03617      * @par JSON-RPC request example
03618      * {"jsonrpc":"2.0","method":"rob1.IoControl.setToolDigitalOutput","params":[0,true],"id":1}
03619      *
03620      * @par JSON-RPC response example
03621      * {"id":1,"jsonrpc":"2.0","result":0}
03622      *
03623      * \endenglish
03624      */
03625     int setToolDigitalOutput(int index, bool value);
03626
03627     /**
03628      * @ingroup IoControl
03629      * \chinese
03630      *
03631      *
03632      * @param index
03633      * @param value
03634      * @param duration
03635      *
03636      * @return    0
03637      * AUBO_REQUEST_IGNORE
03638      * AUBO_BUSY
03639      * AUBO_BAD_STATE
03640      * -AUBO_INVL_ARGUMENT
03641      * -AUBO_BAD_STATE
03642      *
```

```
03643     * @throws arcs::common_interface::AuboException
03644     *
03645     * @par Python
03646     * setToolDigitalOutputPulse(self: pyaubo_sdk.IoControl, arg0: int, arg1:
03647     * bool, arg2: float) -> int
03648     *
03649     * @par Lua
03650     * setToolDigitalOutputPulse(index: number, value: boolean, duration:
03651     * number) -> nil
03652     *
03653     * @par Lua
03654     * setToolDigitalOutputPulse(0,true,2)
03655     *
03656     * @par JSON-RPC
03657     * {"jsonrpc":"2.0","method":"rob1.IoControl.setToolDigitalOutputPulse","params":[0,true,0.5],"id":1}
03658     *
03659     * @par JSON-RPC
03660     * {"id":1,"jsonrpc":"2.0","result":0}
03661     *
03662     * \endchinese
03663     * \english
03664     * Set tool digital output pulse.
03665     *
03666     * @param index
03667     * @param value
03668     * @param duration
03669     *
03670     * @return Returns 0 on success; error code on failure.
03671     * AUBO_REQUEST_IGNORE
03672     * AUBO_BUSY
03673     * AUBO_BAD_STATE
03674     * -AUBO_INVL_ARGUMENT
03675     * -AUBO_BAD_STATE
03676     *
03677     * @throws arcs::common_interface::AuboException
03678     *
03679     * @par Python function prototype
03680     * setToolDigitalOutputPulse(self: pyaubo_sdk.IoControl, arg0: int, arg1:
03681     * bool, arg2: float) -> int
03682     *
03683     * @par Lua function prototype
03684     * setToolDigitalOutputPulse(index: number, value: boolean, duration:
03685     * number) -> nil
03686     *
03687     * @par Lua example
03688     * setToolDigitalOutputPulse(0,true,2)
03689     *
03690     * @par JSON-RPC request example
03691     * {"jsonrpc":"2.0","method":"rob1.IoControl.setToolDigitalOutputPulse","params":[0,true,0.5],"id":1}
03692     *
03693     * @par JSON-RPC response example
03694     * {"id":1,"jsonrpc":"2.0","result":0}
03695     *
03696     * \endenglish
03697     */
03698    int setToolDigitalOutputPulse(int index, bool value, double duration);
03699
03700    /**
03701     * @ingroup IoControl
03702     * \chinese
03703     *
03704     *
03705     * @param index:   IO       0
03706     *   0
03707     * @param value:
03708     *
03709     * @return    0
03710     * AUBO_REQUEST_IGNORE
03711     * AUBO_BUSY
03712     * AUBO_BAD_STATE
03713     * -AUBO_INVL_ARGUMENT
03714     * -AUBO_BAD_STATE
03715     *
03716     * @throws arcs::common_interface::AuboException
03717     *
03718     * @par Python
03719     * setConfigurableDigitalOutput(self: pyaubo_sdk.IoControl, arg0: int, arg1:
03720     * bool) -> int
03721     *
03722     * @par Lua
03723     * setConfigurableDigitalOutput(index: number, value: boolean) -> nil
03724     *
03725     * @par Lua
03726     * setConfigurableDigitalOutput(0,true)
03727     *
03728     * @par JSON-RPC
03729     * {"jsonrpc":"2.0","method":"rob1.IoControl.setConfigurableDigitalOutput","params":[0,true],"id":1}
```

```
03730     *
03731     * @par JSON-RPC
03732     * {"id":1,"jsonrpc":"2.0","result":0}
03733     *
03734     * \endchinese
03735     * \english
03736     * Set the value of configurable digital output.
03737     *
03738     * @param index: Indicates the IO pin, starting from 0.
03739     * For example, 0 means the first pin.
03740     * @param value: Output value.
03741     *
03742     * @return Returns 0 on success; error code on failure.
03743     * AUBO_REQUEST_IGNORE
03744     * AUBO_BUSY
03745     * AUBO_BAD_STATE
03746     * -AUBO_INVL_ARGUMENT
03747     * -AUBO_BAD_STATE
03748     *
03749     * @throws arcs::common_interface::AuboException
03750     *
03751     * @par Python function prototype
03752     * setConfigurableDigitalOutput(self: pyaubo_sdk.IoControl, arg0: int, arg1:
03753     * bool) -> int
03754     *
03755     * @par Lua function prototype
03756     * setConfigurableDigitalOutput(index: number, value: boolean) -> nil
03757     *
03758     * @par Lua example
03759     * setConfigurableDigitalOutput(0,true)
03760     *
03761     * @par JSON-RPC request example
03762     * {"jsonrpc":"2.0","method":"rob1.IoControl.setConfigurableDigitalOutput","params":[0,true],"id":1}
03763     *
03764     * @par JSON-RPC response example
03765     * {"id":1,"jsonrpc":"2.0","result":0}
03766     *
03767     * \endenglish
03768     */
03769    int setConfigurableDigitalOutput(int index, bool value);
03770
03771    /**
03772     * @ingroup IoControl
03773     * \chinese
03774     *
03775     *
03776     * @param index
03777     * @param value
03778     * @param duration
03779     *
03780     * @return    0
03781     * AUBO_REQUEST_IGNORE
03782     * AUBO_BUSY
03783     * AUBO_BAD_STATE
03784     * -AUBO_INVL_ARGUMENT
03785     * -AUBO_BAD_STATE
03786     *
03787     * @throws arcs::common_interface::AuboException
03788     *
03789     * @par Python
03790     * setConfigurableDigitalOutputPulse(self: pyaubo_sdk.IoControl, arg0: int,
03791     * arg1: bool, arg2: float) -> int
03792     *
03793     * @par Lua
03794     * setConfigurableDigitalOutputPulse(index: number, value: boolean,
03795     * duration: number) -> nil
03796     *
03797     * @par Lua
03798     * setConfigurableDigitalOutputPulse(0,true,2.3)
03799     *
03800     * @par JSON-RPC
03801     * {"jsonrpc":"2.0","method":"rob1.IoControl.setConfigurableDigitalOutputPulse","params":[0,true,0.5],"id":1}
03802     *
03803     * @par JSON-RPC
03804     * {"id":1,"jsonrpc":"2.0","result":0}
03805     *
03806     * \endchinese
03807     * \english
03808     * Set configurable digital output pulse.
03809     *
03810     * @param index
03811     * @param value
03812     * @param duration
03813     *
03814     * @return Returns 0 on success; error code on failure.
03815     * AUBO_REQUEST_IGNORE
03816     * AUBO_BUSY
```

```
03817     * AUBO_BAD_STATE
03818     * -AUBO_INVL_ARGUMENT
03819     * -AUBO_BAD_STATE
03820     *
03821     * @throws arcs::common_interface::AuboException
03822     *
03823     * @par Python function prototype
03824     * setConfigurableDigitalOutputPulse(self: pyaubo_sdk.IoControl, arg0: int,
03825     * arg1: bool, arg2: float) -> int
03826     *
03827     * @par Lua function prototype
03828     * setConfigurableDigitalOutputPulse(index: number, value: boolean,
03829     * duration: number) -> nil
03830     *
03831     * @par Lua example
03832     * setConfigurableDigitalOutputPulse(0,true,2.3)
03833     *
03834     * @par JSON-RPC request example
03835     * {"jsonrpc":"2.0","method":"rob1.IoControl.setConfigurableDigitalOutputPulse","params":[0,true,0.5],"id":1}
03836     *
03837     * @par JSON-RPC response example
03838     * {"id":1,"jsonrpc":"2.0","result":0}
03839     *
03840     * \endenglish
03841     */
03842    int setConfigurableDigitalOutputPulse(int index, bool value,
03843                                          double duration);
03844
03845    /**
03846     * @ingroup IoControl
03847     * \chinese
03848     *
03849     *
03850     * @param index:   IO       0
03851     *   0
03852     * @param value:
03853     *
03854     * @return    0
03855     * AUBO_REQUEST_IGNORE
03856     * AUBO_BUSY
03857     * AUBO_BAD_STATE
03858     * -AUBO_INVL_ARGUMENT
03859     * -AUBO_BAD_STATE
03860     *
03861     * @throws arcs::common_interface::AuboException
03862     *
03863     * @par Python
03864     * setStandardAnalogOutput(self: pyaubo_sdk.IoControl, arg0: int, arg1:
03865     * float) -> int
03866     *
03867     * @par Lua
03868     * setStandardAnalogOutput(index: number, value: number) -> nil
03869     *
03870     * @par Lua
03871     * setStandardAnalogOutput(0,5.4)
03872     *
03873     * \endchinese
03874     * \english
03875     * Set the value of standard analog output.
03876     *
03877     * @param index: Indicates the IO pin, starting from 0.
03878     * For example, 0 means the first pin.
03879     * @param value: Output value.
03880     *
03881     * @return Returns 0 on success; error code on failure.
03882     * AUBO_REQUEST_IGNORE
03883     * AUBO_BUSY
03884     * AUBO_BAD_STATE
03885     * -AUBO_INVL_ARGUMENT
03886     * -AUBO_BAD_STATE
03887     *
03888     * @throws arcs::common_interface::AuboException
03889     *
03890     * @par Python function prototype
03891     * setStandardAnalogOutput(self: pyaubo_sdk.IoControl, arg0: int, arg1:
03892     * float) -> int
03893     *
03894     * @par Lua function prototype
03895     * setStandardAnalogOutput(index: number, value: number) -> nil
03896     *
03897     * @par Lua example
03898     * setStandardAnalogOutput(0,5.4)
03899     *
03900     * \endenglish
03901     */
03902    int setStandardAnalogOutput(int index, double value);
03903
```

```
03904     /**
03905      * @ingroup IoControl
03906      * \chinese
03907      *
03908      *
03909      * @param index:   IO       0
03910      *    0
03911      * @param value:
03912      *
03913      * @return    0
03914      * AUBO_REQUEST_IGNORE
03915      * AUBO_BUSY
03916      * AUBO_BAD_STATE
03917      * -AUBO_INVL_ARGUMENT
03918      * -AUBO_BAD_STATE
03919      *
03920      * @throws arcs::common_interface::AuboException
03921      *
03922      * @par Python
03923      * setToolAnalogOutput(self: pyaubo_sdk.IoControl, arg0: int, arg1: float)
03924      * -> int
03925      *
03926      * @par Lua
03927      * setToolAnalogOutput(index: number, value: number) -> nil
03928      *
03929      * @par Lua
03930      * setToolAnalogOutput(0,1.2)
03931      *
03932      * @par JSON-RPC
03933      * {"jsonrpc":"2.0","method":"rob1.IoControl.setToolAnalogOutput","params":[0,0.5],"id":1}
03934      *
03935      * @par JSON-RPC
03936      * {"id":1,"jsonrpc":"2.0","result":13}
03937      *
03938      * \endchinese
03939      * \english
03940      * Set the value of tool analog output.
03941      *
03942      * @param index: Indicates the IO pin, starting from 0.
03943      * For example, 0 means the first pin.
03944      * @param value: Output value.
03945      *
03946      * @return Returns 0 on success; error code on failure.
03947      * AUBO_REQUEST_IGNORE
03948      * AUBO_BUSY
03949      * AUBO_BAD_STATE
03950      * -AUBO_INVL_ARGUMENT
03951      * -AUBO_BAD_STATE
03952      *
03953      * @throws arcs::common_interface::AuboException
03954      *
03955      * @par Python function prototype
03956      * setToolAnalogOutput(self: pyaubo_sdk.IoControl, arg0: int, arg1: float)
03957      * -> int
03958      *
03959      * @par Lua function prototype
03960      * setToolAnalogOutput(index: number, value: number) -> nil
03961      *
03962      * @par Lua example
03963      * setToolAnalogOutput(0,1.2)
03964      *
03965      * @par JSON-RPC request example
03966      * {"jsonrpc":"2.0","method":"rob1.IoControl.setToolAnalogOutput","params":[0,0.5],"id":1}
03967      *
03968      * @par JSON-RPC response example
03969      * {"id":1,"jsonrpc":"2.0","result":13}
03970      *
03971      * \endenglish
03972      */
03973     int setToolAnalogOutput(int index, double value);
03974
03975     /**
03976      * @ingroup IoControl
03977      * \chinese
03978      *
03979      *
03980      * @param index:   IO       0
03981      *    0
03982      *
03983      * @return     true;     false
03984      *
03985      * @throws arcs::common_interface::AuboException
03986      *
03987      * @par Python
03988      * getStandardDigitalInput(self: pyaubo_sdk.IoControl, arg0: int) -> bool
03989      *
03990      * @par Lua
```

```
03991     * getStandardDigitalInput(index: number) -> boolean
03992     *
03993     * @par Lua
03994     * status = getStandardDigitalInput(0)
03995     *
03996     * @par JSON-RPC
03997     * {"jsonrpc":"2.0","method":"rob1.IoControl.getStandardDigitalInput","params":[0],"id":1}
03998     *
03999     * @par JSON-RPC
04000     * {"id":1,"jsonrpc":"2.0","result":false}
04001     *
04002     * \endchinese
04003     * \english
04004     * Get the value of a standard digital input.
04005     *
04006     * @param index: Indicates the IO pin, starting from 0.
04007     * For example, 0 means the first pin.
04008     *
04009     * @return Returns true for high level; false for low level.
04010     *
04011     * @throws arcs::common_interface::AuboException
04012     *
04013     * @par Python function prototype
04014     * getStandardDigitalInput(self: pyaubo_sdk.IoControl, arg0: int) -> bool
04015     *
04016     * @par Lua function prototype
04017     * getStandardDigitalInput(index: number) -> boolean
04018     *
04019     * @par Lua example
04020     * status = getStandardDigitalInput(0)
04021     *
04022     * @par JSON-RPC request example
04023     * {"jsonrpc":"2.0","method":"rob1.IoControl.getStandardDigitalInput","params":[0],"id":1}
04024     *
04025     * @par JSON-RPC response example
04026     * {"id":1,"jsonrpc":"2.0","result":false}
04027     *
04028     * \endenglish
04029     */
04030    bool getStandardDigitalInput(int index);
04031
04032    /**
04033     * @ingroup IoControl
04034     * \chinese
04035     *
04036     *
04037     * @return          \n
04038     *       2863267846 ,  2    10101010101010100000000000000110
04039     *  16
04040     *     DI00     ,      DI01          \n
04041     * 1      0
04042     *
04043     * @throws arcs::common_interface::AuboException
04044     *
04045     * @par Python
04046     * getStandardDigitalInputs(self: pyaubo_sdk.IoControl) -> int
04047     *
04048     * @par Lua
04049     * getStandardDigitalInputs() -> number
04050     *
04051     * @par Lua
04052     * num = getStandardDigitalInputs()
04053     *
04054     * @par JSON-RPC
04055     * {"jsonrpc":"2.0","method":"rob1.IoControl.getStandardDigitalInputs","params":[],"id":1}
04056     *
04057     * @par JSON-RPC
04058     * {"id":1,"jsonrpc":"2.0","result":0}
04059     *
04060     * \endchinese
04061     * \english
04062     * Get all standard digital input values.
04063     *
04064     * @return All standard digital input values.\n
04065     * For example, if the return value is 2863267846, its binary representation
04066     * is 10101010101010100000000000000110. The lower 16 bits represent the
04067     * status of all standard digital inputs, the least significant bit
04068     * indicates the input status of DI00, the second least significant bit
04069     * indicates DI01, and so on.\n 1 means high level, 0 means low level.
04070     *
04071     * @throws arcs::common_interface::AuboException
04072     *
04073     * @par Python function prototype
04074     * getStandardDigitalInputs(self: pyaubo_sdk.IoControl) -> int
04075     *
04076     * @par Lua function prototype
04077     * getStandardDigitalInputs() -> number
```

```
04078     *
04079     * @par Lua example
04080     * num = getStandardDigitalInputs()
04081     *
04082     * @par JSON-RPC request example
04083     * {”jsonrpc”:”2.0”,”method”:”rob1.IoControl.getStandardDigitalInputs”,”params”:[],”id”:1}
04084     *
04085     * @par JSON-RPC response example
04086     * {”id”:1,”jsonrpc”:”2.0”,”result”:0}
04087     *
04088     * \endenglish
04089     */
04090    uint32_t getStandardDigitalInputs();
04091
04092    /**
04093     * @ingroup IoControl
04094     * \chinese
04095     *
04096     *
04097     *
04098     * `indexes[0]‘    bit0 ‘indexes[1]‘    bit1
04099     *
04100     * @param name:
04101     * @param io_type: IO   0  Standard 1  Tool 2  Configurable
04102     * @param indexes:
04103     *
04104     * @return    0
04105     *
04106     * @throws arcs::common_interface::AuboException
04107     *
04108     * @par Python
04109     * addGroupInput(self: pyaubo_sdk.IoControl, arg0: str, arg1: int, arg2:
04110     * List[int]) -> int
04111     *
04112     * @par Lua
04113     * addGroupInput(name: string, io_type: number, indexes: table) -> nil
04114     *
04115     * @par Lua
04116     * addGroupInput(”gi_state”, 0, {0,1,2,3})
04117     *
04118     * @par JSON-RPC
04119     * {”jsonrpc”:”2.0”,”method”:”rob1.IoControl.addGroupInput”,”params”:[”gi_state”,0,[0,1,2,3]],”id”:1}
04120     *
04121     * @par JSON-RPC
04122     * {”id”:1,”jsonrpc”:”2.0”,”result”:0}
04123     *
04124     * \endchinese
04125     * \english
04126     * Register a grouped digital input.
04127     *
04128     * Registers a named digital input group. `indexes[0]‘ maps to bit0,
04129     * `indexes[1]‘ maps to bit1, and so on.
04130     *
04131     * @param name: Group name.
04132     * @param io_type: IO type. 0 means Standard, 1 means Tool, and 2 means
04133     * Configurable.
04134     * @param indexes: Index list of digital inputs in the group.
04135     *
04136     * @return Returns 0 on success; error code on failure.
04137     *
04138     * @throws arcs::common_interface::AuboException
04139     *
04140     * @par Python function prototype
04141     * addGroupInput(self: pyaubo_sdk.IoControl, arg0: str, arg1: int, arg2:
04142     * List[int]) -> int
04143     *
04144     * @par Lua function prototype
04145     * addGroupInput(name: string, io_type: number, indexes: table) -> nil
04146     *
04147     * @par Lua example
04148     * addGroupInput(”gi_state”, 0, {0,1,2,3})
04149     *
04150     * @par JSON-RPC request example
04151     * {”jsonrpc”:”2.0”,”method”:”rob1.IoControl.addGroupInput”,”params”:[”gi_state”,0,[0,1,2,3]],”id”:1}
04152     *
04153     * @par JSON-RPC response example
04154     * {”id”:1,”jsonrpc”:”2.0”,”result”:0}
04155     *
04156     * \endenglish
04157     */
04158    int addGroupInput(const std::string &name, int io_type,
04159                      const std::vector<int> &indexes);
04160
04161    /**
04162     * @ingroup IoControl
04163     * \chinese
04164     *
```

```
04165     *
04166     * @param name:
04167     *
04168     * @return    0
04169     *
04170     * @throws arcs::common_interface::AuboException
04171     *
04172     * @par Python
04173     * deleteGroupInput(self: pyaubo_sdk.IoControl, arg0: str) -> int
04174     *
04175     * @par Lua
04176     * deleteGroupInput(name: string) -> nil
04177     *
04178     * @par JSON-RPC
04179     * {"jsonrpc":"2.0","method":"rob1.IoControl.deleteGroupInput","params":["gi_state"],"id":1}
04180     *
04181     * @par JSON-RPC
04182     * {"id":1,"jsonrpc":"2.0","result":0}
04183     *
04184     * \endchinese
04185     * \english
04186     * Delete a grouped digital input.
04187     *
04188     * @param name: Group name.
04189     *
04190     * @return Returns 0 on success; error code on failure.
04191     *
04192     * @throws arcs::common_interface::AuboException
04193     *
04194     * @par Python function prototype
04195     * deleteGroupInput(self: pyaubo_sdk.IoControl, arg0: str) -> int
04196     *
04197     * @par Lua function prototype
04198     * deleteGroupInput(name: string) -> nil
04199     *
04200     * @par JSON-RPC request example
04201     * {"jsonrpc":"2.0","method":"rob1.IoControl.deleteGroupInput","params":["gi_state"],"id":1}
04202     *
04203     * @par JSON-RPC response example
04204     * {"id":1,"jsonrpc":"2.0","result":0}
04205     *
04206     * \endenglish
04207     */
04208    int deleteGroupInput(const std::string &name);
04209
04210    /**
04211     * @ingroup IoControl
04212     * \chinese
04213     *
04214     *
04215     *
04216     *
04217     * @param name:
04218     *
04219     * @return
04220     *
04221     * @throws arcs::common_interface::AuboException
04222     *
04223     * @par Python
04224     * getGroupInput(self: pyaubo_sdk.IoControl, arg0: str) -> int
04225     *
04226     * @par Lua
04227     * getGroupInput(name: string) -> number
04228     *
04229     * @par Lua
04230     * value = getGroupInput("gi_state")
04231     *
04232     * @par JSON-RPC
04233     * {"jsonrpc":"2.0","method":"rob1.IoControl.getGroupInput","params":["gi_state"],"id":1}
04234     *
04235     * @par JSON-RPC
04236     * {"id":1,"jsonrpc":"2.0","result":13}
04237     *
04238     * \endchinese
04239     * \english
04240     * Get the grouped digital input value.
04241     *
04242     * Looks up the named grouped digital input, reads the group's members, and
04243     * combines them into an integer value.
04244     *
04245     * @param name: Group name.
04246     *
04247     * @return Returns the combined grouped digital input value.
04248     *
04249     * @throws arcs::common_interface::AuboException
04250     *
04251     * @par Python function prototype
```

```
04252     * getGroupInput(self: pyaubo_sdk.IoControl, arg0: str) -> int
04253     *
04254     * @par Lua function prototype
04255     * getGroupInput(name: string) -> number
04256     *
04257     * @par Lua example
04258     * value = getGroupInput("gi_state")
04259     *
04260     * @par JSON-RPC request example
04261     * {"jsonrpc":"2.0","method":"rob1.IoControl.getGroupInput","params":["gi_state"],"id":1}
04262     *
04263     * @par JSON-RPC response example
04264     * {"id":1,"jsonrpc":"2.0","result":13}
04265     *
04266     * \endenglish
04267     */
04268    uint32_t getGroupInput(const std::string &name);
04269
04270    /**
04271     * @ingroup IoControl
04272     * \chinese
04273     *
04274     *
04275     * @param index:   IO       0
04276     *   0
04277     *
04278     * @return     true;     false
04279     *
04280     * @throws arcs::common_interface::AuboException
04281     *
04282     * @par Python
04283     * getToolDigitalInput(self: pyaubo_sdk.IoControl, arg0: int) -> bool
04284     *
04285     * @par Lua
04286     * getToolDigitalInput(index: number) -> boolean
04287     *
04288     * @par Lua
04289     * status = getToolDigitalInput(0)
04290     *
04291     * @par JSON-RPC
04292     * {"jsonrpc":"2.0","method":"rob1.IoControl.getToolDigitalInput","params":[0],"id":1}
04293     *
04294     * @par JSON-RPC
04295     * {"id":1,"jsonrpc":"2.0","result":false}
04296     *
04297     * \endchinese
04298     * \english
04299     * Get the value of tool digital input.
04300     *
04301     * @param index: Indicates the IO pin, starting from 0.
04302     * For example, 0 means the first pin.
04303     *
04304     * @return Returns true for high level; false for low level.
04305     *
04306     * @throws arcs::common_interface::AuboException
04307     *
04308     * @par Python function prototype
04309     * getToolDigitalInput(self: pyaubo_sdk.IoControl, arg0: int) -> bool
04310     *
04311     * @par Lua function prototype
04312     * getToolDigitalInput(index: number) -> boolean
04313     *
04314     * @par Lua example
04315     * status = getToolDigitalInput(0)
04316     *
04317     * @par JSON-RPC request example
04318     * {"jsonrpc":"2.0","method":"rob1.IoControl.getToolDigitalInput","params":[0],"id":1}
04319     *
04320     * @par JSON-RPC response example
04321     * {"id":1,"jsonrpc":"2.0","result":false}
04322     *
04323     * \endenglish
04324     */
04325    bool getToolDigitalInput(int index);
04326
04327    /**
04328     * @ingroup IoControl
04329     * \chinese
04330     *
04331     *
04332     * @return
04333     *
04334     * @throws arcs::common_interface::AuboException
04335     *
04336     * @par Python
04337     * getToolDigitalInputs(self: pyaubo_sdk.IoControl) -> int
04338     *
```

```
04339     * @par Lua
04340     * getToolDigitalInputs() -> number
04341     *
04342     * @par Lua
04343     * num = getToolDigitalInputs()
04344     *
04345     * @par JSON-RPC
04346     * {"jsonrpc":"2.0","method":"rob1.IoControl.getToolDigitalInputs","params":[],"id":1}
04347     *
04348     * @par JSON-RPC
04349     * {"id":1,"jsonrpc":"2.0","result":0}
04350     *
04351     * \endchinese
04352     * \english
04353     * Get all tool digital input values.
04354     *
04355     * @return Returns all tool digital input values.
04356     *
04357     * @throws arcs::common_interface::AuboException
04358     *
04359     * @par Python function prototype
04360     * getToolDigitalInputs(self: pyaubo_sdk.IoControl) -> int
04361     *
04362     * @par Lua function prototype
04363     * getToolDigitalInputs() -> number
04364     *
04365     * @par Lua example
04366     * num = getToolDigitalInputs()
04367     *
04368     * @par JSON-RPC request example
04369     * {"jsonrpc":"2.0","method":"rob1.IoControl.getToolDigitalInputs","params":[],"id":1}
04370     *
04371     * @par JSON-RPC response example
04372     * {"id":1,"jsonrpc":"2.0","result":0}
04373     *
04374     * \endenglish
04375     */
04376    uint32_t getToolDigitalInputs();
04377
04378    /**
04379     * @ingroup IoControl
04380     * \chinese
04381     *
04382     *
04383     * @note      IO
04384     *
04385     * @param index:   IO       0
04386     *   0
04387     *
04388     * @return     true;     false
04389     *
04390     * @throws arcs::common_interface::AuboException
04391     *
04392     * @par Python
04393     * getConfigurableDigitalInput(self: pyaubo_sdk.IoControl, arg0: int) ->
04394     * bool
04395     *
04396     * @par Lua
04397     * getConfigurableDigitalInput(index: number) -> boolean
04398     *
04399     * @par Lua
04400     * status = getConfigurableDigitalInput(0)
04401     *
04402     * @par JSON-RPC
04403     * {"jsonrpc":"2.0","method":"rob1.IoControl.getConfigurableDigitalInput","params":[0],"id":1}
04404     *
04405     * @par JSON-RPC
04406     * {"id":1,"jsonrpc":"2.0","result":false}
04407     *
04408     * \endchinese
04409     * \english
04410     * Get the value of configurable digital input.
04411     *
04412     * @note Can be used to get the value of safety IO.
04413     *
04414     * @param index: Indicates the IO pin, starting from 0.
04415     * For example, 0 means the first pin.
04416     *
04417     * @return Returns true for high level; false for low level.
04418     *
04419     * @throws arcs::common_interface::AuboException
04420     *
04421     * @par Python function prototype
04422     * getConfigurableDigitalInput(self: pyaubo_sdk.IoControl, arg0: int) ->
04423     * bool
04424     *
04425     * @par Lua function prototype
```

```
04426     * getConfigurableDigitalInput(index: number) -> boolean
04427     *
04428     * @par Lua example
04429     * status = getConfigurableDigitalInput(0)
04430     *
04431     * @par JSON-RPC request example
04432     * {"jsonrpc":"2.0","method":"rob1.IoControl.getConfigurableDigitalInput","params":[0],"id":1}
04433     *
04434     * @par JSON-RPC response example
04435     * {"id":1,"jsonrpc":"2.0","result":false}
04436     *
04437     * \endenglish
04438     */
04439    bool getConfigurableDigitalInput(int index);
04440
04441    /**
04442     * @ingroup IoControl
04443     * \chinese
04444     *
04445     *
04446     * @note      IO
04447     *
04448     * @return        \n
04449     *      2863267846 ,  2   10101010101010100000000000000110
04450     *  16
04451     *      0    ,      1        \n
04452     * 1     0
04453     *
04454     * @throws arcs::common_interface::AuboException
04455     *
04456     * @par Python
04457     * getConfigurableDigitalInputs(self: pyaubo_sdk.IoControl) -> int
04458     *
04459     * @par Lua
04460     * getConfigurableDigitalInputs() -> number
04461     *
04462     * @par Lua
04463     * num = getConfigurableDigitalInputs()
04464     *
04465     * @par JSON-RPC
04466     * {"jsonrpc":"2.0","method":"rob1.IoControl.getConfigurableDigitalInputs","params":[],"id":1}
04467     *
04468     * @par JSON-RPC
04469     * {"id":1,"jsonrpc":"2.0","result":0}
04470     *
04471     * \endchinese
04472     * \english
04473     * Get all configurable digital input values.
04474     *
04475     * @note Can be used to get the value of safety IO.
04476     *
04477     * @return All configurable digital input values.\n
04478     * For example, if the return value is 2863267846, its binary representation
04479     * is 10101010101010100000000000000110. The lower 16 bits represent the
04480     * status of all input pins, the least significant bit indicates the input
04481     * status of pin 0, the second least significant bit indicates pin 1, and so
04482     * on.\n 1 means high level, 0 means low level.
04483     *
04484     * @throws arcs::common_interface::AuboException
04485     *
04486     * @par Python function prototype
04487     * getConfigurableDigitalInputs(self: pyaubo_sdk.IoControl) -> int
04488     *
04489     * @par Lua function prototype
04490     * getConfigurableDigitalInputs() -> number
04491     *
04492     * @par Lua example
04493     * num = getConfigurableDigitalInputs()
04494     *
04495     * @par JSON-RPC request example
04496     * {"jsonrpc":"2.0","method":"rob1.IoControl.getConfigurableDigitalInputs","params":[],"id":1}
04497     *
04498     * @par JSON-RPC response example
04499     * {"id":1,"jsonrpc":"2.0","result":0}
04500     *
04501     * \endenglish
04502     */
04503    uint32_t getConfigurableDigitalInputs();
04504
04505    /**
04506     * @ingroup IoControl
04507     * \chinese
04508     *
04509     *
04510     * @param index:   IO       0
04511     *    0
04512     *
```

```
04513     * @return     true;     false
04514     *
04515     * @throws arcs::common_interface::AuboException
04516     *
04517     * @par Python
04518     * getStandardDigitalOutput(self: pyaubo_sdk.IoControl, arg0: int) -> bool
04519     *
04520     * @par Lua
04521     * getStandardDigitalOutput(index: number) -> boolean
04522     *
04523     * @par Lua
04524     * status = getStandardDigitalOutput(0)
04525     *
04526     * @par JSON-RPC
04527     * {"jsonrpc":"2.0","method":"rob1.IoControl.getStandardDigitalOutput","params":[0],"id":1}
04528     *
04529     * @par JSON-RPC
04530     * {"id":1,"jsonrpc":"2.0","result":true}
04531     *
04532     * \endchinese
04533     * \english
04534     * Get the value of a standard digital output.
04535     *
04536     * @param index: Indicates the IO pin, starting from 0.
04537     * For example, 0 means the first pin.
04538     *
04539     * @return Returns true for high level; false for low level.
04540     *
04541     * @throws arcs::common_interface::AuboException
04542     *
04543     * @par Python function prototype
04544     * getStandardDigitalOutput(self: pyaubo_sdk.IoControl, arg0: int) -> bool
04545     *
04546     * @par Lua function prototype
04547     * getStandardDigitalOutput(index: number) -> boolean
04548     *
04549     * @par Lua example
04550     * status = getStandardDigitalOutput(0)
04551     *
04552     * @par JSON-RPC request example
04553     * {"jsonrpc":"2.0","method":"rob1.IoControl.getStandardDigitalOutput","params":[0],"id":1}
04554     *
04555     * @par JSON-RPC response example
04556     * {"id":1,"jsonrpc":"2.0","result":true}
04557     *
04558     * \endenglish
04559     */
04560    bool getStandardDigitalOutput(int index);
04561
04562    /**
04563     * @ingroup IoControl
04564     * \chinese
04565     *
04566     *
04567     * @return          \n
04568     *       2863267846 ,  2    10101010101010100000000000000110
04569     *  16
04570     *     DI00     ,      DI01          \n
04571     * 1       0      .
04572     *
04573     * @throws arcs::common_interface::AuboException
04574     *
04575     * @par Python
04576     * getStandardDigitalOutputs(self: pyaubo_sdk.IoControl) -> int
04577     *
04578     * @par Lua
04579     * getStandardDigitalOutputs() -> number
04580     *
04581     * @par Lua
04582     * num = getStandardDigitalOutputs()
04583     *
04584     * @par JSON-RPC
04585     * {"jsonrpc":"2.0","method":"rob1.IoControl.getStandardDigitalOutputs","params":[],"id":1}
04586     *
04587     * @par JSON-RPC
04588     * {"id":1,"jsonrpc":"2.0","result":69}
04589     *
04590     * \endchinese
04591     * \english
04592     * Get all standard digital output values.
04593     *
04594     * @return All standard digital output values.\n
04595     * For example, if the return value is 2863267846, its binary representation
04596     * is 10101010101010100000000000000110. The lower 16 bits represent the
04597     * status of all standard digital outputs, the least significant bit
04598     * indicates the output status of DO00, the second least significant bit
04599     * indicates DO01, and so on.\n 1 means high level, 0 means low level.
```

```
04600     *
04601     * @throws arcs::common_interface::AuboException
04602     *
04603     * @par Python function prototype
04604     * getStandardDigitalOutputs(self: pyaubo_sdk.IoControl) -> int
04605     *
04606     * @par Lua function prototype
04607     * getStandardDigitalOutputs() -> number
04608     *
04609     * @par Lua example
04610     * num = getStandardDigitalOutputs()
04611     *
04612     * @par JSON-RPC request example
04613     * {"jsonrpc":"2.0","method":"rob1.IoControl.getStandardDigitalOutputs","params":[],"id":1}
04614     *
04615     * @par JSON-RPC response example
04616     * {"id":1,"jsonrpc":"2.0","result":69}
04617     *
04618     * \endenglish
04619     */
04620    uint32_t getStandardDigitalOutputs();
04621
04622    /**
04623     * @ingroup IoControl
04624     * \chinese
04625     *
04626     *
04627     *
04628     *
04629     * @param name:
04630     *
04631     * @return
04632     *
04633     * @throws arcs::common_interface::AuboException
04634     *
04635     * @par Python
04636     * getGroupOutput(self: pyaubo_sdk.IoControl, arg0: str) -> int
04637     *
04638     * @par Lua
04639     * getGroupOutput(name: string) -> number
04640     *
04641     * @par Lua
04642     * value = getGroupOutput("go_state")
04643     *
04644     * @par JSON-RPC
04645     * {"jsonrpc":"2.0","method":"rob1.IoControl.getGroupOutput","params":["go_state"],"id":1}
04646     *
04647     * @par JSON-RPC
04648     * {"id":1,"jsonrpc":"2.0","result":13}
04649     *
04650     * \endchinese
04651     * \english
04652     * Get the grouped digital output value.
04653     *
04654     * Looks up the named grouped digital output, reads the group's members,
04655     * and combines them into an integer value.
04656     *
04657     * @param name: Group name.
04658     *
04659     * @return Returns the combined grouped digital output value.
04660     *
04661     * @throws arcs::common_interface::AuboException
04662     *
04663     * @par Python function prototype
04664     * getGroupOutput(self: pyaubo_sdk.IoControl, arg0: str) -> int
04665     *
04666     * @par Lua function prototype
04667     * getGroupOutput(name: string) -> number
04668     *
04669     * @par Lua example
04670     * value = getGroupOutput("go_state")
04671     *
04672     * @par JSON-RPC request example
04673     * {"jsonrpc":"2.0","method":"rob1.IoControl.getGroupOutput","params":["go_state"],"id":1}
04674     *
04675     * @par JSON-RPC response example
04676     * {"id":1,"jsonrpc":"2.0","result":13}
04677     *
04678     * \endenglish
04679     */
04680    uint32_t getGroupOutput(const std::string &name);
04681
04682    /**
04683     * @ingroup IoControl
04684     * \chinese
04685     *
04686     *
```

```
04687     * @param index:   IO       0
04688     *   0
04689     *
04690     * @return     true;     false
04691     *
04692     * @throws arcs::common_interface::AuboException
04693     *
04694     * @par Python
04695     * getToolDigitalOutput(self: pyaubo_sdk.IoControl, arg0: int) -> bool
04696     *
04697     * @par Lua
04698     * getToolDigitalOutput(index: number) -> boolean
04699     *
04700     * @par Lua
04701     * status = getToolDigitalOutput(0)
04702     *
04703     * @par JSON-RPC
04704     * {"jsonrpc":"2.0","method":"rob1.IoControl.getToolDigitalOutput","params":[0],"id":1}
04705     *
04706     * @par JSON-RPC
04707     * {"id":1,"jsonrpc":"2.0","result":false}
04708     *
04709     * \endchinese
04710     * \english
04711     * Get the value of tool digital output.
04712     *
04713     * @param index: Indicates the IO pin, starting from 0.
04714     * For example, 0 means the first pin.
04715     *
04716     * @return Returns true for high level; false for low level.
04717     *
04718     * @throws arcs::common_interface::AuboException
04719     *
04720     * @par Python function prototype
04721     * getToolDigitalOutput(self: pyaubo_sdk.IoControl, arg0: int) -> bool
04722     *
04723     * @par Lua function prototype
04724     * getToolDigitalOutput(index: number) -> boolean
04725     *
04726     * @par Lua example
04727     * status = getToolDigitalOutput(0)
04728     *
04729     * @par JSON-RPC request example
04730     * {"jsonrpc":"2.0","method":"rob1.IoControl.getToolDigitalOutput","params":[0],"id":1}
04731     *
04732     * @par JSON-RPC response example
04733     * {"id":1,"jsonrpc":"2.0","result":false}
04734     *
04735     * \endenglish
04736     */
04737    bool getToolDigitalOutput(int index);
04738
04739    /**
04740     * @ingroup IoControl
04741     * \chinese
04742     *
04743     *
04744     * @return
04745     *
04746     * @throws arcs::common_interface::AuboException
04747     *
04748     * @par Python
04749     * getToolDigitalOutputs(self: pyaubo_sdk.IoControl) -> int
04750     *
04751     * @par Lua
04752     * getToolDigitalOutputs() -> number
04753     *
04754     * @par Lua
04755     * num = getToolDigitalOutputs()
04756     *
04757     * @par JSON-RPC
04758     * {"jsonrpc":"2.0","method":"rob1.IoControl.getToolDigitalOutputs","params":[],"id":1}
04759     *
04760     * @par JSON-RPC
04761     * {"id":1,"jsonrpc":"2.0","result":9}
04762     *
04763     * \endchinese
04764     * \english
04765     * Get all tool digital output values.
04766     *
04767     * @return All tool digital output values.
04768     *
04769     * @throws arcs::common_interface::AuboException
04770     *
04771     * @par Python function prototype
04772     * getToolDigitalOutputs(self: pyaubo_sdk.IoControl) -> int
04773     *
```

```
04774     * @par Lua function prototype
04775     * getToolDigitalOutputs() -> number
04776     *
04777     * @par Lua example
04778     * num = getToolDigitalOutputs()
04779     *
04780     * @par JSON-RPC request example
04781     * {”jsonrpc”:”2.0”,”method”:”rob1.IoControl.getToolDigitalOutputs”,”params”:[],”id”:1}
04782     *
04783     * @par JSON-RPC response example
04784     * {”id”:1,”jsonrpc”:”2.0”,”result”:9}
04785     *
04786     * \endenglish
04787     */
04788    uint32_t getToolDigitalOutputs();
04789
04790    /**
04791     * @ingroup IoControl
04792     * \chinese
04793     *
04794     *
04795     * @note      IO
04796     *
04797     * @param index:   IO       0
04798     *   0
04799     *
04800     * @return     true;     false
04801     *
04802     * @throws arcs::common_interface::AuboException
04803     *
04804     * @par Python
04805     * getConfigurableDigitalOutput(self: pyaubo_sdk.IoControl, arg0: int) ->
04806     * bool
04807     *
04808     * @par Lua
04809     * getConfigurableDigitalOutput(index: number) -> boolean
04810     *
04811     * @par Lua
04812     * status = getConfigurableDigitalOutput(0)
04813     *
04814     * @par JSON-RPC
04815     * {”jsonrpc”:”2.0”,”method”:”rob1.IoControl.getConfigurableDigitalOutput”,”params”:[0],”id”:1}
04816     *
04817     * @par JSON-RPC
04818     * {”id”:1,”jsonrpc”:”2.0”,”result”:true}
04819     *
04820     * \endchinese
04821     * \english
04822     * Get the value of configurable digital output.
04823     *
04824     * @note Can be used to get the value of safety IO.
04825     *
04826     * @param index: Indicates the IO pin, starting from 0.
04827     * For example, 0 means the first pin.
04828     *
04829     * @return Returns true for high level; false for low level.
04830     *
04831     * @throws arcs::common_interface::AuboException
04832     *
04833     * @par Python function prototype
04834     * getConfigurableDigitalOutput(self: pyaubo_sdk.IoControl, arg0: int) ->
04835     * bool
04836     *
04837     * @par Lua function prototype
04838     * getConfigurableDigitalOutput(index: number) -> boolean
04839     *
04840     * @par Lua example
04841     * status = getConfigurableDigitalOutput(0)
04842     *
04843     * @par JSON-RPC request example
04844     * {”jsonrpc”:”2.0”,”method”:”rob1.IoControl.getConfigurableDigitalOutput”,”params”:[0],”id”:1}
04845     *
04846     * @par JSON-RPC response example
04847     * {”id”:1,”jsonrpc”:”2.0”,”result”:true}
04848     *
04849     * \endenglish
04850     */
04851    bool getConfigurableDigitalOutput(int index);
04852
04853    /**
04854     * @ingroup IoControl
04855     * \chinese
04856     *
04857     *
04858     * @note      IO
04859     *
04860     * @return       \n
```

```
04861     *       2863267846 , 2    10101010101010100000000000000110
04862     *  16
04863     *       0     ,       1         \n
04864     * 1      0      .
04865     *
04866     * @throws arcs::common_interface::AuboException
04867     *
04868     * @par Python
04869     * getConfigurableDigitalOutputs(self: pyaubo_sdk.IoControl) -> int
04870     *
04871     * @par Lua
04872     * getConfigurableDigitalOutputs() -> number
04873     *
04874     * @par Lua
04875     * num = getConfigurableDigitalOutputs()
04876     *
04877     * @par JSON-RPC
04878     * {”jsonrpc”:”2.0”,”method”:”rob1.IoControl.getConfigurableDigitalOutputs”,”params”:[],”id”:1}
04879     *
04880     * @par JSON-RPC
04881     * {”id”:1,”jsonrpc”:”2.0”,”result”:1}
04882     *
04883     * \endchinese
04884     * \english
04885     * Get all configurable digital output values.
04886     *
04887     * @note Can be used to get the value of safety IO.
04888     *
04889     * @return All configurable digital output values.\n
04890     * For example, if the return value is 2863267846, its binary representation
04891     * is 10101010101010100000000000000110. The lower 16 bits represent the
04892     * status of all output pins, the least significant bit indicates the output
04893     * status of pin 0, the second least significant bit indicates pin 1, and so
04894     * on.\n 1 means high level, 0 means low level.
04895     *
04896     * @throws arcs::common_interface::AuboException
04897     *
04898     * @par Python function prototype
04899     * getConfigurableDigitalOutputs(self: pyaubo_sdk.IoControl) -> int
04900     *
04901     * @par Lua function prototype
04902     * getConfigurableDigitalOutputs() -> number
04903     *
04904     * @par Lua example
04905     * num = getConfigurableDigitalOutputs()
04906     *
04907     * @par JSON-RPC request example
04908     * {”jsonrpc”:”2.0”,”method”:”rob1.IoControl.getConfigurableDigitalOutputs”,”params”:[],”id”:1}
04909     *
04910     * @par JSON-RPC response example
04911     * {”id”:1,”jsonrpc”:”2.0”,”result”:1}
04912     *
04913     * \endenglish
04914     */
04915    uint32_t getConfigurableDigitalOutputs();
04916
04917    /**
04918     * @ingroup IoControl
04919     * \chinese
04920     *
04921     *
04922     * @param index:   IO       0
04923     *   0
04924     *
04925     * @return
04926     *
04927     * @throws arcs::common_interface::AuboException
04928     *
04929     * @par Python
04930     * getStandardAnalogInput(self: pyaubo_sdk.IoControl, arg0: int) -> float
04931     *
04932     * @par Lua
04933     * getStandardAnalogInput(index: number) -> number
04934     *
04935     * @par Lua
04936     * getStandardAnalogInput(0)
04937     *
04938     * @par JSON-RPC
04939     * {”jsonrpc”:”2.0”,”method”:”rob1.IoControl.getStandardAnalogInput”,”params”:[0],”id”:1}
04940     *
04941     * @par JSON-RPC
04942     * {”id”:1,”jsonrpc”:”2.0”,”result”:0.0}
04943     *
04944     * \endchinese
04945     * \english
04946     * Get the value of standard analog input.
04947     *
```

```
04948     * @param index: Indicates the IO pin, starting from 0.
04949     * For example, 0 means the first pin.
04950     *
04951     * @return Standard analog input value.
04952     *
04953     * @throws arcs::common_interface::AuboException
04954     *
04955     * @par Python function prototype
04956     * getStandardAnalogInput(self: pyaubo_sdk.IoControl, arg0: int) -> float
04957     *
04958     * @par Lua function prototype
04959     * getStandardAnalogInput(index: number) -> number
04960     *
04961     * @par Lua example
04962     * getStandardAnalogInput(0)
04963     *
04964     * @par JSON-RPC request example
04965     * {"jsonrpc":"2.0","method":"rob1.IoControl.getStandardAnalogInput","params":[0],"id":1}
04966     *
04967     * @par JSON-RPC response example
04968     * {"id":1,"jsonrpc":"2.0","result":0.0}
04969     *
04970     * \endenglish
04971     */
04972    double getStandardAnalogInput(int index);
04973
04974    /**
04975     * @ingroup IoControl
04976     * \chinese
04977     *
04978     *
04979     * @param index:   IO       0
04980     *   0
04981     *
04982     * @return
04983     *
04984     * @throws arcs::common_interface::AuboException
04985     *
04986     * @par Python
04987     * getToolAnalogInput(self: pyaubo_sdk.IoControl, arg0: int) -> float
04988     *
04989     * @par Lua
04990     * getToolAnalogInput(index: number) -> number
04991     *
04992     * @par Lua
04993     * num = getToolAnalogInput(0)
04994     *
04995     * @par JSON-RPC
04996     * {"jsonrpc":"2.0","method":"rob1.IoControl.getToolAnalogInput","params":[0],"id":1}
04997     *
04998     * @par JSON-RPC
04999     * {"id":1,"jsonrpc":"2.0","result":0.0}
05000     *
05001     * \endchinese
05002     * \english
05003     * Get the value of tool analog input.
05004     *
05005     * @param index: Indicates the IO pin, starting from 0.
05006     * For example, 0 means the first pin.
05007     *
05008     * @return Tool analog input value.
05009     *
05010     * @throws arcs::common_interface::AuboException
05011     *
05012     * @par Python function prototype
05013     * getToolAnalogInput(self: pyaubo_sdk.IoControl, arg0: int) -> float
05014     *
05015     * @par Lua function prototype
05016     * getToolAnalogInput(index: number) -> number
05017     *
05018     * @par Lua example
05019     * num = getToolAnalogInput(0)
05020     *
05021     * @par JSON-RPC request example
05022     * {"jsonrpc":"2.0","method":"rob1.IoControl.getToolAnalogInput","params":[0],"id":1}
05023     *
05024     * @par JSON-RPC response example
05025     * {"id":1,"jsonrpc":"2.0","result":0.0}
05026     *
05027     * \endenglish
05028     */
05029    double getToolAnalogInput(int index);
05030
05031    /**
05032     * @ingroup IoControl
05033     * \chinese
05034     *
```

```
05035     *
05036     * @param index:   IO       0
05037     *   0
05038     *
05039     * @return
05040     *
05041     * @throws arcs::common_interface::AuboException
05042     *
05043     * @par Python
05044     * getStandardAnalogOutput(self: pyaubo_sdk.IoControl, arg0: int) -> float
05045     *
05046     * @par Lua
05047     * getStandardAnalogOutput(index: number) -> number
05048     *
05049     * @par Lua
05050     * num = getStandardAnalogOutput(0)
05051     *
05052     * @par JSON-RPC
05053     * {"jsonrpc":"2.0","method":"rob1.IoControl.getStandardAnalogOutput","params":[0],"id":1}
05054     *
05055     * @par JSON-RPC
05056     * {"id":1,"jsonrpc":"2.0","result":0.0}
05057     *
05058     * \endchinese
05059     * \english
05060     * Get the value of standard analog output.
05061     *
05062     * @param index: Indicates the IO pin, starting from 0.
05063     * For example, 0 means the first pin.
05064     *
05065     * @return Standard analog output value.
05066     *
05067     * @throws arcs::common_interface::AuboException
05068     *
05069     * @par Python function prototype
05070     * getStandardAnalogOutput(self: pyaubo_sdk.IoControl, arg0: int) -> float
05071     *
05072     * @par Lua function prototype
05073     * getStandardAnalogOutput(index: number) -> number
05074     *
05075     * @par Lua example
05076     * num = getStandardAnalogOutput(0)
05077     *
05078     * @par JSON-RPC request example
05079     * {"jsonrpc":"2.0","method":"rob1.IoControl.getStandardAnalogOutput","params":[0],"id":1}
05080     *
05081     * @par JSON-RPC response example
05082     * {"id":1,"jsonrpc":"2.0","result":0.0}
05083     *
05084     * \endenglish
05085     */
05086    double getStandardAnalogOutput(int index);
05087
05088    /**
05089     * @ingroup IoControl
05090     * \chinese
05091     *
05092     *
05093     * @param index:   IO       0
05094     *   0
05095     *
05096     * @return
05097     *
05098     * @throws arcs::common_interface::AuboException
05099     *
05100     * @par Python
05101     * getToolAnalogOutput(self: pyaubo_sdk.IoControl, arg0: int) -> float
05102     *
05103     * @par Lua
05104     * getToolAnalogOutput(index: number) -> number
05105     *
05106     * @par Lua
05107     * num = getToolAnalogOutput(0)
05108     *
05109     * @par JSON-RPC
05110     * {"jsonrpc":"2.0","method":"rob1.IoControl.getToolAnalogOutput","params":[0],"id":1}
05111     *
05112     * @par JSON-RPC
05113     * {"id":1,"jsonrpc":"2.0","result":0.0}
05114     *
05115     * \endchinese
05116     * \english
05117     * Get the value of tool analog output.
05118     *
05119     * @param index: Indicates the IO pin, starting from 0.
05120     * For example, 0 means the first pin.
05121     *
```

```
05122     * @return Tool analog output value.
05123     *
05124     * @throws arcs::common_interface::AuboException
05125     *
05126     * @par Python function prototype
05127     * getToolAnalogOutput(self: pyaubo_sdk.IoControl, arg0: int) -> float
05128     *
05129     * @par Lua function prototype
05130     * getToolAnalogOutput(index: number) -> number
05131     *
05132     * @par Lua example
05133     * num = getToolAnalogOutput(0)
05134     *
05135     * @par JSON-RPC request example
05136     * {"jsonrpc":"2.0","method":"rob1.IoControl.getToolAnalogOutput","params":[0],"id":1}
05137     *
05138     * @par JSON-RPC response example
05139     * {"id":1,"jsonrpc":"2.0","result":0.0}
05140     *
05141     * \endenglish
05142     */
05143    double getToolAnalogOutput(int index);
05144
05145    /**
05146     * @ingroup IoControl
05147     * \chinese
05148     *
05149     *
05150     * @return
05151     *
05152     * @throws arcs::common_interface::AuboException
05153     *
05154     * @par Python
05155     * getStaticLinkInputNum(self: pyaubo_sdk.IoControl) -> int
05156     *
05157     * @par Lua
05158     * getStaticLinkInputNum() -> number
05159     *
05160     * @par Lua
05161     * num = getStaticLinkInputNum()
05162     *
05163     * @par JSON-RPC
05164     * {"jsonrpc":"2.0","method":"rob1.IoControl.getStaticLinkInputNum","params":[],"id":1}
05165     *
05166     * @par JSON-RPC
05167     * {"id":1,"jsonrpc":"2.0","result":8}
05168     *
05169     * \endchinese
05170     * \english
05171     * Get the number of static link inputs.
05172     *
05173     * @return Number of static link inputs.
05174     *
05175     * @throws arcs::common_interface::AuboException
05176     *
05177     * @par Python function prototype
05178     * getStaticLinkInputNum(self: pyaubo_sdk.IoControl) -> int
05179     *
05180     * @par Lua function prototype
05181     * getStaticLinkInputNum() -> number
05182     *
05183     * @par Lua example
05184     * num = getStaticLinkInputNum()
05185     *
05186     * @par JSON-RPC request example
05187     * {"jsonrpc":"2.0","method":"rob1.IoControl.getStaticLinkInputNum","params":[],"id":1}
05188     *
05189     * @par JSON-RPC response example
05190     * {"id":1,"jsonrpc":"2.0","result":8}
05191     *
05192     * \endenglish
05193     */
05194    int getStaticLinkInputNum();
05195
05196    /**
05197     * @ingroup IoControl
05198     * \chinese
05199     *
05200     *
05201     * @return
05202     *
05203     * @throws arcs::common_interface::AuboException
05204     *
05205     * @par Python
05206     * getStaticLinkOutputNum(self: pyaubo_sdk.IoControl) -> int
05207     *
05208     * @par Lua
```

```
05209     * getStaticLinkOutputNum() -> number
05210     *
05211     * @par Lua
05212     * num = getStaticLinkOutputNum()
05213     *
05214     * @par JSON-RPC
05215     * {”jsonrpc”:”2.0”,”method”:”rob1.IoControl.getStaticLinkOutputNum”,”params”:[],”id”:1}
05216     *
05217     * @par JSON-RPC
05218     * {”id”:1,”jsonrpc”:”2.0”,”result”:0}
05219     *
05220     * \endchinese
05221     * \english
05222     * Get the number of static link outputs.
05223     *
05224     * @return Number of static link outputs.
05225     *
05226     * @throws arcs::common_interface::AuboException
05227     *
05228     * @par Python function prototype
05229     * getStaticLinkOutputNum(self: pyaubo_sdk.IoControl) -> int
05230     *
05231     * @par Lua function prototype
05232     * getStaticLinkOutputNum() -> number
05233     *
05234     * @par Lua example
05235     * num = getStaticLinkOutputNum()
05236     *
05237     * @par JSON-RPC request example
05238     * {”jsonrpc”:”2.0”,”method”:”rob1.IoControl.getStaticLinkOutputNum”,”params”:[],”id”:1}
05239     *
05240     * @par JSON-RPC response example
05241     * {”id”:1,”jsonrpc”:”2.0”,”result”:0}
05242     *
05243     * \endenglish
05244     */
05245    int getStaticLinkOutputNum();
05246
05247    /**
05248     * @ingroup IoControl
05249     * \chinese
05250     *
05251     *
05252     * @return       \n
05253     *       2863267846 ,  2    10101010101010100000000000000110
05254     *  16
05255     *       0     ,       1         \n
05256     * 1      0      .
05257     *
05258     * @throws arcs::common_interface::AuboException
05259     *
05260     * @par Python
05261     * getStaticLinkInputs(self: pyaubo_sdk.IoControl) -> int
05262     *
05263     * @par Lua
05264     * getStaticLinkInputs() -> number
05265     *
05266     * @par Lua
05267     * num = getStaticLinkInputs()
05268     *
05269     * @par JSON-RPC
05270     * {”jsonrpc”:”2.0”,”method”:”rob1.IoControl.getStaticLinkInputs”,”params”:[],”id”:1}
05271     *
05272     * @par JSON-RPC
05273     * {”id”:1,”jsonrpc”:”2.0”,”result”:0}
05274     *
05275     * \endchinese
05276     * \english
05277     * Get all static link input values.
05278     *
05279     * @return All static link input values.\n
05280     * For example, if the return value is 2863267846, its binary representation
05281     * is 10101010101010100000000000000110. The lower 16 bits represent the
05282     * status of all static link inputs, the least significant bit indicates the
05283     * input status of pin 0, the second least significant bit indicates pin 1,
05284     * and so on.\n 1 means high level, 0 means low level.
05285     *
05286     * @throws arcs::common_interface::AuboException
05287     *
05288     * @par Python function prototype
05289     * getStaticLinkInputs(self: pyaubo_sdk.IoControl) -> int
05290     *
05291     * @par Lua function prototype
05292     * getStaticLinkInputs() -> number
05293     *
05294     * @par Lua example
05295     * num = getStaticLinkInputs()
```

```
05296     *
05297     * @par JSON-RPC request example
05298     * {”jsonrpc”:”2.0”,”method”:”rob1.IoControl.getStaticLinkInputs”,”params”:[],”id”:1}
05299     *
05300     * @par JSON-RPC response example
05301     * {”id”:1,”jsonrpc”:”2.0”,”result”:0}
05302     *
05303     * \endenglish
05304     */
05305    uint32_t getStaticLinkInputs();
05306
05307    /**
05308     * @ingroup IoControl
05309     * \chinese
05310     *
05311     *
05312     * @return          \n
05313     *       2863267846 ,  2   10101010101010100000000000000110
05314     *  16
05315     *       0     ,       1         \n
05316     * 1      0      .
05317     *
05318     * @throws arcs::common_interface::AuboException
05319     *
05320     * @par Python
05321     * getStaticLinkOutputs(self: pyaubo_sdk.IoControl) -> int
05322     *
05323     * @par Lua
05324     * getStaticLinkOutputs() -> number
05325     *
05326     * @par Lua
05327     * num = getStaticLinkOutputs()
05328     *
05329     * @par JSON-RPC
05330     * {”jsonrpc”:”2.0”,”method”:”rob1.IoControl.getStaticLinkOutputs”,”params”:[],”id”:1}
05331     *
05332     * @par JSON-RPC
05333     * {”id”:1,”jsonrpc”:”2.0”,”result”:0}
05334     *
05335     * \endchinese
05336     * \english
05337     * Get all static link output values.
05338     *
05339     * @return Returns all static link output values.\n
05340     * For example, if the return value is 2863267846, its binary representation
05341     * is 10101010101010100000000000000110. The lower 16 bits represent the
05342     * status of all static link outputs, the least significant bit indicates
05343     * the output status of pin 0, the second least significant bit indicates
05344     * pin 1, and so on.\n 1 means high level, 0 means low level.
05345     *
05346     * @throws arcs::common_interface::AuboException
05347     *
05348     * @par Python function prototype
05349     * getStaticLinkOutputs(self: pyaubo_sdk.IoControl) -> int
05350     *
05351     * @par Lua function prototype
05352     * getStaticLinkOutputs() -> number
05353     *
05354     * @par Lua example
05355     * num = getStaticLinkOutputs()
05356     *
05357     * @par JSON-RPC request example
05358     * {”jsonrpc”:”2.0”,”method”:”rob1.IoControl.getStaticLinkOutputs”,”params”:[],”id”:1}
05359     *
05360     * @par JSON-RPC response example
05361     * {”id”:1,”jsonrpc”:”2.0”,”result”:0}
05362     *
05363     * \endenglish
05364     */
05365    uint32_t getStaticLinkOutputs();
05366
05367    /**
05368     * @ingroup IoControl
05369     * \chinese
05370     *
05371     *       0
05372     *
05373     * @return          true,     false
05374     *
05375     * @throws arcs::common_interface::AuboException
05376     *
05377     * @par JSON-RPC
05378     * {”jsonrpc”:”2.0”,”method”:”rob1.IoControl.hasEncoderSensor”,”params”:[],”id”:1}
05379     *
05380     * @par JSON-RPC
05381     * {”id”:1,”jsonrpc”:”2.0”,”result”:true}
05382     *
```

```
05383     * \endchinese
05384     * \english
05385     * Whether the robot is equipped with an encoder.
05386     * The integrated encoder number is 0.
05387     *
05388     * @return Returns true if the robot is equipped with an encoder, otherwise
05389     * false.
05390     *
05391     * @throws arcs::common_interface::AuboException
05392     *
05393     * @par JSON-RPC request example
05394     * {"jsonrpc":"2.0","method":"rob1.IoControl.hasEncoderSensor","params":[],"id":1}
05395     *
05396     * @par JSON-RPC response example
05397     * {"id":1,"jsonrpc":"2.0","result":true}
05398     *
05399     * \endenglish
05400     */
05401    bool hasEncoderSensor();
05402    /**
05403     * @ingroup IoControl
05404     * \chinese
05405     *
05406     *
05407     * @param type
05408     * 0-
05409     * 1-AB
05410     * 2-AB +Z
05411     * 3-AB
05412     * 4-AB  +Z
05413     *
05414     * @param range_id
05415     * 0  32        [-2147483648, 2147483647]
05416     * 1  8        [0, 255]
05417     * 2  16        [0, 65535]
05418     * 3  24        [0, 16777215]
05419     * 4  32        [0, 4294967295]
05420     *
05421     * @return   0
05422     * AUBO_NO_ACCESS
05423     * AUBO_BUSY
05424     * AUBO_BAD_STATE
05425     * -AUBO_INVL_ARGUMENT
05426     * -AUBO_BAD_STATE
05427     *
05428     * @throws arcs::common_interface::AuboException
05429     * \endchinese
05430     * \english
05431     * Set the decoding method of the integrated encoder.
05432     *
05433     * @param type
05434     * 0 - Disable encoder
05435     * 1 - AB quadrature
05436     * 2 - AB quadrature + Z
05437     * 3 - AB differential quadrature
05438     * 4 - AB differential quadrature + Z differential
05439     *
05440     * @param range_id
05441     * 0 is a 32-bit signed encoder, range [-2147483648, 2147483647]
05442     * 1 is an 8-bit unsigned encoder, range [0, 255]
05443     * 2 is a 16-bit unsigned encoder, range [0, 65535]
05444     * 3 is a 24-bit unsigned encoder, range [0, 16777215]
05445     * 4 is a 32-bit unsigned encoder, range [0, 4294967295]
05446     *
05447     * @return Returns 0 on success; error code on failure.
05448     * AUBO_NO_ACCESS
05449     * AUBO_BUSY
05450     * AUBO_BAD_STATE
05451     * -AUBO_INVL_ARGUMENT
05452     * -AUBO_BAD_STATE
05453     *
05454     * @throws arcs::common_interface::AuboException
05455     * \endenglish
05456     */
05457    int setEncDecoderType(int type, int range_id);
05458
05459    /**
05460     * @ingroup IoControl
05461     * \chinese
05462     *
05463     *
05464     * @param tick
05465     *
05466     * @return   0
05467     * AUBO_NO_ACCESS
05468     * AUBO_BUSY
05469     * AUBO_BAD_STATE
```

```
05470     * -AUBO_INVL_ARGUMENT
05471     * -AUBO_BAD_STATE
05472     *
05473     * @throws arcs::common_interface::AuboException
05474     * \endchinese
05475     * \english
05476     * Set the tick count of the integrated encoder.
05477     *
05478     * @param tick Tick count
05479     *
05480     * @return Returns 0 on success; error code on failure.
05481     * AUBO_NO_ACCESS
05482     * AUBO_BUSY
05483     * AUBO_BAD_STATE
05484     * -AUBO_INVL_ARGUMENT
05485     * -AUBO_BAD_STATE
05486     *
05487     * @throws arcs::common_interface::AuboException
05488     * \endenglish
05489     */
05490    int setEncTickCount(int tick);
05491
05492    /**
05493     * @ingroup IoControl
05494     * \chinese
05495     *
05496     *
05497     * @return    0
05498     * AUBO_NO_ACCESS
05499     * AUBO_BUSY
05500     * AUBO_BAD_STATE
05501     * -AUBO_BAD_STATE
05502     *
05503     * @throws arcs::common_interface::AuboException
05504     *
05505     * @par JSON-RPC
05506     * {"jsonrpc":"2.0","method":"rob1.IoControl.getEncDecoderType","params":[],"id":1}
05507     *
05508     * @par JSON-RPC
05509     * {"id":1,"jsonrpc":"2.0","result":0}
05510     *
05511     * \endchinese
05512     * \english
05513     * Get the decoder type of the encoder.
05514     *
05515     * @return Returns 0 on success; error code on failure.
05516     * AUBO_NO_ACCESS
05517     * AUBO_BUSY
05518     * AUBO_BAD_STATE
05519     * -AUBO_BAD_STATE
05520     *
05521     * @throws arcs::common_interface::AuboException
05522     *
05523     * @par JSON-RPC request example
05524     * {"jsonrpc":"2.0","method":"rob1.IoControl.getEncDecoderType","params":[],"id":1}
05525     *
05526     * @par JSON-RPC response example
05527     * {"id":1,"jsonrpc":"2.0","result":0}
05528     *
05529     * \endenglish
05530     */
05531    int getEncDecoderType();
05532
05533    /**
05534     * @ingroup IoControl
05535     * \chinese
05536     *
05537     *
05538     * @return    0
05539     * AUBO_NO_ACCESS
05540     * AUBO_BUSY
05541     * AUBO_BAD_STATE
05542     * -AUBO_BAD_STATE
05543     *
05544     * @throws arcs::common_interface::AuboException
05545     *
05546     * @par JSON-RPC
05547     * {"jsonrpc":"2.0","method":"rob1.IoControl.getEncTickCount","params":[],"id":1}
05548     *
05549     * @par JSON-RPC
05550     * {"id":1,"jsonrpc":"2.0","result":0}
05551     *
05552     * \endchinese
05553     * \english
05554     * Get the tick count
05555     *
05556     * @return Returns 0 on success; error code on failure.
```

```
05557     * AUBO_NO_ACCESS
05558     * AUBO_BUSY
05559     * AUBO_BAD_STATE
05560     * -AUBO_BAD_STATE
05561     *
05562     * @throws arcs::common_interface::AuboException
05563     *
05564     * @par JSON-RPC request example
05565     * {"jsonrpc":"2.0","method":"rob1.IoControl.getEncTickCount","params":[],"id":1}
05566     *
05567     * @par JSON-RPC response example
05568     * {"id":1,"jsonrpc":"2.0","result":0}
05569     *
05570     * \endenglish
05571     */
05572    int getEncTickCount();
05573
05574    /**
05575     * @ingroup IoControl
05576     * \chinese
05577     *
05578     *
05579     * @param delta_count
05580     *
05581     * @return    0
05582     * AUBO_NO_ACCESS
05583     * AUBO_BUSY
05584     * AUBO_BAD_STATE
05585     * -AUBO_BAD_STATE
05586     *
05587     * @throws arcs::common_interface::AuboException
05588     * \endchinese
05589     * \english
05590     * Prevent counting errors when the count exceeds the range
05591     *
05592     * @param delta_count
05593     *
05594     * @return Returns 0 on success; error code on failure.
05595     * AUBO_NO_ACCESS
05596     * AUBO_BUSY
05597     * AUBO_BAD_STATE
05598     * -AUBO_BAD_STATE
05599     *
05600     * @throws arcs::common_interface::AuboException
05601     * \endenglish
05602     */
05603    int unwindEncDeltaTickCount(int delta_count);
05604
05605    /**
05606     * @ingroup IoControl
05607     * \chinese
05608     *
05609     *
05610     * @return    true;    false
05611     *
05612     * @throws arcs::common_interface::AuboException
05613     *
05614     * @par Python
05615     * getToolButtonStatus() -> bool
05616     *
05617     * @par Lua
05618     * getToolButtonStatus() -> boolean
05619     *
05620     * @par Lua
05621     * status = getToolButtonStatus()
05622     *
05623     * @par JSON-RPC
05624     * {"jsonrpc":"2.0","method":"rob1.IoControl.getToolButtonStatus","params":[],"id":1}
05625     *
05626     * @par JSON-RPC
05627     * {"id":1,"jsonrpc":"2.0","result":false}
05628     *
05629     * \endchinese
05630     * \english
05631     * Get the status of the tool button.
05632     *
05633     * @return Returns true if pressed; otherwise false.
05634     *
05635     * @throws arcs::common_interface::AuboException
05636     *
05637     * @par Python function prototype
05638     * getToolButtonStatus() -> bool
05639     *
05640     * @par Lua function prototype
05641     * getToolButtonStatus() -> boolean
05642     *
05643     * @par Lua example
```

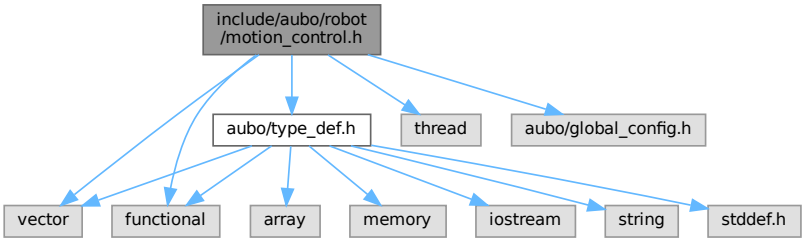
```
05644     * status = getToolButtonStatus()
05645     *
05646     * @par JSON-RPC request example
05647     * {”jsonrpc”:”2.0”,”method”:”rob1.IoControl.getToolButtonStatus”,”params”:[],”id”:1}
05648     *
05649     * @par JSON-RPC response example
05650     * {”id”:1,”jsonrpc”:”2.0”,”result”:false}
05651     *
05652     * \endenglish
05653     */
05654    bool getToolButtonStatus();
05655
05656    /**
05657     * @ingroup IoControl
05658     * \chinese
05659     *
05660     *
05661     * @note
05662     *
05663     * @return        \n
05664     *       2863267846 ,  2    10101010101010100000000000000110
05665     *  16
05666     *       0     ,       1         \n
05667     * 1      0
05668     *
05669     * @throws arcs::common_interface::AuboException
05670     *
05671     * @par Python
05672     * getHandleIoStatus(self: pyaubo_sdk.IoControl) -> int
05673     *
05674     * @par Lua
05675     * getHandleIoStatus() -> number
05676     *
05677     * @par Lua
05678     * num = getHandleIoStatus()
05679     *
05680     * @par JSON-RPC
05681     * {”jsonrpc”:”2.0”,”method”:”rob1.IoControl.getHandleIoStatus”,”params”:[],”id”:1}
05682     *
05683     * @par JSON-RPC
05684     * {”id”:1,”jsonrpc”:”2.0”,”result”:0}
05685     *
05686     * \endchinese
05687     * \english
05688     * Get the status of handle buttons.
05689     *
05690     * @note Get the status of handle buttons.
05691     *
05692     * @return All handle button input values.\n
05693     * For example, if the return value is 2863267846, its binary representation
05694     * is 10101010101010100000000000000110. The lower 16 bits represent the
05695     * status of all input pins, the least significant bit indicates the input
05696     * status of pin 0, the second least significant bit indicates pin 1, and so
05697     * on.\n 1 means high level, 0 means low level.
05698     *
05699     * @throws arcs::common_interface::AuboException
05700     *
05701     * @par Python function prototype
05702     * getHandleIoStatus(self: pyaubo_sdk.IoControl) -> int
05703     *
05704     * @par Lua function prototype
05705     * getHandleIoStatus() -> number
05706     *
05707     * @par Lua example
05708     * num = getHandleIoStatus()
05709     *
05710     * @par JSON-RPC request example
05711     * {”jsonrpc”:”2.0”,”method”:”rob1.IoControl.getHandleIoStatus”,”params”:[],”id”:1}
05712     *
05713     * @par JSON-RPC response example
05714     * {”id”:1,”jsonrpc”:”2.0”,”result”:0}
05715     *
05716     * \endenglish
05717     */
05718    uint32_t getHandleIoStatus();
05719
05720    /**
05721     * @ingroup IoControl
05722     * \chinese
05723     *
05724     *
05725     * @return type
05726     *
05727     * @throws arcs::common_interface::AuboException
05728     *
05729     * @par Python
05730     * getHandleType() -> int
```

```
05731     *
05732     * @par Lua
05733     * getHandleType() -> int
05734     *
05735     * @par Lua
05736     * int_num = getHandleType()
05737     *
05738     * @par JSON-RPC
05739     * {"jsonrpc":"2.0","method":"rob1.IoControl.getHandleType","params":[],"id":1}
05740     *
05741     * @par JSON-RPC
05742     * {"id":1,"jsonrpc":"2.0","result":0}
05743     *
05744     * \endchinese
05745     * \english
05746     * Get the handle type.
05747     *
05748     * @return type
05749     *
05750     * @throws arcs::common_interface::AuboException
05751     *
05752     * @par Python function prototype
05753     * getHandleType() -> int
05754     *
05755     * @par Lua function prototype
05756     * getHandleType() -> int
05757     *
05758     * @par Lua example
05759     * int_num = getHandleType()
05760     *
05761     * @par JSON-RPC request example
05762     * {"jsonrpc":"2.0","method":"rob1.IoControl.getHandleType","params":[],"id":1}
05763     *
05764     * @par JSON-RPC response example
05765     * {"id":1,"jsonrpc":"2.0","result":0}
05766     *
05767     * \endenglish
05768     */
05769    int getHandleType();
05770
05771 protected:
05772    void *d_;
05773 };
05774 using IoControlPtr = std::shared_ptr<IoControl>;
05775 } // namespace common_interface
05776 } // namespace arcs
05777
05778 #endif // AUBO_SDK_IO_CONTROL_INTERFACE_H
```

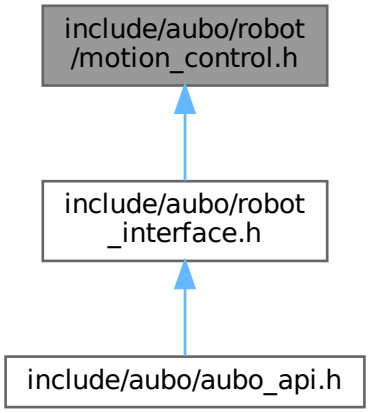
12.24 include/aubo/robot/motion_control.h File Reference

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() -> int
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() -> int
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     *
00368     * @param enable
00369     * @return    0;
00370     * AUBO_BAD_STATE
00371     * AUBO_BUSY
00372     * -AUBO_BAD_STATE
00373     *
00374     * @throws arcs::common_interface::AuboException
00375     *
00376     * @par Lua
00377     * speedFractionCritical1() -> int
00378     *
00379     * @par Lua
00380     * speedFractionCritical1(true)
00381     *
00382     * @par JSON-RPC
00383     * {"jsonrpc":"2.0","method":"rob1.MotionControl.speedFractionCritical1","params":[true],"id":1}
00384     *
00385     * @par JSON-RPC
00386     * {"id":1,"jsonrpc":"2.0","result":0}
00387     * \endchinese
00388     * \english
00389     * Speed fraction critical section. When enabled, the speed fraction is
00390     * forced to the currently effective speed fraction. When disabled, the
00391     * previous speed fraction is restored.
00392     *
00393     * @param enable
00394     * @return Returns 0 on success; otherwise returns an error code:
00395     * AUBO_BAD_STATE
00396     * AUBO_BUSY
00397     * -AUBO_BAD_STATE
00398     *
00399     * @throws arcs::common_interface::AuboException
00400     *
00401     * @par Lua function prototype
00402     * speedFractionCritical1() -> int
00403     *
00404     * @par Lua example
00405     * speedFractionCritical1(true)
00406     *
00407     * @par JSON-RPC request example
00408     * {"jsonrpc":"2.0","method":"rob1.MotionControl.speedFractionCritical1","params":[true],"id":1}
00409     *
00410     * @par JSON-RPC response example
00411     * {"id":1,"jsonrpc":"2.0","result":0}
00412     * \endenglish
00413     */
00414    int speedFractionCritical1(bool enable);
00415
00416    /**
00417     * @ingroup MotionControl
00418     * \chinese
00419     *
00420     *
00421     * @return           true;     false
00422     *
00423     * @throws arcs::common_interface::AuboException
00424     *
00425     * @par Lua
00426     * isSpeedFractionCritical() -> bool
00427     *
00428     * @par Lua
00429     * status = isSpeedFractionCritical()
00430     *
```

```
00431     * @par JSON-RPC
00432     * {"jsonrpc":"2.0","method":"rob1.MotionControl.isSpeedFractionCritical","params":[],"id":1}
00433     *
00434     * @par JSON-RPC
00435     * {"id":1,"jsonrpc":"2.0","result":true}
00436     * \endchinese
00437     * \english
00438     * Whether it is in the speed fraction critical section
00439     *
00440     * @return Returns true if in the speed fraction critical section; otherwise
00441     * returns false
00442     *
00443     * @throws arcs::common_interface::AuboException
00444     *
00445     * @par Lua function prototype
00446     * isSpeedFractionCritical() -> bool
00447     *
00448     * @par Lua example
00449     * status = isSpeedFractionCritical()
00450     *
00451     * @par JSON-RPC request example
00452     * {"jsonrpc":"2.0","method":"rob1.MotionControl.isSpeedFractionCritical","params":[],"id":1}
00453     *
00454     * @par JSON-RPC response example
00455     * {"id":1,"jsonrpc":"2.0","result":true}
00456     * \endenglish
00457     */
00458    bool isSpeedFractionCritical();
00459
00460    /**
00461     * @ingroup MotionControl
00462     * \chinese
00463     *
00464     *
00465     * @return      true;     false
00466     *
00467     * @throws arcs::common_interface::AuboException
00468     *
00469     * @par Lua
00470     * isBlending() -> bool
00471     *
00472     * @par Lua
00473     * status = isBlending()
00474     *
00475     * @par JSON-RPC
00476     * {"jsonrpc":"2.0","method":"rob1.MotionControl.isBlending","params":[],"id":1}
00477     *
00478     * @par JSON-RPC
00479     * {"id":1,"jsonrpc":"2.0","result":false}
00480     * \endchinese
00481     * \english
00482     * Whether it is in the blending area
00483     *
00484     * @return Returns true if in the blending area; otherwise returns false
00485     *
00486     * @throws arcs::common_interface::AuboException
00487     *
00488     * @par Lua function prototype
00489     * isBlending() -> bool
00490     *
00491     * @par Lua example
00492     * status = isBlending()
00493     *
00494     * @par JSON-RPC request example
00495     * {"jsonrpc":"2.0","method":"rob1.MotionControl.isBlending","params":[],"id":1}
00496     *
00497     * @par JSON-RPC response example
00498     * {"id":1,"jsonrpc":"2.0","result":false}
00499     * \endenglish
00500     */
00501    bool isBlending();
00502
00503    /**
00504     * @ingroup MotionControl
00505     * \chinese
00506     *
00507     *   pathOffsetSet  type=1
00508     * @param v
00509     * @param a
00510     * @return    0
00511     * AUBO_BUSY
00512     * AUBO_BAD_STATE
00513     * -AUBO_INVL_ARGUMENT
00514     * -AUBO_BAD_STATE
00515     *
00516     * @throws arcs::common_interface::AuboException
00517     *
```

```
00518     * @par Python
00519     * pathOffsetLimits(self: pyaubo_sdk.MotionControl, arg0: float, arg1:
00520     * float) -> int
00521     *
00522     * @par Lua
00523     * pathOffsetLimits(v: number, a: number) -> nil
00524     *
00525     * @par Lua
00526     * pathOffsetLimits(1.5,2.5)
00527     *
00528     * \endchinese
00529     * \english
00530     * Set the maximum speed and maximum acceleration for offset.
00531     * Only valid when type=1 in pathOffsetSet.
00532     * @param v Maximum speed
00533     * @param a Maximum acceleration
00534     * @return Returns 0 on success; otherwise returns an error code:
00535     * AUBO_BUSY
00536     * AUBO_BAD_STATE
00537     * -AUBO_INVL_ARGUMENT
00538     * -AUBO_BAD_STATE
00539     *
00540     * @throws arcs::common_interface::AuboException
00541     *
00542     * @par Python function prototype
00543     * pathOffsetLimits(self: pyaubo_sdk.MotionControl, arg0: float, arg1:
00544     * float) -> int
00545     *
00546     * @par Lua function prototype
00547     * pathOffsetLimits(v: number, a: number) -> nil
00548     *
00549     * @par Lua example
00550     * pathOffsetLimits(1.5,2.5)
00551     *
00552     * \endenglish
00553     */
00554    int pathOffsetLimits(double v, double a);
00555
00556    /**
00557     * @ingroup MotionControl
00558     * \chinese
00559     *
00560     *  pathOffsetSet  type=1
00561     * @param ref_coord      0-    1-TCP
00562     *
00563     * @throws arcs::common_interface::AuboException
00564     *
00565     * @par Python
00566     * pathOffsetCoordinate(self: pyaubo_sdk.MotionControl, arg0: int) -> float
00567     *
00568     * @par Lua
00569     * pathOffsetCoordinate(ref_coord: number) -> number
00570     *
00571     * @par Lua
00572     * num = pathOffsetCoordinate(0)
00573     *
00574     * @par JSON-RPC
00575     * {"jsonrpc":"2.0","method":"rob1.MotionControl.pathOffsetCoordinate","params":[0],"id":1}
00576     *
00577     * @par JSON-RPC
00578     * {"id":1,"jsonrpc":"2.0","result":0}
00579     * \endchinese
00580     * \english
00581     * Set the reference coordinate system for offset.
00582     * Only valid when type=1 in pathOffsetSet.
00583     * @param ref_coord Reference coordinate system: 0-base coordinate, 1-TCP
00584     *
00585     * @throws arcs::common_interface::AuboException
00586     *
00587     * @par Python function prototype
00588     * pathOffsetCoordinate(self: pyaubo_sdk.MotionControl, arg0: int) -> float
00589     *
00590     * @par Lua function prototype
00591     * pathOffsetCoordinate(ref_coord: number) -> number
00592     *
00593     * @par Lua example
00594     * num = pathOffsetCoordinate(0)
00595     *
00596     * @par JSON-RPC request example
00597     * {"jsonrpc":"2.0","method":"rob1.MotionControl.pathOffsetCoordinate","params":[0],"id":1}
00598     *
00599     * @par JSON-RPC response example
00600     * {"id":1,"jsonrpc":"2.0","result":0}
00601     * \endenglish
00602     */
00603    int pathOffsetCoordinate(int ref_coord);
00604
```

```
00605     /**
00606      * @ingroup MotionControl
00607      * \chinese
00608      *
00609      *
00610      * @return    0;
00611      * AUBO_BUSY
00612      * AUBO_BAD_STATE
00613      * -AUBO_BAD_STATE
00614      *
00615      * @throws arcs::common_interface::AuboException
00616      *
00617      * @par Python
00618      * pathOffsetEnable(self: pyaubo_sdk.MotionControl) -> int
00619      *
00620      * @par Lua
00621      * pathOffsetEnable() -> number
00622      *
00623      * @par Lua
00624      * num = pathOffsetEnable()
00625      *
00626      * @par JSON-RPC
00627      * {"jsonrpc":"2.0","method":"rob1.MotionControl.pathOffsetEnable","params":[],"id":1}
00628      *
00629      * @par JSON-RPC
00630      * {"id":1,"jsonrpc":"2.0","result":0}
00631      * \endchinese
00632      * \english
00633      * Enable path offset
00634      *
00635      * @return Returns 0 on success; otherwise returns an error code:
00636      * AUBO_BUSY
00637      * AUBO_BAD_STATE
00638      * -AUBO_BAD_STATE
00639      *
00640      * @throws arcs::common_interface::AuboException
00641      *
00642      * @par Python function prototype
00643      * pathOffsetEnable(self: pyaubo_sdk.MotionControl) -> int
00644      *
00645      * @par Lua function prototype
00646      * pathOffsetEnable() -> number
00647      *
00648      * @par Lua example
00649      * num = pathOffsetEnable()
00650      *
00651      * @par JSON-RPC request example
00652      * {"jsonrpc":"2.0","method":"rob1.MotionControl.pathOffsetEnable","params":[],"id":1}
00653      *
00654      * @par JSON-RPC response example
00655      * {"id":1,"jsonrpc":"2.0","result":0}
00656      * \endenglish
00657      */
00658     int pathOffsetEnable();
00659
00660     /**
00661      * @ingroup MotionControl
00662      * \chinese
00663      *
00664      *
00665      * @param offset
00666      * @param type     0-    1-
00667      * @return    0;
00668      * AUBO_BAD_STATE
00669      * AUBO_BUSY
00670      * -AUBO_INVL_ARGUMENT
00671      * -AUBO_BAD_STATE
00672      *
00673      * @throws arcs::common_interface::AuboException
00674      *
00675      * @par Python
00676      * pathOffsetSet(self: pyaubo_sdk.MotionControl, arg0: List[float], arg1:
00677      * int) -> int
00678      *
00679      * @par Lua
00680      * pathOffsetSet(offset: table, type: number) -> nil
00681      *
00682      * @par Lua
00683      * pathOffsetSet({ 0, 0, 0.1, 0, 0, 0 }, 0)
00684      *
00685      * @par JSON-RPC
00686      * {"jsonrpc":"2.0","method":"rob1.MotionControl.pathOffsetSet","params":[[0,0,0.01,0,0,0],0],"id":1}
00687      *
00688      * @par JSON-RPC
00689      * {"id":1,"jsonrpc":"2.0","result":0}
00690      * \endchinese
00691      * \english
```

```
00692     * Set path offset
00693     *
00694     * @param offset Pose offset in each direction
00695     * @param type Motion type 0-position planning 1-velocity planning
00696     * @return Returns 0 on success; otherwise returns error code
00697     * AUBO_BAD_STATE
00698     * AUBO_BUSY
00699     * -AUBO_INVL_ARGUMENT
00700     * -AUBO_BAD_STATE
00701     *
00702     * @throws arcs::common_interface::AuboException
00703     *
00704     * @par Python function prototype
00705     * pathOffsetSet(self: pyaubo_sdk.MotionControl, arg0: List[float], arg1:
00706     * int) -> int
00707     *
00708     * @par Lua function prototype
00709     * pathOffsetSet(offset: table, type: number) -> nil
00710     *
00711     * @par Lua example
00712     * pathOffsetSet({ 0, 0, 0.1, 0, 0, 0 }, 0)
00713     *
00714     * @par JSON-RPC request example
00715     * {"jsonrpc":"2.0","method":"rob1.MotionControl.pathOffsetSet","params":[[0,0,0.01,0,0,0],0],"id":1}
00716     *
00717     * @par JSON-RPC response example
00718     * {"id":1,"jsonrpc":"2.0","result":0}
00719     * \endenglish
00720     */
00721    int pathOffsetSet(const std::vector<double> &offset, int type = 0);
00722
00723    /**
00724     * @ingroup MotionControl
00725     * \chinese
00726     *
00727     *
00728     * @return    0;
00729     * AUBO_BAD_STATE
00730     * AUBO_BUSY
00731     * -AUBO_BAD_STATE
00732     *
00733     * @throws arcs::common_interface::AuboException
00734     *
00735     * @par Python
00736     * pathOffsetDisable(self: pyaubo_sdk.MotionControl) -> int
00737     *
00738     * @par Lua
00739     * pathOffsetDisable() -> nil
00740     *
00741     * @par Lua
00742     * pathOffsetDisable()
00743     *
00744     * @par JSON-RPC
00745     * {"jsonrpc":"2.0","method":"rob1.MotionControl.pathOffsetDisable","params":[],"id":1}
00746     *
00747     * @par JSON-RPC
00748     * {"id":1,"jsonrpc":"2.0","result":0}
00749     * \endchinese
00750     * \english
00751     * Disable path offset
00752     *
00753     * @return Returns 0 on success; otherwise returns an error code:
00754     * AUBO_BAD_STATE
00755     * AUBO_BUSY
00756     * -AUBO_BAD_STATE
00757     *
00758     * @throws arcs::common_interface::AuboException
00759     *
00760     * @par Python function prototype
00761     * pathOffsetDisable(self: pyaubo_sdk.MotionControl) -> int
00762     *
00763     * @par Lua function prototype
00764     * pathOffsetDisable() -> nil
00765     *
00766     * @par Lua example
00767     * pathOffsetDisable()
00768     *
00769     * @par JSON-RPC request example
00770     * {"jsonrpc":"2.0","method":"rob1.MotionControl.pathOffsetDisable","params":[],"id":1}
00771     *
00772     * @par JSON-RPC response example
00773     * {"id":1,"jsonrpc":"2.0","result":0}
00774     * \endenglish
00775     */
00776    int pathOffsetDisable();
00777
00778    /**
```

```
00779     * @ingroup MotionControl
00780     * \chinese
00781     * @brief
00782     * @param min:
00783     * @param max:
00784     * @param strategy:
00785     *          0-
00786     *          1-
00787     *          2-
00788     * @return
00789     *
00790     * @throws arcs::common_interface::AuboException
00791     *
00792     * @par Python
00793     * pathOffsetSupv(self: pyaubo_sdk.MotionControl, arg0: List[float], arg1:
00794     * List[float], arg2: int) -> int
00795     *
00796     * @par Lua
00797     * pathOffsetSupv(min: table, max: table, strategy: number) -> number
00798     *
00799     * @par Lua
00800     * num = pathOffsetSupv({0,0,-0.2,0,0,0},{0,0,0.5,0,0,0},0)
00801     *
00802     * \endchinese
00803     * \english
00804     * @brief Monitor trajectory offset range
00805     * @param min: Maximum offset in the negative direction of the coordinate
00806     * axis
00807     * @param max: Maximum offset in the positive direction of the coordinate
00808     * axis
00809     * @param strategy: Monitoring strategy after reaching the maximum offset
00810     *          0 - Disable monitoring
00811     *          1 - Saturation limit, i.e., maintain maximum pose
00812     *          2 - Protective stop
00813     * @return
00814     *
00815     * @throws arcs::common_interface::AuboException
00816     *
00817     * @par Python function prototype
00818     * pathOffsetSupv(self: pyaubo_sdk.MotionControl, arg0: List[float], arg1:
00819     * List[float], arg2: int) -> int
00820     *
00821     * @par Lua function prototype
00822     * pathOffsetSupv(min: table, max: table, strategy: number) -> number
00823     *
00824     * @par Lua example
00825     * num = pathOffsetSupv({0,0,-0.2,0,0,0},{0,0,0.5,0,0,0},0)
00826     *
00827     * \endenglish
00828     */
00829    int pathOffsetSupv(const std::vector<double> &min,
00830                       const std::vector<double> &max, int strategy);
00831
00832    /**
00833     * @ingroup MotionControl
00834     * \chinese
00835     *
00836     *
00837     * @return    0;
00838     * AUBO_BAD_STATE
00839     * AUBO_BUSY
00840     * -AUBO_BAD_STATE
00841     *
00842     * @throws arcs::common_interface::AuboException
00843     *
00844     * @par Python
00845     * jointOffsetEnable(self: pyaubo_sdk.MotionControl) -> int
00846     *
00847     * @par Lua
00848     * jointOffsetEnable() -> nil
00849     *
00850     * @par Lua
00851     * jointOffsetEnable()
00852     *
00853     * @par JSON-RPC
00854     * {"jsonrpc":"2.0","method":"rob1.MotionControl.jointOffsetEnable","params":[],"id":1}
00855     *
00856     * @par JSON-RPC
00857     * {"id":1,"jsonrpc":"2.0","result":0}
00858     * \endchinese
00859     * \english
00860     * Enable joint offset
00861     *
00862     * @return Returns 0 on success; otherwise returns an error code:
00863     * AUBO_BAD_STATE
00864     * AUBO_BUSY
00865     * -AUBO_BAD_STATE
```

```
00866     *
00867     * @throws arcs::common_interface::AuboException
00868     *
00869     * @par Python function prototype
00870     * jointOffsetEnable(self: pyaubo_sdk.MotionControl) -> int
00871     *
00872     * @par Lua function prototype
00873     * jointOffsetEnable() -> nil
00874     *
00875     * @par Lua example
00876     * jointOffsetEnable()
00877     *
00878     * @par JSON-RPC request example
00879     * {”jsonrpc”:”2.0”,”method”:”rob1.MotionControl.jointOffsetEnable”,”params”:[],”id”:1}
00880     *
00881     * @par JSON-RPC response example
00882     * {”id”:1,”jsonrpc”:”2.0”,”result”:0}
00883     * \endenglish
00884     */
00885    int jointOffsetEnable();
00886
00887    /**
00888     * @ingroup MotionControl
00889     * \chinese
00890     *
00891     *
00892     * @param offset
00893     * @param type
00894     * @return    0;
00895     * AUBO_BAD_STATE
00896     * AUBO_BUSY
00897     * -AUBO_INVL_ARGUMENT
00898     * -AUBO_BAD_STATE
00899     *
00900     * @throws arcs::common_interface::AuboException
00901     *
00902     * @par Python
00903     * jointOffsetSet(self: pyaubo_sdk.MotionControl, arg0: List[float], arg1:
00904     * int) -> int
00905     *
00906     * @par Lua
00907     * jointOffsetSet(offset: table, type: number) -> nil
00908     *
00909     * @par Lua
00910     * jointOffsetSet({0.1,0,0,0,0,0},1)
00911     *
00912     * @par JSON-RPC
00913     * {”jsonrpc”:”2.0”,”method”:”rob1.MotionControl.jointOffsetSet”,”params”:[[0.1,0,0,0,0,0],1],”id”:1}
00914     *
00915     * @par JSON-RPC
00916     * {”id”:1,”jsonrpc”:”2.0”,”result”:0}
00917     * \endchinese
00918     * \english
00919     * Set joint offset
00920     *
00921     * @param offset Pose offset for each joint
00922     * @param type
00923     * @return Returns 0 on success; otherwise returns error code
00924     * AUBO_BAD_STATE
00925     * AUBO_BUSY
00926     * -AUBO_INVL_ARGUMENT
00927     * -AUBO_BAD_STATE
00928     *
00929     * @throws arcs::common_interface::AuboException
00930     *
00931     * @par Python function prototype
00932     * jointOffsetSet(self: pyaubo_sdk.MotionControl, arg0: List[float], arg1:
00933     * int) -> int
00934     *
00935     * @par Lua function prototype
00936     * jointOffsetSet(offset: table, type: number) -> nil
00937     *
00938     * @par Lua example
00939     * jointOffsetSet({0.1,0,0,0,0,0},1)
00940     *
00941     * @par JSON-RPC request example
00942     * {”jsonrpc”:”2.0”,”method”:”rob1.MotionControl.jointOffsetSet”,”params”:[[0.1,0,0,0,0,0],1],”id”:1}
00943     *
00944     * @par JSON-RPC response example
00945     * {”id”:1,”jsonrpc”:”2.0”,”result”:0}
00946     * \endenglish
00947     */
00948    int jointOffsetSet(const std::vector<double> &offset, int type = 1);
00949
00950    /**
00951     * @ingroup MotionControl
00952     * \chinese
```

```
00953     *
00954     *
00955     * @return    0;
00956     * AUBO_BAD_STATE
00957     * AUBO_BUSY
00958     * -AUBO_BAD_STATE
00959     *
00960     * @throws arcs::common_interface::AuboException
00961     *
00962     * @par Python
00963     * jointOffsetDisable(self: pyaubo_sdk.MotionControl) -> int
00964     *
00965     * @par Lua
00966     * jointOffsetDisable() -> nil
00967     *
00968     * @par Lua
00969     * jointOffsetDisable()
00970     *
00971     * @par JSON-RPC
00972     * {"jsonrpc":"2.0","method":"rob1.MotionControl.jointOffsetDisable","params":[],"id":1}
00973     *
00974     * @par JSON-RPC
00975     * {"id":1,"jsonrpc":"2.0","result":0}
00976     * \endchinese
00977     * \english
00978     * Disable joint offset
00979     *
00980     * @return Returns 0 on success; otherwise returns an error code:
00981     * AUBO_BAD_STATE
00982     * AUBO_BUSY
00983     * -AUBO_BAD_STATE
00984     *
00985     * @throws arcs::common_interface::AuboException
00986     *
00987     * @par Python function prototype
00988     * jointOffsetDisable(self: pyaubo_sdk.MotionControl) -> int
00989     *
00990     * @par Lua function prototype
00991     * jointOffsetDisable() -> nil
00992     *
00993     * @par Lua example
00994     * jointOffsetDisable()
00995     *
00996     * @par JSON-RPC request example
00997     * {"jsonrpc":"2.0","method":"rob1.MotionControl.jointOffsetDisable","params":[],"id":1}
00998     *
00999     * @par JSON-RPC response example
01000     * {"id":1,"jsonrpc":"2.0","result":0}
01001     * \endenglish
01002     */
01003    int jointOffsetDisable();
01004
01005    /**
01006     * @ingroup MotionControl
01007     * \chinese
01008     *       (INST)
01009     * moveJoint/moveLine/moveCircle       setPayload
01010     *
01011     *           _INST   ,    moveJoint
01012     *   _INST(MotionControl, 5, moveJoint, q, a, v, blend_radius, duration)
01013     *
01014     * @return
01015     *
01016     * @throws arcs::common_interface::AuboException
01017     *
01018     * @par Python
01019     * getQueueSize(self: pyaubo_sdk.MotionControl) -> int
01020     *
01021     * @par Lua
01022     * getQueueSize() -> number
01023     *
01024     * @par Lua
01025     * num = getQueueSize()
01026     *
01027     * @par JSON-RPC
01028     * {"jsonrpc":"2.0","method":"rob1.MotionControl.getQueueSize","params":[],"id":1}
01029     *
01030     * @par JSON-RPC
01031     * {"id":1,"jsonrpc":"2.0","result":0}
01032     *
01033     * \endchinese
01034     * \english
01035     * Get the number of enqueued instruction segments (INST), including motion
01036     * instructions such as moveJoint/moveLine/moveCircle and configuration
01037     * instructions such as setPayload.
01038     *
01039     * Instructions are generally indicated with _INST in macro definitions, for
```

```
01040     * example: _INST(MotionControl, 5, moveJoint, q, a, v, blend_radius,
01041     * duration)
01042     *
01043     * @return The number of enqueued instruction segments.
01044     *
01045     * @throws arcs::common_interface::AuboException
01046     *
01047     * @par Python function prototype
01048     * getQueueSize(self: pyaubo_sdk.MotionControl) -> int
01049     *
01050     * @par Lua function prototype
01051     * getQueueSize() -> number
01052     *
01053     * @par Lua example
01054     * num = getQueueSize()
01055     *
01056     * @par JSON-RPC request example
01057     * {"jsonrpc":"2.0","method":"rob1.MotionControl.getQueueSize","params":[],"id":1}
01058     *
01059     * @par JSON-RPC response example
01060     * {"id":1,"jsonrpc":"2.0","result":0}
01061     *
01062     * \endenglish
01063     */
01064    int getQueueSize();
01065
01066    /**
01067     * @ingroup MotionControl
01068     * \chinese
01069     *
01070     *
01071     * @return
01072     *
01073     * @throws arcs::common_interface::AuboException
01074     *
01075     * @par Python
01076     * getTrajectoryQueueSize(self: pyaubo_sdk.MotionControl) -> int
01077     *
01078     * @par Lua
01079     * getTrajectoryQueueSize() -> number
01080     *
01081     * @par Lua
01082     * num = getTrajectoryQueueSize()
01083     *
01084     * @par JSON-RPC
01085     * {"jsonrpc":"2.0","method":"rob1.MotionControl.getTrajectoryQueueSize","params":[],"id":1}
01086     *
01087     * @par JSON-RPC
01088     * {"id":1,"jsonrpc":"2.0","result":0}
01089     * \endchinese
01090     * \english
01091     * Get the number of enqueued motion planning interpolation points
01092     *
01093     * @return The number of enqueued motion planning interpolation points
01094     *
01095     * @throws arcs::common_interface::AuboException
01096     *
01097     * @par Python function prototype
01098     * getTrajectoryQueueSize(self: pyaubo_sdk.MotionControl) -> int
01099     *
01100     * @par Lua function prototype
01101     * getTrajectoryQueueSize() -> number
01102     *
01103     * @par Lua example
01104     * num = getTrajectoryQueueSize()
01105     *
01106     * @par JSON-RPC request example
01107     * {"jsonrpc":"2.0","method":"rob1.MotionControl.getTrajectoryQueueSize","params":[],"id":1}
01108     *
01109     * @par JSON-RPC response example
01110     * {"id":1,"jsonrpc":"2.0","result":0}
01111     * \endenglish
01112     */
01113    int getTrajectoryQueueSize();
01114
01115    /**
01116     * @ingroup MotionControl
01117     * \chinese
01118     *           ID
01119     *
01120     * @return          ID
01121     * @retval -1       \n
01122     *  movePathBuffer   buffer    (moveJoint moveLine )
01123     *
01124     * @throws arcs::common_interface::AuboException
01125     *
01126     * @par Python
```

```
01127     * getExecId(self: pyaubo_sdk.MotionControl) -> int
01128     *
01129     * @par Lua
01130     * getExecId() -> number
01131     *
01132     * @par Lua
01133     * num = getExecId()
01134     *
01135     * @par JSON-RPC
01136     * {"jsonrpc":"2.0","method":"rob1.MotionControl.getExecId","params":[],"id":1}
01137     *
01138     * @par JSON-RPC
01139     * {"id":1,"jsonrpc":"2.0","result":-1}
01140     *
01141     * \endchinese
01142     * \english
01143     * Get the ID of the currently interpolating motion instruction segment.
01144     *
01145     * @return The ID of the currently interpolating motion instruction segment.
01146     * @retval -1 Indicates the trajectory queue is empty. \n
01147     * Both the buffer in movePathBuffer motion and the queue in the planner
01148     * (such as moveJoint and moveLine) belong to the trajectory queue.
01149     *
01150     * @throws arcs::common_interface::AuboException
01151     *
01152     * @par Python function prototype
01153     * getExecId(self: pyaubo_sdk.MotionControl) -> int
01154     *
01155     * @par Lua function prototype
01156     * getExecId() -> number
01157     *
01158     * @par Lua example
01159     * num = getExecId()
01160     *
01161     * @par JSON-RPC request example
01162     * {"jsonrpc":"2.0","method":"rob1.MotionControl.getExecId","params":[],"id":1}
01163     *
01164     * @par JSON-RPC response example
01165     * {"id":1,"jsonrpc":"2.0","result":-1}
01166     *
01167     * \endenglish
01168     */
01169    int getExecId();
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     * getDuration(self: pyaubo_sdk.MotionControl, arg0: int) -> float
01183     *
01184     * @par Lua
01185     * getDuration(id: number) -> number
01186     *
01187     * @par Lua
01188     * num = getDuration(16)
01189     *
01190     * @par JSON-RPC
01191     * {"jsonrpc":"2.0","method":"rob1.MotionControl.getDuration","params":[16],"id":1}
01192     *
01193     * @par JSON-RPC
01194     * {"id":1,"jsonrpc":"2.0","result":0.0}
01195     *
01196     * \endchinese
01197     * \english
01198     * Get the expected execution duration of the motion segment with the
01199     * specified ID.
01200     *
01201     * @param id Motion segment ID
01202     * @return Returns the expected execution duration
01203     *
01204     * @throws arcs::common_interface::AuboException
01205     *
01206     * @par Python function prototype
01207     * getDuration(self: pyaubo_sdk.MotionControl, arg0: int) -> float
01208     *
01209     * @par Lua function prototype
01210     * getDuration(id: number) -> number
01211     *
01212     * @par Lua example
01213     * num = getDuration(16)
```

```
01214     *
01215     * @par JSON-RPC request example
01216     * {"jsonrpc":"2.0","method":"rob1.MotionControl.getDuration","params":[16],"id":1}
01217     *
01218     * @par JSON-RPC response example
01219     * {"id":1,"jsonrpc":"2.0","result":0.0}
01220     *
01221     * \endenglish
01222     */
01223    double getDuration(int id);
01224
01225    /**
01226     * @ingroup MotionControl
01227     * \chinese
01228     *    ID
01229     *
01230     * @param id     ID
01231     * @return
01232     *
01233     * @throws arcs::common_interface::AuboException
01234     *
01235     * @par Python
01236     * getMotionLeftTime(self: pyaubo_sdk.MotionControl, arg0: int) -> float
01237     *
01238     * @par Lua
01239     * getMotionLeftTime(id: number) -> number
01240     *
01241     * @par Lua
01242     * num = getMotionLeftTime(16)
01243     *
01244     * @par JSON-RPC
01245     * {"jsonrpc":"2.0","method":"rob1.MotionControl.getMotionLeftTime","params":[16],"id":1}
01246     *
01247     * @par JSON-RPC
01248     * {"id":1,"jsonrpc":"2.0","result":0.0}
01249     * \endchinese
01250     * \english
01251     * Get the remaining execution time of the motion segment with the specified
01252     * ID.
01253     *
01254     * @param id Motion segment ID
01255     * @return Returns the remaining execution time
01256     *
01257     * @throws arcs::common_interface::AuboException
01258     *
01259     * @par Python function prototype
01260     * getMotionLeftTime(self: pyaubo_sdk.MotionControl, arg0: int) -> float
01261     *
01262     * @par Lua function prototype
01263     * getMotionLeftTime(id: number) -> number
01264     *
01265     * @par Lua example
01266     * num = getMotionLeftTime(16)
01267     *
01268     * @par JSON-RPC request example
01269     * {"jsonrpc":"2.0","method":"rob1.MotionControl.getMotionLeftTime","params":[16],"id":1}
01270     *
01271     * @par JSON-RPC response example
01272     * {"id":1,"jsonrpc":"2.0","result":0.0}
01273     * \endenglish
01274     */
01275    double getMotionLeftTime(int id);
01276
01277    /**
01278     * @ingroup MotionControl
01279     * \chinese
01280     * StopMove
01281     * StartMove
01282     *
01283     *
01284     *
01285     * @param quick true:               quick
01286     *
01287     *
01288     * @return    0;
01289     * AUBO_BAD_STATE
01290     * AUBO_BUSY
01291     * -AUBO_BAD_STATE
01292     *
01293     * @throws arcs::common_interface::AuboException
01294     *
01295     * @par Lua
01296     * StopMove(quick: bool,all_tasks: bool) -> number
01297     *
01298     * @par Lua
01299     * num = StopMove(true,true)
01300     *
```

```
01301     * \endchinese
01302     * \english
01303     * StopMove is used to stop robot and external axes movements and any
01304     * belonging process temporarily. If the instruction StartMove is given then
01305     * the movement and process resumes.
01306     *
01307     * This instruction can, for example, be used in a trap routine to stop the
01308     * robot temporarily when an interrupt occurs.
01309     *
01310     * @param quick true: Stops the robot on the path as fast as possible.
01311     * Without the optional parameter \Quick, the robot stops on the path, but
01312     * the braking distance is longer (same as for normal Program Stop).
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     * StopMove(quick: bool,all_tasks: bool) -> number
01323     *
01324     * @par Lua example
01325     * num = StopMove(true,true)
01326     *
01327     * \endenglish
01328     */
01329    int stopMove(bool quick, bool all_tasks);
01330
01331    /**
01332     * @ingroup MotionControl
01333     * \chinese
01334     * StartMove
01335     * •    StopMove
01336     * •    StorePath ... RestoPath
01337     * •        ERR_PATH_STOP        ERROR
01338     *
01339     *
01340     * @return    0;
01341     * AUBO_BAD_STATE
01342     * AUBO_BUSY
01343     * -AUBO_BAD_STATE
01344     *
01345     * @throws arcs::common_interface::AuboException
01346     *
01347     * @par Lua
01348     * startMove() -> number
01349     *
01350     * @par Lua
01351     * num = startMove()
01352     *
01353     * @par JSON-RPC
01354     * {"jsonrpc":"2.0","method":"rob1.MotionControl.startMove","params":[],"id":1}
01355     *
01356     * @par JSON-RPC
01357     * {"id":1,"jsonrpc":"2.0","result":0}
01358     * \endchinese
01359     * \english
01360     * StartMove is used to resume robot, external axes movement and belonging
01361     * process after the movement has been stopped
01362     *
01363     * • by the instruction StopMove.
01364     * • after execution of StorePath ... RestoPath sequence.
01365     * • after asynchronously raised movements errors, such as ERR_PATH_STOP or
01366     * specific process error after handling in the ERROR handler.
01367     *
01368     * @return Returns 0 on success; otherwise returns an error code:
01369     * AUBO_BAD_STATE
01370     * AUBO_BUSY
01371     * -AUBO_BAD_STATE
01372     *
01373     * @throws arcs::common_interface::AuboException
01374     *
01375     * @par Lua function prototype
01376     * startMove() -> number
01377     *
01378     * @par Lua example
01379     * num = startMove()
01380     *
01381     * @par JSON-RPC request example
01382     * {"jsonrpc":"2.0","method":"rob1.MotionControl.startMove","params":[],"id":1}
01383     *
01384     * @par JSON-RPC response example
01385     * {"id":1,"jsonrpc":"2.0","result":0}
01386     * \endenglish
01387     */
```

```
01388     int startMove();
01389
01390     /**
01391      * @ingroup MotionControl
01392      * \chinese
01393      * storePath
01394      *
01395      * @param keep_sync
01396      * @return    0;
01397      * AUBO_BAD_STATE
01398      * AUBO_BUSY
01399      * -AUBO_BAD_STATE
01400      *
01401      * @throws arcs::common_interface::AuboException
01402      *
01403      * @par Lua
01404      * storePath(keep_sync: bool) -> number
01405      *
01406      * @par Lua
01407      * num = storePath()
01408      *
01409      * \endchinese
01410      * \english
01411      * storePath
01412      *
01413      * @param keep_sync
01414      * @return Returns 0 on success; otherwise returns an error code:
01415      * AUBO_BAD_STATE
01416      * AUBO_BUSY
01417      * -AUBO_BAD_STATE
01418      *
01419      * @throws arcs::common_interface::AuboException
01420      *
01421      * @par Lua function prototype
01422      * storePath(keep_sync: bool) -> number
01423      *
01424      * @par Lua example
01425      * num = storePath()
01426      *
01427      * \endenglish
01428      */
01429     int storePath(bool keep_sync);
01430
01431     /**
01432      * @ingroup MotionControl
01433      * \chinese
01434      * ClearPath (   )              StorePath
01435      *
01436      *
01437      * @return    0;
01438      * AUBO_BAD_STATE
01439      * AUBO_BUSY
01440      * -AUBO_BAD_STATE
01441      *
01442      * @throws arcs::common_interface::AuboException
01443      *
01444      * @par Lua
01445      * clearPath() -> number
01446      *
01447      * @par Lua
01448      * num = clearPath()
01449      *
01450      * @par JSON-RPC
01451      * {"jsonrpc":"2.0","method":"rob1.MotionControl.clearPath","params":[],"id":1}
01452      *
01453      * @par JSON-RPC
01454      * {"id":1,"jsonrpc":"2.0","result":0}
01455      *
01456      * \endchinese
01457      * \english
01458      * ClearPath clears the whole motion path on the current motion path level
01459      * (base level or StorePath level).
01460      *
01461      * @return Returns 0 on success; otherwise returns an error code:
01462      * AUBO_BAD_STATE
01463      * AUBO_BUSY
01464      * -AUBO_BAD_STATE
01465      *
01466      * @throws arcs::common_interface::AuboException
01467      *
01468      * @par Lua function prototype
01469      * clearPath() -> number
01470      *
01471      * @par Lua example
01472      * num = clearPath()
01473      *
01474      * @par JSON-RPC request example
```

```
01475     * {"jsonrpc":"2.0","method":"rob1.MotionControl.clearPath","params":[],"id":1}
01476     *
01477     * @par JSON-RPC response example
01478     * {"id":1,"jsonrpc":"2.0","result":0}
01479     *
01480     * \endenglish
01481     */
01482    int clearPath();
01483
01484    /**
01485     * @ingroup MotionControl
01486     * \chinese
01487     * restoPath
01488     *
01489     * @return    0;
01490     * AUBO_BAD_STATE
01491     * AUBO_BUSY
01492     * -AUBO_BAD_STATE
01493     *
01494     * @throws arcs::common_interface::AuboException
01495     *
01496     * @par Lua
01497     * restoPath() -> number
01498     *
01499     * @par Lua
01500     * num = restoPath()
01501     *
01502     * @par JSON-RPC
01503     * {"jsonrpc":"2.0","method":"rob1.MotionControl.restoPath","params":[],"id":1}
01504     *
01505     * @par JSON-RPC
01506     * {"id":1,"jsonrpc":"2.0","result":0}
01507     * \endchinese
01508     * \english
01509     * restoPath
01510     *
01511     * @return Returns 0 on success; otherwise returns an error code:
01512     * AUBO_BAD_STATE
01513     * AUBO_BUSY
01514     * -AUBO_BAD_STATE
01515     *
01516     * @throws arcs::common_interface::AuboException
01517     *
01518     * @par Lua function prototype
01519     * restoPath() -> number
01520     *
01521     * @par Lua example
01522     * num restoPath()
01523     *
01524     * @par JSON-RPC request example
01525     * {"jsonrpc":"2.0","method":"rob1.MotionControl.restoPath","params":[],"id":1}
01526     *
01527     * @par JSON-RPC response example
01528     * {"id":1,"jsonrpc":"2.0","result":0}
01529     * \endenglish
01530     */
01531    int restoPath();
01532
01533    /**
01534     * @ingroup MotionControl
01535     * \chinese
01536     *
01537     *
01538     * @return
01539     *
01540     * @throws arcs::common_interface::AuboException
01541     *
01542     * @par Python
01543     * getProgress(self: pyaubo_sdk.MotionControl) -> float
01544     *
01545     * @par Lua
01546     * getProgress() -> number
01547     *
01548     * @par Lua
01549     * num = getProgress()
01550     *
01551     * @par JSON-RPC
01552     * {"jsonrpc":"2.0","method":"rob1.MotionControl.getProgress","params":[],"id":1}
01553     *
01554     * @par JSON-RPC
01555     * {"id":1,"jsonrpc":"2.0","result":0.0}
01556     *
01557     * \endchinese
01558     * \english
01559     * Get the execution progress of the current motion instruction segment.
01560     *
01561     * @return Returns the execution progress.
```

```
01562     *
01563     * @throws arcs::common_interface::AuboException
01564     *
01565     * @par Python function prototype
01566     * getProgress(self: pyaubo_sdk.MotionControl) -> float
01567     *
01568     * @par Lua function prototype
01569     * getProgress() -> number
01570     *
01571     * @par Lua example
01572     * num = getProgress()
01573     *
01574     * @par JSON-RPC request example
01575     * {"jsonrpc":"2.0","method":"rob1.MotionControl.getProgress","params":[],"id":1}
01576     *
01577     * @par JSON-RPC response example
01578     * {"id":1,"jsonrpc":"2.0","result":0.0}
01579     *
01580     * \endenglish
01581     */
01582    double getProgress();
01583
01584    /**
01585     * @ingroup MotionControl
01586     * \chinese
01587     *
01588     *
01589     * @note
01590     *
01591     * @param module_name
01592     * @param mounting_pose
01593     *                TCP
01594     * @return    0;
01595     * AUBO_BAD_STATE
01596     * AUBO_BUSY
01597     * -AUBO_INVL_ARGUMENT
01598     * -AUBO_BAD_STATE
01599     *
01600     * @throws arcs::common_interface::AuboException
01601     *
01602     * @par Python
01603     * setWorkObjectHold(self: pyaubo_sdk.MotionControl, arg0: str, arg1:
01604     * List[float]) -> int
01605     *
01606     * @par Lua
01607     * setWorkObjectHold(module_name: string, mounting_pose: table) -> nil
01608     *
01609     * @par Lua
01610     * setWorkObjectHold("object",{0.2,0.1,-0.4,3.14,0,-1.57})
01611     *
01612     * \endchinese
01613     * \english
01614     * Specify the name and mounting position when the workpiece is installed on
01615     * the end of another robot or external axis.
01616     *
01617     * @note Not implemented yet
01618     *
01619     * @param module_name Name of the control module
01620     * @param mounting_pose Relative position of the grasp,
01621     * if it is a robot, it is relative to the robot's end center point (not the
01622     * TCP point)
01623     * @return Returns 0 on success; otherwise returns an error code:
01624     * AUBO_BAD_STATE
01625     * AUBO_BUSY
01626     * -AUBO_INVL_ARGUMENT
01627     * -AUBO_BAD_STATE
01628     *
01629     * @throws arcs::common_interface::AuboException
01630     *
01631     * @par Python function prototype
01632     * setWorkObjectHold(self: pyaubo_sdk.MotionControl, arg0: str, arg1:
01633     * List[float]) -> int
01634     *
01635     * @par Lua function prototype
01636     * setWorkObjectHold(module_name: string, mounting_pose: table) -> nil
01637     *
01638     * @par Lua example
01639     * setWorkObjectHold("object",{0.2,0.1,-0.4,3.14,0,-1.57})
01640     *
01641     * \endenglish
01642     */
01643    int setWorkObjectHold(const std::string &module_name,
01644                          const std::vector<double> &mounting_pose);
01645
01646    /**
01647     * @ingroup MotionControl
01648     * \chinese
```

```
01649     *
01650     *
01651     * @note
01652     *
01653     * @return
01654     *
01655     * @throws arcs::common_interface::AuboException
01656     *
01657     * @par Python
01658     * getWorkObjectHold(self: pyaubo_sdk.MotionControl) -> Tuple[str,
01659     * List[float]]
01660     *
01661     * @par Lua
01662     * getWorkObjectHold() -> table
01663     *
01664     * @par Lua
01665     * Object_table = getWorkObjectHold()
01666     *
01667     * @par JSON-RPC
01668     * {"jsonrpc":"2.0","method":"rob1.MotionControl.getWorkObjectHold","params":[],"id":1}
01669     *
01670     * @par JSON-RPC
01671     * {"id":1,"jsonrpc":"2.0","result":["",[]]}
01672     * \endchinese
01673     * \english
01674     * getWorkObjectHold
01675     *
01676     * @note Not implemented yet
01677     *
01678     * @return Returns a tuple containing the control module name and mounting
01679     * pose
01680     *
01681     * @throws arcs::common_interface::AuboException
01682     *
01683     * @par Python function prototype
01684     * getWorkObjectHold(self: pyaubo_sdk.MotionControl) -> Tuple[str,
01685     * List[float]]
01686     *
01687     * @par Lua function prototype
01688     * getWorkObjectHold() -> table
01689     *
01690     * @par Lua example
01691     * Object_table = getWorkObjectHold()
01692     *
01693     * @par JSON-RPC request example
01694     * {"jsonrpc":"2.0","method":"rob1.MotionControl.getWorkObjectHold","params":[],"id":1}
01695     *
01696     * @par JSON-RPC response example
01697     * {"id":1,"jsonrpc":"2.0","result":["",[]]}
01698     * \endenglish
01699     */
01700    std::tuple<std::string, std::vector<double» getWorkObjectHold();
01701
01702    /**
01703     * @ingroup MotionControl
01704     * \chinese
01705     * @note
01706     *
01707     *              (   resumeMoveJoint resumeMoveLine      ,    )
01708     *
01709     * @return
01710     *
01711     * @throws arcs::common_interface::AuboException
01712     *
01713     * @par Python
01714     * getPauseJointPositions(self: pyaubo_sdk.MotionControl) -> List[float]
01715     *
01716     * @par Lua
01717     * getPauseJointPositions() -> table
01718     *
01719     * @par Lua
01720     * JointPositions = getPauseJointPositions()
01721     *
01722     * @par JSON-RPC
01723     * {"jsonrpc":"2.0","method":"rob1.MotionControl.getPauseJointPositions","params":[],"id":1}
01724     *
01725     * @par JSON-RPC
01726     * {"id":1,"jsonrpc":"2.0","result":[8.2321e-13,-0.200999,1.33999,0.334999,1.206,-6.39383e-12]}
01727     * \endchinese
01728     * \english
01729     * @note Get the joint positions at the pause point.
01730     *
01731     * Commonly used during process recovery after a collision occurs (first
01732     * move to the pause position using resumeMoveJoint or resumeMoveLine, then
01733     * resume the process).
01734     *
01735     * @return Pause joint positions
```

```
01736     *
01737     * @throws arcs::common_interface::AuboException
01738     *
01739     * @par Python function prototype
01740     * getPauseJointPositions(self: pyaubo_sdk.MotionControl) -> List[float]
01741     *
01742     * @par Lua function prototype
01743     * getPauseJointPositions() -> table
01744     *
01745     * @par Lua example
01746     * JointPositions = getPauseJointPositions()
01747     *
01748     * @par JSON-RPC request example
01749     * {"jsonrpc":"2.0","method":"rob1.MotionControl.getPauseJointPositions","params":[],"id":1}
01750     *
01751     * @par JSON-RPC response example
01752     * {"id":1,"jsonrpc":"2.0","result":[8.2321e-13,-0.200999,1.33999,0.334999,1.206,-6.39383e-12]}
01753     * \endenglish
01754     */
01755    std::vector<double> getPauseJointPositions();
01756
01757    /**
01758     * @ingroup MotionControl
01759     * \chinese
01760     *
01761     *
01762     * @param q
01763     * @param move_type 0:    1
01764     * @param blend_radius
01765     * @param qdmax       (6    )
01766     * @param qddmax       (6    )
01767     * @param vmax       ,  (2    )
01768     * @param amax          (2    )
01769     * @return    0;
01770     * AUBO_BAD_STATE
01771     * AUBO_BUSY
01772     * -AUBO_BAD_STATE
01773     *
01774     * @throws arcs::common_interface::AuboException
01775     *
01776     * @par Python
01777     * setResumeStartPoint(self: pyaubo_sdk.MotionControl,
01778     * arg0 List[float],arg1: int  arg2: float arg3: List[float], arg4:
01779     * List[float],arg5:float,arg5:float) -> int
01780     *
01781     * @par Lua
01782     * setResumeStartPoint(q : table, move_type: number,blend_radius:
01783     * number, qdmax: table, qddmax:
01784     * table,vmax:number,amax:number) -> nilr
01785     *
01786     * @par Lua
01787     * 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})
01788     *
01789     * @par JSON-RPC
01790     * {"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}
01791     *
01792     * @par JSON-RPC
01793     * {"id":1,"jsonrpc":"2.0","result":0}
01794     * \endchinese
01795     * \english
01796     * Set resume motion parameters
01797     *
01798     * @param q Resume start position
01799     * @param move_type 0: Joint space, 1: Cartesian space
01800     * @param blend_radius Blend radius
01801     * @param qdmax Maximum joint speed (6 elements)
01802     * @param qddmax Maximum joint acceleration (6 elements)
01803     * @param vmax Maximum linear and angular speed for linear motion (2
01804     * elements)
01805     * @param amax Maximum linear and angular acceleration for linear motion (2
01806     * elements)
01807     * @return Returns 0 on success; otherwise returns error code
01808     * AUBO_BAD_STATE
01809     * AUBO_BUSY
01810     * -AUBO_BAD_STATE
01811     *
01812     * @throws arcs::common_interface::AuboException
01813     *
01814     * @par Python function prototype
01815     * setResumeStartPoint(self: pyaubo_sdk.MotionControl,
01816     * arg0: List[float], arg1: int, arg2: float, arg3: List[float], arg4:
01817     * List[float], arg5: float, arg6: float) -> int
01818     *
01819     * @par Lua function prototype
01820     * setResumeStartPoint(q: table, move_type: number, blend_radius:
01821     * number, qdmax: table, qddmax:
```

```
01822     * table, vmax: number, amax: number) -> nil
01823     *
01824     * @par Lua example
01825     * 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})
01826     *
01827     * @par JSON-RPC request example
01828     * {"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}
01829     *
01830     * @par JSON-RPC response example
01831     * {"id":1,"jsonrpc":"2.0","result":0}
01832     * \endenglish
01833     */
01834    int setResumeStartPoint(const std::vector<double> &q, int move_type,
01835                            double blend_radius,
01836                            const std::vector<double> &qdmax,
01837                            const std::vector<double> &qddmax,
01838                            const std::vector<double> &vmax,
01839                            const std::vector<double> &amax);
01840    /**
01841     * @ingroup MotionControl
01842     * \chinese
01843     *
01844     *
01845     * @return 0:         1:
01846     *
01847     * @throws arcs::common_interface::AuboException
01848     *
01849     * @par Python
01850     * getResumeMode(self: pyaubo_sdk.MotionControl) -> int
01851     *
01852     * @par Lua
01853     * getResumeMode() -> int
01854     *
01855     * @par Lua
01856     * num = getResumeMode()
01857     *
01858     * @par JSON-RPC
01859     * {"jsonrpc":"2.0","method":"rob1.MotionControl.getResumeMode","params":[],"id":1}
01860     *
01861     * @par JSON-RPC
01862     * {"id":1,"jsonrpc":"2.0","result":false}
01863     *
01864     * \endchinese
01865     * \english
01866     * Get resume motion mode
01867     *
01868     * @return 0: Resume start point is the pause point; 1: Resume start point
01869     * is the specified point
01870     *
01871     * @throws arcs::common_interface::AuboException
01872     *
01873     * @par Python function prototype
01874     * getResumeMode(self: pyaubo_sdk.MotionControl) -> int
01875     *
01876     * @par Lua function prototype
01877     * getResumeMode() -> int
01878     *
01879     * @par Lua example
01880     * num = getResumeMode()
01881     *
01882     * @par JSON-RPC request example
01883     * {"jsonrpc":"2.0","method":"rob1.MotionControl.getResumeMode","params":[],"id":1}
01884     *
01885     * @par JSON-RPC response example
01886     * {"id":1,"jsonrpc":"2.0","result":false}
01887     *
01888     * \endenglish
01889     */
01890    int getResumeMode();
01891
01892    /**
01893     * @ingroup MotionControl
01894     * \chinese
01895     *
01896     *    setServoModeSelect
01897     *
01898     * @param enable
01899     * @return    0;
01900     * AUBO_BAD_STATE
01901     * AUBO_BUSY
01902     * -AUBO_BAD_STATE
01903     *
01904     * @throws arcs::common_interface::AuboException
01905     *
01906     * @par Python
01907     * setServoMode(self: pyaubo_sdk.MotionControl, arg0: bool) -> int
```

```
01908     *
01909     * @par Lua
01910     * setServoMode(enable: boolean) -> nil
01911     *
01912     * @par Lua
01913     * setServoMode(true)
01914     *
01915     * @par JSON-RPC
01916     * {"jsonrpc":"2.0","method":"rob1.MotionControl.setServoMode","params":[true],"id":1}
01917     *
01918     * @par JSON-RPC
01919     * {"id":1,"jsonrpc":"2.0","result":0}
01920     * \endchinese
01921     * \english
01922     * Set servo mode
01923     * Use setServoModeSelect instead
01924     *
01925     * @param enable Whether to enable
01926     * @return Returns 0 on success; otherwise returns an error code:
01927     * AUBO_BAD_STATE
01928     * AUBO_BUSY
01929     * -AUBO_BAD_STATE
01930     *
01931     * @throws arcs::common_interface::AuboException
01932     *
01933     * @par Python function prototype
01934     * setServoMode(self: pyaubo_sdk.MotionControl, arg0: bool) -> int
01935     *
01936     * @par Lua function prototype
01937     * setServoMode(enable: boolean) -> nil
01938     *
01939     * @par Lua example
01940     * setServoMode(true)
01941     *
01942     * @par JSON-RPC request example
01943     * {"jsonrpc":"2.0","method":"rob1.MotionControl.setServoMode","params":[true],"id":1}
01944     *
01945     * @par JSON-RPC response example
01946     * {"id":1,"jsonrpc":"2.0","result":0}
01947     * \endenglish
01948     */
01949    ARCS_DEPRECATED int setServoMode(bool enable);
01950
01951    /**
01952     * @ingroup MotionControl
01953     * \chinese
01954     *
01955     *   getServoModeSelect
01956     *
01957     * @return     true     false
01958     *
01959     * @throws arcs::common_interface::AuboException
01960     *
01961     * @par Python
01962     * isServoModeEnabled(self: pyaubo_sdk.MotionControl) -> bool
01963     *
01964     * @par Lua
01965     * isServoModeEnabled() -> boolean
01966     *
01967     * @par Lua
01968     * Servo_status = isServoModeEnabled()
01969     *
01970     * @par JSON-RPC
01971     * {"jsonrpc":"2.0","method":"rob1.MotionControl.isServoModeEnabled","params":[],"id":1}
01972     *
01973     * @par JSON-RPC
01974     * {"id":1,"jsonrpc":"2.0","result":false}
01975     *
01976     * \endchinese
01977     * \english
01978     * Determine whether the servo mode is enabled.
01979     * Use getServoModeSelect instead.
01980     *
01981     * @return Returns true if enabled, otherwise returns false.
01982     *
01983     * @throws arcs::common_interface::AuboException
01984     *
01985     * @par Python function prototype
01986     * isServoModeEnabled(self: pyaubo_sdk.MotionControl) -> bool
01987     *
01988     * @par Lua function prototype
01989     * isServoModeEnabled() -> boolean
01990     *
01991     * @par Lua example
01992     * Servo_status = isServoModeEnabled()
01993     *
01994     * @par JSON-RPC request example
```

```
01995     * {"jsonrpc":"2.0","method":"rob1.MotionControl.isServoModeEnabled","params":[],"id":1}
01996     *
01997     * @par JSON-RPC response example
01998     * {"id":1,"jsonrpc":"2.0","result":false}
01999     *
02000     * \endenglish
02001     */
02002    ARCS_DEPRECATED bool isServoModeEnabled();
02003
02004    /**
02005     * @ingroup MotionControl
02006     * \chinese
02007     *
02008     *
02009     * @param mode
02010     * 0-
02011     * 1-(  )
02012     * 2-   (  )
02013     * 3-   ( )
02014     * 4-1ms   ( )
02015     * 5-
02016     * 6-(  )    ,
02017     * 7-
02018     *    1            (                )
02019     *    5
02020     * @return
02021     *
02022     * @par Lua
02023     * setServoModeSelect(0) -> nil
02024     *
02025     * @par Lua
02026     * setServoModeSelect(0)
02027     *
02028     * @par JSON-RPC
02029     * {"jsonrpc":"2.0","method":"rob1.MotionControl.setServoModeSelect","params":[0],"id":1}
02030     *
02031     * @par JSON-RPC
02032     * {"id":1,"jsonrpc":"2.0","result":0}
02033     *
02034     * \endchinese
02035     * \english
02036     * Set servo motion mode
02037     *
02038     * @param mode
02039     * 0 - Exit servo mode
02040     * 1 - Planning servo mode
02041     * 2 - Transparent mode (direct send)
02042     * 3 - Transparent mode (buffered)
02043     * @return
02044     *
02045     * @par Lua function prototype
02046     * setServoModeSelect(0) -> nil
02047     *
02048     * @par Lua example
02049     * setServoModeSelect(0)
02050     *
02051     * @par JSON-RPC request example
02052     * {"jsonrpc":"2.0","method":"rob1.MotionControl.setServoModeSelect","params":[0],"id":1}
02053     *
02054     * @par JSON-RPC response example
02055     * {"id":1,"jsonrpc":"2.0","result":0}
02056     *
02057     * \endenglish
02058     */
02059    int setServoModeSelect(int mode);
02060
02061    /**
02062     * @ingroup MotionControl
02063     * \chinese
02064     *
02065     *
02066     * @return
02067     *
02068     * @par Lua
02069     * getServoModeSelect() -> number
02070     *
02071     * @par Lua
02072     * num = getServoModeSelect()
02073     *
02074     * @par JSON-RPC
02075     * {"jsonrpc":"2.0","method":"rob1.MotionControl.getServoModeSelect","params":[],"id":1}
02076     *
02077     * @par JSON-RPC
02078     * {"id":1,"jsonrpc":"2.0","result":0}
02079     *
02080     * \endchinese
02081     * \english
```

```
02082     * Get the servo motion mode
02083     *
02084     * @return
02085     *
02086     * @par Lua function prototype
02087     * getServoModeSelect() -> number
02088     *
02089     * @par Lua example
02090     * num = getServoModeSelect()
02091     *
02092     * @par JSON-RPC request example
02093     * {"jsonrpc":"2.0","method":"rob1.MotionControl.getServoModeSelect","params":[],"id":1}
02094     *
02095     * @par JSON-RPC response example
02096     * {"id":1,"jsonrpc":"2.0","result":0}
02097     *
02098     * \endenglish
02099     */
02100    int getServoModeSelect();
02101
02102    /**
02103     * @ingroup MotionControl
02104     * \chinese
02105     *
02106     *
02107     *       q   t;
02108     * @param q    ,    rad,
02109     * @param a    ,    rad/s^2,
02110     * @param v      rad/s,
02111     * @param t       s \n
02112     * t    ,      ,     ;
02113     *         servoJoint
02114     * @param lookahead_time       s \n
02115     *                    [0.03, 0.2],
02116     *   lookahead_time               ,
02117     *
02118     * @param gain
02119     *         [100, 200],
02120     *           ,
02121     *
02122     *
02123     * @retval 0
02124     * @retval AUBO_BAD_STATE(1)
02125     * Normal ReducedMode Recovery
02126     * @retval AUBO_QUEUE_FULL(2)
02127     * @retval AUBO_BUSY(3)
02128     * @retval -AUBO_BAD_STATE(-1)
02129     *                  task_id
02130     *            Running
02131     * @retval -AUBO_TIMEOUT(-4)
02132     * @retval -AUBO_INVL_ARGUMENT(-5)                (±15°)
02133     * @retval -AUBO_REQUEST_IGNORE(-13)      servo
02134     * @retval -AUBO_POS_BOUND_ERR(-18)
02135     * @retval -AUBO_TRAJ_SELF_COLLISION(-22)
02136     * @retval -AUBO_ALGORITHM_PLAN_FAILED(-30)
02137     *
02138     * @throws arcs::common_interface::AuboException
02139     *
02140     * @par Python
02141     * servoJoint(self: pyaubo_sdk.MotionControl, arg0: List[float], arg1:
02142     * float, arg2: float, arg3: float, arg4: float, arg5: float) -> int
02143     *
02144     * @par Lua
02145     * servoJoint(q: table, a: number, v: number, t: number, lookahead_time:
02146     * number, gain: number) -> nil
02147     *
02148     * @par Lua
02149     * servoJoint({0,0,0,0,0,0},0,0,10,0,0)
02150     *
02151     * @par JSON-RPC
02152     * {"jsonrpc":"2.0","method":"rob1.MotionControl.servoJoint","params":[[0,0,0,0,0,0],0,0,10,0,0],"id":1}
02153     *
02154     * @par JSON-RPC
02155     * {"id":1,"jsonrpc":"2.0","result":-13}
02156     *
02157     * \endchinese
02158     * \english
02159     * Joint space servo
02160     *
02161     * Currently only q and t parameters are available;
02162     * @param q Joint angles, unit: rad
02163     * @param a Acceleration, unit: rad/s^2
02164     * @param v Velocity, unit: rad/s
02165     * @param t Execution time, unit: s \n
02166     * The larger the t value, the slower the robot moves, and vice versa;
02167     * The optimal value for this parameter is the interval time for continuous
02168     * calls to the servoJoint interface.
```

```
02169     * @param lookahead_time Lookahead time, unit: s \n
02170     * Specifies the duration to move before the robot starts to decelerate,
02171     * used to smooth the trajectory [0.03, 0.2]. When lookahead_time is less
02172     * than one control cycle, the smaller it is, the greater the overshoot. The
02173     * optimal value for this parameter is one control cycle.
02174     * @param gain Proportional gain
02175     * Proportional gain for tracking the target position [100, 200],
02176     * used to control the smoothness and accuracy of the motion.
02177     * The larger the proportional gain, the longer it takes to reach the target
02178     * position, and the smaller the overshoot.
02179     *
02180     * @retval 0 Success
02181     * @retval AUBO_BAD_STATE(1) The current safety mode is not Normal,
02182     * ReducedMode, or Recovery
02183     * @retval AUBO_QUEUE_FULL(2) Planning queue is full
02184     * @retval AUBO_BUSY(3) The previous instruction is being executed
02185     * @retval -AUBO_BAD_STATE(-1)
02186     * Possible reasons include but are not limited to: thread has been
02187     * detached, thread terminated, task_id not found, or the current robot mode
02188     * is not Running
02189     * @retval -AUBO_TIMEOUT(-4) Interface call timeout
02190     * @retval -AUBO_INVL_ARGUMENT(-5) Trajectory position out of range, velocity
02191     * out of range, or adjacent points angle difference exceeded(±15°)
02192     * @retval -AUBO_REQUEST_IGNORE(-13) Not in servo mode
02193     * @retval -AUBO_POS_BOUND_ERR(-18) Joint position exceeds robot maximum limits
02194     * @retval -AUBO_TRAJ_SELF_COLLISION(-22) Trajectory has self-collision risk
02195     * @retval -AUBO_ALGORITHM_PLAN_FAILED(-30) Trajectory planning failed
02196     *
02197     * @throws arcs::common_interface::AuboException
02198     *
02199     * @par Python function prototype
02200     * servoJoint(self: pyaubo_sdk.MotionControl, arg0: List[float], arg1:
02201     * float, arg2: float, arg3: float, arg4: float, arg5: float) -> int
02202     *
02203     * @par Lua function prototype
02204     * servoJoint(q: table, a: number, v: number, t: number, lookahead_time:
02205     * number, gain: number) -> nil
02206     *
02207     * @par Lua example
02208     * servoJoint({0,0,0,0,0,0},0,0,10,0,0)
02209     *
02210     * @par JSON-RPC request example
02211     * {"jsonrpc":"2.0","method":"rob1.MotionControl.servoJoint","params":[[0,0,0,0,0,0],0,0,10,0,0],"id":1}
02212     *
02213     * @par JSON-RPC response example
02214     * {"id":1,"jsonrpc":"2.0","result":-13}
02215     *
02216     * \endenglish
02217     */
02218    int servoJoint(const std::vector<double> &q, double a, double v, double t,
02219                   double lookahead_time, double gain);
02220
02221    /**
02222     * @ingroup MotionControl
02223     * \chinese
02224     *
02225     *
02226     *       pose   t;
02227     * @param pose   ,    m,
02228     * @param a    ,    m/s^2,
02229     * @param v      m/s,
02230     * @param t       s \n
02231     * t   ,     ,     ;
02232     *        servoCartesian
02233     * @param lookahead_time       s \n
02234     *                    [0.03, 0.2],
02235     *   lookahead_time              ,
02236     *
02237     * @param gain
02238     *        [100, 200],
02239     *          ,
02240     *
02241     *
02242     * @retval 0
02243     * @retval AUBO_BAD_STATE(1)
02244     * Normal ReducedMode Recovery
02245     * @retval AUBO_QUEUE_FULL(2)
02246     * @retval AUBO_BUSY(3)
02247     * @retval -AUBO_BAD_STATE(-1)
02248     *                task_id
02249     *           Running
02250     * @retval -AUBO_TIMEOUT(-4)
02251     * @retval -AUBO_INVL_ARGUMENT(-5)
02252     * @retval -AUBO_REQUEST_IGNORE(-13)      servo
02253     * @retval -AUBO_IK_NO_CONVERGE(-23)         ;
02254     * @retval -AUBO_IK_OUT_OF_RANGE(-24)          ;
02255     * @retval AUBO_IK_CONFIG_DISMATCH(-25)        ;
```

```
02256     * @retval -AUBO_IK_JACOBIAN_FAILED(-26)          ;
02257     * @retval -AUBO_IK_NO_SOLU(-27)                  ;
02258     * @retval -AUBO_IK_UNKOWN_ERROR(-28)             ;
02259     *
02260     * @throws arcs::common_interface::AuboException
02261     *
02262     * @par Python
02263     * servoCartesian(self: pyaubo_sdk.MotionControl, arg0: List[float], arg1:
02264     * float, arg2: float, arg3: float, arg4: float, arg5: float) -> int
02265     *
02266     * @par Lua
02267     * servoCartesian(pose: table, a: number, v: number, t: number,
02268     * lookahead_time: number, gain: number) -> nil
02269     *
02270     * @par Lua
02271     * servoCartesian({0.58712,-0.15775,0.48703,2.76,0.344,1.432},0,0,10,0,0)
02272     *
02273     * @par JSON-RPC
02274     * {"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}
02275     *
02276     * @par JSON-RPC
02277     * {"id":1,"jsonrpc":"2.0","result":-13}
02278     * \endchinese
02279     * \english
02280     * Cartesian space servo
02281     *
02282     * Currently, only pose and t parameters are available;
02283     * @param pose Pose, unit: m
02284     * @param a Acceleration, unit: m/s^2
02285     * @param v Velocity, unit: m/s
02286     * @param t Execution time, unit: s \n
02287     * The larger the t value, the slower the robot moves, and vice versa;
02288     * The optimal value for this parameter is the interval time for continuous
02289     * calls to the servoCartesian interface.
02290     * @param lookahead_time Lookahead time, unit: s \n
02291     * Specifies the duration to move before the robot starts to decelerate,
02292     * used to smooth the trajectory [0.03, 0.2]. When lookahead_time is less
02293     * than one control cycle, the smaller it is, the greater the overshoot. The
02294     * optimal value for this parameter is one control cycle.
02295     * @param gain Proportional gain
02296     * Proportional gain for tracking the target position [100, 200],
02297     * used to control the smoothness and accuracy of the motion,
02298     * The larger the proportional gain, the longer it takes to reach the target
02299     * position, and the smaller the overshoot.
02300     *
02301     * @retval 0 Success
02302     * @retval AUBO_BAD_STATE(1) The current safety mode is not Normal,
02303     * ReducedMode, or Recovery
02304     * @retval AUBO_QUEUE_FULL(2) Planning queue is full
02305     * @retval AUBO_BUSY(3) The previous instruction is being executed
02306     * @retval -AUBO_BAD_STATE(-1)
02307     * Possible reasons include but are not limited to: thread has been
02308     * detached, thread terminated, task_id not found, or the current robot mode
02309     * is not Running
02310     * @retval -AUBO_TIMEOUT(-4) Interface call timeout
02311     * @retval -AUBO_INVL_ARGUMENT(-5) Trajectory position or velocity out of
02312     * range
02313     * @retval -AUBO_REQUEST_IGNORE(-13) Not in servo mode
02314     * @retval -AUBO_IK_NO_CONVERGE(-23) Inverse kinematics calculation does not
02315     * converge, calculation error;
02316     * @retval -AUBO_IK_OUT_OF_RANGE(-24) Inverse kinematics calculation exceeds
02317     * robot maximum limits;
02318     * @retval AUBO_IK_CONFIG_DISMATCH(-25) Inverse kinematics input
02319     * configuration error;
02320     * @retval -AUBO_IK_JACOBIAN_FAILED(-26) Inverse kinematics Jacobian
02321     * calculation failed;
02322     * @retval -AUBO_IK_NO_SOLU(-27) Analytical solution exists for the target
02323     * point, but none meet the selection criteria;
02324     * @retval -AUBO_IK_UNKOWN_ERROR(-28) Inverse kinematics returned an unknown
02325     * error type;
02326     *
02327     * @throws arcs::common_interface::AuboException
02328     *
02329     * @par Python function prototype
02330     * servoCartesian(self: pyaubo_sdk.MotionControl, arg0: List[float], arg1:
02331     * float, arg2: float, arg3: float, arg4: float, arg5: float) -> int
02332     *
02333     * @par Lua function prototype
02334     * servoCartesian(pose: table, a: number, v: number, t: number,
02335     * lookahead_time: number, gain: number) -> nil
02336     *
02337     * @par Lua example
02338     * servoCartesian({0.58712,-0.15775,0.48703,2.76,0.344,1.432},0,0,10,0,0)
02339     *
02340     * @par JSON-RPC request example
02341     * {"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}
02342     *
02343     * @par JSON-RPC response example
02344     * {"id":1,"jsonrpc":"2.0","result":-13}
02345     * \endenglish
02346     */
02347     int servoCartesian(const std::vector<double> &pose, double a, double v,
02348                        double t, double lookahead_time, double gain);
02349
02350     /**
02351     * @ingroup MotionControl
02352     * \chinese
02353     *
02354     *
02355     * @param q
02356     * @param extq
02357     * @param t
02358     * @param smooth_scale
02359     * @param delay_sacle
02360     * @return
02361     * \endchinese
02362     * \english
02363     * Servo motion (with external axes), used for executing offline
02364     * trajectories, pass-through user planned trajectories, etc.
02365     *
02366     * @param q
02367     * @param extq
02368     * @param t
02369     * @param smooth_scale
02370     * @param delay_sacle
02371     * @return
02372     * \endenglish
02373     */
02374     int servoJointWithAxes(const std::vector<double> &q,
02375                            const std::vector<double> &extq, double a, double v,
02376                            double t, double lookahead_time, double gain);
02377
02378     int servoJointWithAxisGroup(const std::vector<double> &q, double a,
02379                                 double v, double t, double lookahead_time,
02380                                 double gain, const std::string &group_name,
02381                                 const std::vector<double> &extq);
02382
02383     /**
02384     * @ingroup MotionControl
02385     * \chinese
02386     *
02387     *   servoJointWithAxes
02388     * (       )
02389     *
02390     * @param pose
02391     * @param extq
02392     * @param t
02393     * @param smooth_scale
02394     * @param delay_sacle
02395     * @return
02396     * \endchinese
02397     * \english
02398     * Servo motion (with external axes), used for executing offline
02399     * trajectories, pass-through user planned trajectories, etc. The difference
02400     * from servoJointWithAxes is that it receives Cartesian poses instead of
02401     * joint angles (inverse kinematics is performed internally by the
02402     * software).
02403     *
02404     * @param pose
02405     * @param extq
02406     * @param t
02407     * @param smooth_scale
02408     * @param delay_sacle
02409     * @return
02410     * \endenglish
02411     */
02412     int servoCartesianWithAxes(const std::vector<double> &pose,
02413                                const std::vector<double> &extq, double a,
02414                                double v, double t, double lookahead_time,
02415                                double gain);
02416
02417     int servoCartesianWithAxisGroup(const std::vector<double> &pose, double a,
02418                                     double v, double t, double lookahead_time,
02419                                     double gain, const std::string &group_name,
02420                                     const std::vector<double> &extq);
02421
02422     /**
02423     * @ingroup MotionControl
02424     * \chinese
02425     *
02426     *
02427     * @param q
```

```
02428     * @param smooth_scale
02429     * @param delay_sacle
02430     * @return
02431     *
02432     * @par Lua
02433     * trackJoint(q: table,t: number,smooth_scale: number,delay_sacle: number)
02434     * -> nil
02435     *
02436     * @par Lua
02437     * trackJoint({0,0,0,0,0,0},0.01,0.5,1)
02438     *
02439     * @par JSON-RPC
02440     * {"jsonrpc":"2.0","method":"rob1.MotionControl.trackJoint","params":[[0,0,0,0,0,0],0.01,0.5,1],"id":1}
02441     *
02442     * @par JSON-RPC
02443     * {"id":1,"jsonrpc":"2.0","result":0}
02444     *
02445     * \endchinese
02446     * \english
02447     * Tracking motion, used for executing offline trajectories or passing
02448     * through user-planned trajectories, etc.
02449     *
02450     * @param q
02451     * @param smooth_scale
02452     * @param delay_sacle
02453     * @return
02454     *
02455     * @par Lua function prototype
02456     * trackJoint(q: table,t: number,smooth_scale: number,delay_sacle: number)
02457     * -> nil
02458     *
02459     * @par Lua example
02460     * trackJoint({0,0,0,0,0,0},0.01,0.5,1)
02461     *
02462     * @par JSON-RPC request example
02463     * {"jsonrpc":"2.0","method":"rob1.MotionControl.trackJoint","params":[[0,0,0,0,0,0],0.01,0.5,1],"id":1}
02464     *
02465     * @par JSON-RPC response example
02466     * {"id":1,"jsonrpc":"2.0","result":0}
02467     *
02468     * \endenglish
02469     */
02470    int trackJoint(const std::vector<double> &q, double t, double smooth_scale,
02471                   double delay_sacle);
02472
02473    /**
02474     * @ingroup MotionControl
02475     * \chinese
02476     *
02477     *   trackJoint
02478     * (       )
02479     *
02480     * @param pose
02481     * @param t
02482     * @param smooth_scale
02483     * @param delay_sacle
02484     * @return
02485     *
02486     * @par Lua
02487     * trackCartesian(pose: table,t: number,smooth_scale: number,delay_sacle:
02488     * number) -> nil
02489     *
02490     * @par Lua
02491     * trackCartesian({0.58712,-0.15775,0.48703,2.76,0.344,1.432},0.01,0.5,1)
02492     *
02493     * @par JSON-RPC
02494     * {"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}
02495     *
02496     * @par JSON-RPC
02497     * {"id":1,"jsonrpc":"2.0","result":0}
02498     * \endchinese
02499     * \english
02500     * Tracking motion, used for executing offline trajectories or passing
02501     * through user-planned trajectories, etc. The difference from trackJoint is
02502     * that it receives Cartesian poses instead of joint angles (inverse
02503     * kinematics is performed internally by the software).
02504     *
02505     * @param pose
02506     * @param t
02507     * @param smooth_scale
02508     * @param delay_sacle
02509     * @return
02510     *
02511     * @par Lua function prototype
02512     * trackCartesian(pose: table,t: number,smooth_scale: number,delay_sacle:
02513     * number) -> nil
```

```
02514     *
02515     * @par Lua example
02516     * trackCartesian({0.58712,-0.15775,0.48703,2.76,0.344,1.432},0.01,0.5,1)
02517     *
02518     * @par JSON-RPC request example
02519     * {"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}
02520     *
02521     * @par JSON-RPC response example
02522     * {"id":1,"jsonrpc":"2.0","result":0}
02523     * \endenglish
02524     */
02525    int trackCartesian(const std::vector<double> &pose, double t,
02526                       double smooth_scale, double delay_sacle);
02527
02528    /**
02529     * @ingroup MotionControl
02530     * \chinese
02531     *
02532     *
02533     * @note
02534     *
02535     * @throws arcs::common_interface::AuboException
02536     *
02537     * @par Python
02538     * followJoint(self: pyaubo_sdk.MotionControl, arg0: List[float]) -> int
02539     *
02540     * @par Lua
02541     * followJoint(q: table) -> nil
02542     *
02543     * @par Lua
02544     * followJoint({0,0,0,0,0,0})
02545     *
02546     * \endchinese
02547     * \english
02548     * Joint space following
02549     *
02550     * @note Not implemented yet
02551     *
02552     * @throws arcs::common_interface::AuboException
02553     *
02554     * @par Python function prototype
02555     * followJoint(self: pyaubo_sdk.MotionControl, arg0: List[float]) -> int
02556     *
02557     * @par Lua function prototype
02558     * followJoint(q: table) -> nil
02559     *
02560     * @par Lua example
02561     * followJoint({0,0,0,0,0,0})
02562     *
02563     * \endenglish
02564     */
02565    int followJoint(const std::vector<double> &q);
02566
02567    /**
02568     * @ingroup MotionControl
02569     * \chinese
02570     *
02571     *
02572     * @note
02573     *
02574     * @throws arcs::common_interface::AuboException
02575     *
02576     * @par Python
02577     * followLine(self: pyaubo_sdk.MotionControl, arg0: List[float]) -> int
02578     *
02579     * @par Lua
02580     * followLine(pose: table) -> nil
02581     *
02582     * @par Lua
02583     * followLine({0.58712,-0.15775,0.48703,2.76,0.344,1.432})
02584     *
02585     * \endchinese
02586     * \english
02587     * Cartesian space following
02588     *
02589     * @note Not implemented yet
02590     *
02591     * @throws arcs::common_interface::AuboException
02592     *
02593     * @par Python function prototype
02594     * followLine(self: pyaubo_sdk.MotionControl, arg0: List[float]) -> int
02595     *
02596     * @par Lua function prototype
02597     * followLine(pose: table) -> nil
02598     *
02599     * @par Lua example
```

```
02600     * followLine({0.58712,-0.15775,0.48703,2.76,0.344,1.432})
02601     *
02602     * \endenglish
02603     */
02604    int followLine(const std::vector<double> &pose);
02605
02606    /**
02607     * @ingroup MotionControl
02608     * \chinese
02609     *
02610     *
02611     *
02612     *
02613     * @param qd      ,    rad/s
02614     * @param a       ,    rad/s^2
02615     * @param t         ,    s \n
02616     *    t = 0
02617     *       t                \n
02618     *
02619     *
02620     * @retval 0
02621     * @retval AUBO_BAD_STATE(1)
02622     * Normal ReducedMode Recovery
02623     * @retval AUBO_BUSY(3)
02624     * @retval -AUBO_BAD_STATE(-1)
02625     *                 task_id
02626     *           Running
02627     * @retval -AUBO_TIMEOUT(-4)
02628     * @retval -AUBO_INVL_ARGUMENT(-5)    qd
02629     *
02630     * @throws arcs::common_interface::AuboException
02631     *
02632     * @par Python
02633     * speedJoint(self: pyaubo_sdk.MotionControl, arg0: List[float], arg1:
02634     * float, arg2: float) -> int
02635     *
02636     * @par Lua
02637     * speedJoint(qd: table, a: number, t: number) -> nil
02638     *
02639     * @par Lua
02640     * speedJoint({0.2,0,0,0,0,0}, 1.5,10)
02641     *
02642     * @par JSON-RPC
02643     * {"jsonrpc":"2.0","method":"rob1.MotionControl.speedJoint","params":[[0.2,0,0,0,0,0],1.5,100],"id":1}
02644     *
02645     * @par JSON-RPC
02646     * {"id":1,"jsonrpc":"2.0","result":0}
02647     *
02648     * \endchinese
02649     * \english
02650     * Joint space velocity following
02651     *
02652     * When the robot arm has not yet reached the target velocity, giving a new
02653     * target velocity will cause the robot arm to immediately reach the new
02654     * target velocity.
02655     *
02656     * @param qd Target joint velocity, unit: rad/s
02657     * @param a Main axis acceleration, unit: rad/s^2
02658     * @param t Time required for the function to return, unit: s \n
02659     * If t = 0, the function returns when the target velocity is reached;
02660     * otherwise, the function returns after t seconds, regardless of whether
02661     * the target velocity is reached.\n If the target velocity is not reached,
02662     * it will decelerate to zero. If the target velocity is reached, it will
02663     * move at a constant speed.
02664     * @retval 0 Success
02665     * @retval AUBO_BAD_STATE(1) The current safety mode is not Normal,
02666     * ReducedMode, or Recovery
02667     * @retval AUBO_BUSY(3) The previous instruction is being executed
02668     * @retval -AUBO_BAD_STATE(-1)
02669     * Possible reasons include but are not limited to: thread has been
02670     * detached, thread terminated, task_id not found, or the current robot mode
02671     * is not Running
02672     * @retval -AUBO_TIMEOUT(-4) Interface call timeout
02673     * @retval -AUBO_INVL_ARGUMENT(-5) The length of the qd array is less than
02674     * the degrees of freedom of the current robot arm
02675     *
02676     * @throws arcs::common_interface::AuboException
02677     *
02678     * @par Python function prototype
02679     * speedJoint(self: pyaubo_sdk.MotionControl, arg0: List[float], arg1:
02680     * float, arg2: float) -> int
02681     *
02682     * @par Lua function prototype
02683     * speedJoint(qd: table, a: number, t: number) -> nil
02684     *
02685     * @par Lua example
02686     * speedJoint({0.2,0,0,0,0,0}, 1.5,10)
```

```
02687     *
02688     * @par JSON-RPC request example
02689     * {”jsonrpc”:”2.0”,”method”:”rob1.MotionControl.speedJoint”,”params”:[[0.2,0,0,0,0,0],1.5,100],”id”:1}
02690     *
02691     * @par JSON-RPC response example
02692     * {”id”:1,”jsonrpc”:”2.0”,”result”:0}
02693     *
02694     * \endenglish
02695     */
02696    int speedJoint(const std::vector<double> &qd, double a, double t);
02697
02698    /**
02699     * @ingroup MotionControl
02700     * \chinese
02701     *      (        ,        )
02702     *
02703     *
02704     *
02705     * @param qd     ,    rad/s
02706     * @param a      ,    rad/s^2
02707     * @param t        ,    s
02708     *    t = 0
02709     *      t
02710     *
02711     *
02712     * @return    0;
02713     * AUBO_BUSY
02714     * -AUBO_INVL_ARGUMENT
02715     * -AUBO_BAD_STATE
02716     *
02717     * @throws arcs::common_interface::AuboException
02718     *
02719     * @par Python
02720     * resumeSpeedJoint(self: pyaubo_sdk.MotionControl, arg0: List[float], arg1:
02721     * float, arg2: float) -> int
02722     *
02723     * @par Lua
02724     * resumeSpeedJoint(q: table, a: number, t: number) -> nil
02725     *
02726     * @par Lua
02727     * resumeSpeedJoint({0.2,0,0,0,0,0},1.5,10)
02728     *
02729     * @par JSON-RPC
02730     * {”jsonrpc”:”2.0”,”method”:”rob1.MotionControl.resumeSpeedJoint”,”params”:[[0.2,0,0,0,0,0],1.5,100],”id”:1}
02731     *
02732     * @par JSON-RPC
02733     * {”id”:1,”jsonrpc”:”2.0”,”result”:-1}
02734     * \endchinese
02735     * \english
02736     * Joint space velocity following (used to move to a safe position after a
02737     * collision during process execution)
02738     *
02739     * When the robot arm has not yet reached the target velocity, giving a new
02740     * target velocity will cause the robot arm to immediately reach the new
02741     * target velocity.
02742     *
02743     * @param qd Target joint velocity, unit: rad/s
02744     * @param a Main axis acceleration, unit: rad/s^2
02745     * @param t Time required for the function to return, unit: s
02746     * If t = 0, the function returns when the target velocity is reached;
02747     * otherwise, the function returns after t seconds, regardless of whether
02748     * the target velocity is reached. If the target velocity is not reached, it
02749     * will decelerate to zero. If the target velocity is reached, it will move
02750     * at a constant speed.
02751     * @return Returns 0 on success; otherwise returns error code
02752     * AUBO_BUSY
02753     * -AUBO_INVL_ARGUMENT
02754     * -AUBO_BAD_STATE
02755     *
02756     * @throws arcs::common_interface::AuboException
02757     *
02758     * @par Python function prototype
02759     * resumeSpeedJoint(self: pyaubo_sdk.MotionControl, arg0: List[float], arg1:
02760     * float, arg2: float) -> int
02761     *
02762     * @par Lua function prototype
02763     * resumeSpeedJoint(q: table, a: number, t: number) -> nil
02764     *
02765     * @par Lua example
02766     * resumeSpeedJoint({0.2,0,0,0,0,0},1.5,10)
02767     *
02768     * @par JSON-RPC request example
02769     * {”jsonrpc”:”2.0”,”method”:”rob1.MotionControl.resumeSpeedJoint”,”params”:[[0.2,0,0,0,0,0],1.5,100],”id”:1}
02770     *
02771     * @par JSON-RPC response example
02772     * {”id”:1,”jsonrpc”:”2.0”,”result”:-1}
02773     * \endenglish
```

```
02774     */
02775     int resumeSpeedJoint(const std::vector<double> &qd, double a, double t);
02776
02777     /**
02778      * @ingroup MotionControl
02779      * \chinese
02780      *
02781      *
02782      *
02783      *
02784      * @param xd    ,    m/s
02785      * @param a      ,    m/s^2
02786      * @param t        ,    s \n
02787      *   t = 0
02788      *      t
02789      *
02790      *
02791      * @retval 0
02792      * @retval AUBO_BAD_STATE(1)
02793      * Normal ReducedMode Recovery
02794      * @retval AUBO_BUSY(3)
02795      * @retval -AUBO_BAD_STATE(-1)
02796      *                 task_id
02797      *           Running
02798      * @retval -AUBO_TIMEOUT(-4)
02799      *
02800      * @throws arcs::common_interface::AuboException
02801      *
02802      * @par Python
02803      * speedLine(self: pyaubo_sdk.MotionControl, arg0: List[float], arg1: float,
02804      * arg2: float) -> int
02805      *
02806      * @par Lua
02807      * speedLine(pose: table, a: number, t: number) -> nil
02808      *
02809      * @par Lua
02810      * speedLine({0.25,0,0,0,0,0},1.2,100)
02811      *
02812      * @par JSON-RPC
02813      * {"jsonrpc":"2.0","method":"rob1.MotionControl.speedLine","params":[[0.25,0,0,0,0,0],1.2,100],"id":1}
02814      *
02815      * @par JSON-RPC
02816      * {"id":1,"jsonrpc":"2.0","result":0}
02817      *
02818      * \endchinese
02819      * \english
02820      * Cartesian space velocity following
02821      *
02822      * When the robot arm has not yet reached the target velocity, giving a new
02823      * target velocity will cause the robot arm to immediately reach the new
02824      * target velocity.
02825      *
02826      * @param xd Tool velocity, unit: m/s
02827      * @param a Tool position acceleration, unit: m/s^2
02828      * @param t Time required for the function to return, unit: s \n
02829      * If t = 0, the function returns when the target velocity is reached;
02830      * otherwise, the function returns after t seconds, regardless of whether
02831      * the target velocity is reached. If the target velocity is not reached, it
02832      * will decelerate to zero. If the target velocity is reached, it will move
02833      * at a constant speed.
02834      * @retval 0 Success
02835      * @retval AUBO_BAD_STATE(1) The current safety mode is not Normal,
02836      * ReducedMode, or Recovery
02837      * @retval AUBO_BUSY(3) The previous instruction is being executed
02838      * @retval -AUBO_BAD_STATE(-1)
02839      * Possible reasons include but are not limited to: thread has been
02840      * detached, thread terminated, task_id not found, or the current robot mode
02841      * is not Running
02842      * @retval -AUBO_TIMEOUT(-4) Interface call timeout
02843      *
02844      * @throws arcs::common_interface::AuboException
02845      *
02846      * @par Python function prototype
02847      * speedLine(self: pyaubo_sdk.MotionControl, arg0: List[float], arg1: float,
02848      * arg2: float) -> int
02849      *
02850      * @par Lua function prototype
02851      * speedLine(pose: table, a: number, t: number) -> nil
02852      *
02853      * @par Lua example
02854      * speedLine({0.25,0,0,0,0,0},1.2,100)
02855      *
02856      * @par JSON-RPC request example
02857      * {"jsonrpc":"2.0","method":"rob1.MotionControl.speedLine","params":[[0.25,0,0,0,0,0],1.2,100],"id":1}
02858      *
02859      * @par JSON-RPC response example
02860      * {"id":1,"jsonrpc":"2.0","result":0}
```

```
02861     *
02862     * \endenglish
02863     */
02864    int speedLine(const std::vector<double> &xd, double a, double t);
02865
02866    /**
02867     * @ingroup MotionControl
02868     * \chinese
02869     *       (         ,         )
02870     *
02871     *
02872     *
02873     * @param xd    ,    m/s
02874     * @param a      ,    m/s^2
02875     * @param t        ,    s \n
02876     *   t = 0
02877     *      t
02878     *
02879     *
02880     * @return    0;
02881     * -AUBO_BAD_STATE
02882     *
02883     * @throws arcs::common_interface::AuboException
02884     *
02885     * @par Python
02886     * resumeSpeedLine(self: pyaubo_sdk.MotionControl, arg0: List[float], arg1:
02887     * float, arg2: float) -> int
02888     *
02889     * @par Lua
02890     * resumeSpeedLine(pose: table, a: number, t: number) -> nil
02891     *
02892     * @par Lua
02893     * resumeSpeedLine({0.25,0,0,0,0,0},1.2,100)
02894     *
02895     * @par JSON-RPC
02896     * {"jsonrpc":"2.0","method":"rob1.MotionControl.resumeSpeedLine","params":[[0.25,0,0,0,0,0],1.2,100],"id":1}
02897     *
02898     * @par JSON-RPC
02899     * {"id":1,"jsonrpc":"2.0","result":-1}
02900     * \endchinese
02901     * \english
02902     * Cartesian space velocity following (used to move to a safe position after
02903     * a collision during process execution)
02904     *
02905     * When the robot arm has not yet reached the target velocity, giving a new
02906     * target velocity will cause the robot arm to immediately reach the new
02907     * target velocity.
02908     *
02909     * @param xd Tool velocity, unit: m/s
02910     * @param a Tool position acceleration, unit: m/s^2
02911     * @param t Time required for the function to return, unit: s \n
02912     * If t = 0, the function returns when the target velocity is reached;
02913     * otherwise, the function returns after t seconds, regardless of whether
02914     * the target velocity is reached. If the target velocity is not reached, it
02915     * will decelerate to zero. If the target velocity is reached, it will move
02916     * at a constant speed.
02917     * @return Returns 0 on success; otherwise returns error code
02918     * -AUBO_BAD_STATE
02919     *
02920     * @throws arcs::common_interface::AuboException
02921     *
02922     * @par Python function prototype
02923     * resumeSpeedLine(self: pyaubo_sdk.MotionControl, arg0: List[float], arg1:
02924     * float, arg2: float) -> int
02925     *
02926     * @par Lua function prototype
02927     * resumeSpeedLine(pose: table, a: number, t: number) -> nil
02928     *
02929     * @par Lua example
02930     * resumeSpeedLine({0.25,0,0,0,0,0},1.2,100)
02931     *
02932     * @par JSON-RPC request example
02933     * {"jsonrpc":"2.0","method":"rob1.MotionControl.resumeSpeedLine","params":[[0.25,0,0,0,0,0],1.2,100],"id":1}
02934     *
02935     * @par JSON-RPC response example
02936     * {"id":1,"jsonrpc":"2.0","result":-1}
02937     * \endenglish
02938     */
02939    int resumeSpeedLine(const std::vector<double> &xd, double a, double t);
02940
02941    /**
02942     * @ingroup MotionControl
02943     * \chinese
02944     *
02945     *
02946     * @param q          0
02947     * @param a    ,    rad/s^2,
```

```
02948     *     RobotConfig    getJointMaxAccelerations()
02949     * @param v       rad/s,
02950     *     RobotConfig    getJointMaxSpeeds()
02951     * @param duration
02952     * @return    0;
02953     * AUBO_BUSY
02954     * AUBO_BAD_STATE
02955     * -AUBO_BAD_STATE
02956     *
02957     * @throws arcs::common_interface::AuboException
02958     *
02959     * @par Python
02960     * moveSpline(self: pyaubo_sdk.MotionControl, arg0: List[float], arg1:
02961     * float, arg2: float, arg3: float) -> int
02962     *
02963     * @par Lua
02964     * moveSpline(q: table, a: number, v: number, duration: number) -> nil
02965     *
02966     * @par Lua
02967     * moveSpline({-2.05177,-0.400292, 1.19625, 0.0285152, 1.57033,
02968     * -2.28774},1.3,1.0,0)
02969     *
02970     * @par JSON-RPC
02971     * {"jsonrpc":"2.0","method":"rob1.MotionControl.moveSpline","params":[[0,0,0,0,0,0],1,1,0],"id":1}
02972     *
02973     * @par JSON-RPC
02974     * {"id":1,"jsonrpc":"2.0","result":0}
02975     * \endchinese
02976     * \english
02977     * Perform spline interpolation in joint space
02978     *
02979     * @param q Joint angles. If the input vector is of size 0, it indicates the
02980     * end of the spline motion.
02981     * @param a Acceleration, unit: rad/s^2. The maximum value can be obtained
02982     * via RobotConfig::getJointMaxAccelerations().
02983     * @param v Velocity, unit: rad/s. The maximum value can be obtained via
02984     * RobotConfig::getJointMaxSpeeds().
02985     * @param duration
02986     * @return Returns 0 on success; otherwise returns an error code:
02987     * AUBO_BUSY
02988     * AUBO_BAD_STATE
02989     * -AUBO_BAD_STATE
02990     *
02991     * @throws arcs::common_interface::AuboException
02992     *
02993     * @par Python function prototype
02994     * moveSpline(self: pyaubo_sdk.MotionControl, arg0: List[float], arg1:
02995     * float, arg2: float, arg3: float) -> int
02996     *
02997     * @par Lua function prototype
02998     * moveSpline(q: table, a: number, v: number, duration: number) -> nil
02999     *
03000     * @par Lua example
03001     * moveSpline({-2.05177,-0.400292, 1.19625, 0.0285152, 1.57033,
03002     * -2.28774},1.3,1.0,0)
03003     *
03004     * @par JSON-RPC request example
03005     * {"jsonrpc":"2.0","method":"rob1.MotionControl.moveSpline","params":[[0,0,0,0,0,0],1,1,0],"id":1}
03006     *
03007     * @par JSON-RPC response example
03008     * {"id":1,"jsonrpc":"2.0","result":0}
03009     * \endenglish
03010     */
03011    int moveSpline(const std::vector<double> &q, double a, double v,
03012                   double duration);
03013
03014    /**
03015     * @ingroup MotionControl
03016     * \chinese
03017     *
03018     *
03019     * @param q    ,    rad
03020     * @param a    ,    rad/s^2,
03021     *     RobotConfig    getJointMaxAccelerations()
03022     * @param v       rad/s,
03023     *     RobotConfig    getJointMaxSpeeds()
03024     * @param blend_radius    ,    m
03025     * @param duration       s \n
03026     *
03027     *             duration
03028     * duration                \n
03029     *   duration = 0
03030     *
03031     *   duration   0 a    v
03032     * @retval 0
03033     * @retval AUBO_BAD_STATE(1)
03034     * Normal ReducedMode Recovery
```

```
03035     * @retval AUBO_QUEUE_FULL(2)
03036     * @retval AUBO_BUSY(3)
03037     * @retval -AUBO_BAD_STATE(-1)
03038     *                 task_id
03039     *           Running
03040     * @retval -AUBO_TIMEOUT(-4)
03041     * @retval -AUBO_INVL_ARGUMENT(-5)    q
03042     * @retval AUBO_REQUEST_IGNORE(13)    q
03043     *
03044     * @throws arcs::common_interface::AuboException
03045     *
03046     * @par Python
03047     * moveJoint(self: pyaubo_sdk.MotionControl, arg0: List[float], arg1: float,
03048     * arg2: float, arg3: float, arg4: float) -> int
03049     *
03050     * @par Lua
03051     * moveJoint(q: table, a: number, v: number, blend_radius: number, duration:
03052     * number) -> nil
03053     *
03054     * @par Lua
03055     * moveJoint({-2.05177,-0.400292, 1.19625, 0.0285152, 1.57033,
03056     * -2.28774},0.3,0.3,0,0)
03057     *
03058     * @par JSON-RPC
03059     * {"jsonrpc":"2.0","method":"rob1.MotionControl.moveJoint","params":[[-2.05177,
03060     * -0.400292, 1.19625, 0.0285152, 1.57033, -2.28774],0.3,0.3,0,0],"id":1}
03061     *
03062     * @par JSON-RPC
03063     * {"id":1,"jsonrpc":"2.0","result":0}
03064     *
03065     * \endchinese
03066     * \english
03067     * Add joint motion
03068     *
03069     * @param q Joint angles, unit: rad
03070     * @param a Acceleration, unit: rad/s^2,
03071     * The maximum value can be obtained via
03072     * RobotConfig::getJointMaxAccelerations()
03073     * @param v Velocity, unit: rad/s,
03074     * The maximum value can be obtained via RobotConfig::getJointMaxSpeeds()
03075     * @param blend_radius Blend radius, unit: m
03076     * @param duration Execution time, unit: s \n
03077     * Normally, when speed and acceleration are given, the trajectory duration
03078     * can be determined. If you want to extend the trajectory duration, set the
03079     * duration parameter. Duration can extend the trajectory time, but cannot
03080     * shorten it.\n When duration = 0, it means the execution time is not
03081     * specified, and the time will be calculated based on speed and
03082     * acceleration. If duration is not 0, a and v values will be ignored.
03083     * @retval 0 Success
03084     * @retval AUBO_BAD_STATE(1) The current safety mode is not Normal,
03085     * ReducedMode, or Recovery
03086     * @retval AUBO_QUEUE_FULL(2) Planning queue is full
03087     * @retval AUBO_BUSY(3) The previous instruction is being executed
03088     * @retval -AUBO_BAD_STATE(-1)
03089     * Possible reasons include but are not limited to: thread has been
03090     * detached, thread terminated, task_id not found, or the current robot mode
03091     * is not Running
03092     * @retval -AUBO_TIMEOUT(-4) Interface call timeout
03093     * @retval -AUBO_INVL_ARGUMENT(-5) The length of q is less than the degrees
03094     * of freedom of the current robot arm
03095     * @retval AUBO_REQUEST_IGNORE(13) The length of q is too short and there is
03096     * no need to stop at that point
03097     *
03098     * @throws arcs::common_interface::AuboException
03099     *
03100     * @par Python function prototype
03101     * moveJoint(self: pyaubo_sdk.MotionControl, arg0: List[float], arg1: float,
03102     * arg2: float, arg3: float, arg4: float) -> int
03103     *
03104     * @par Lua function prototype
03105     * moveJoint(q: table, a: number, v: number, blend_radius: number, duration:
03106     * number) -> nil
03107     *
03108     * @par Lua example
03109     * moveJoint({-2.05177,-0.400292, 1.19625, 0.0285152, 1.57033,
03110     * -2.28774},0.3,0.3,0,0)
03111     *
03112     * @par JSON-RPC request example
03113     * {"jsonrpc":"2.0","method":"rob1.MotionControl.moveJoint","params":[[-2.05177,
03114     * -0.400292, 1.19625, 0.0285152, 1.57033, -2.28774],0.3,0.3,0,0],"id":1}
03115     *
03116     * @par JSON-RPC response example
03117     * {"id":1,"jsonrpc":"2.0","result":0}
03118     *
03119     * \endenglish
03120     */
03121    int moveJoint(const std::vector<double> &q, double a, double v,
```

```
03122                 double blend_radius, double duration);
03123
03124     /**
03125      * @ingroup MotionControl
03126      * \chinese
03127      *
03128      *
03129      * @param group_name
03130      * @param q
03131      * @param a
03132      * @param v
03133      * @param blend_radius
03134      * @param duration
03135      * @return
03136      * \endchinese
03137      * \english
03138      * Synchronous motion of robot and external axes
03139      *
03140      * @param group_name
03141      * @param q
03142      * @param a
03143      * @param v
03144      * @param blend_radius
03145      * @param duration
03146      * @return
03147      * \endenglish
03148      */
03149     int moveJointWithAxisGroup(const std::vector<double> &q, double a, double v,
03150                                double blend_radius, double duration,
03151                                const std::string &group_name,
03152                                const std::vector<double> &extq);
03153
03154     /**
03155      * @ingroup MotionControl
03156      * \chinese
03157      *
03158      *
03159      * @param q    ,    rad
03160      * @param a    ,    rad/s^2
03161      * @param v       rad/s
03162      * @param duration       s \n
03163      *
03164      *             duration
03165      * duration                \n
03166      *   duration = 0
03167      *
03168      *   duration   0 a    v
03169      * @return    0
03170      * -AUBO_INVL_ARGUMENT
03171      * -AUBO_BAD_STATE
03172      *
03173      * @throws arcs::common_interface::AuboException
03174      *
03175      * @par Python
03176      * resumeMoveJoint(self: pyaubo_sdk.MotionControl, arg0: List[float], arg1:
03177      * float, arg2: float, arg3: float) -> int
03178      *
03179      * @par Lua
03180      * resumeMoveJoint(q: table, a: number, v: number, duration: number) -> nil
03181      *
03182      * @par Lua
03183      * resumeMoveJoint({-2.05177,-0.400292, 1.19625, 0.0285152, 1.57033,
03184      * -2.28774},1.3,1.0,0)
03185      *
03186      * @par JSON-RPC
03187      * {"jsonrpc":"2.0","method":"rob1.MotionControl.resumeMoveJoint","params":[[-2.05177,
03188      * -0.400292, 1.19625, 0.0285152, 1.57033, -2.28774],0.3,0.3,0],"id":1}
03189      *
03190      * @par JSON-RPC
03191      * {"id":1,"jsonrpc":"2.0","result":-1}
03192      * \endchinese
03193      * \english
03194      * Move to the pause point using joint motion.
03195      *
03196      * @param q Joint angles, unit: rad
03197      * @param a Acceleration, unit: rad/s^2
03198      * @param v Velocity, unit: rad/s
03199      * @param duration Execution time, unit: s \n
03200      * Normally, when speed and acceleration are given, the trajectory duration
03201      * can be determined. If you want to extend the trajectory duration, set the
03202      * duration parameter. Duration can extend the trajectory time, but cannot
03203      * shorten it.\n When duration = 0, it means the execution time is not
03204      * specified, and the time will be calculated based on speed and
03205      * acceleration. If duration is not 0, a and v values will be ignored.
03206      * @return Returns 0 on success; otherwise returns error code
03207      * -AUBO_INVL_ARGUMENT
03208      * -AUBO_BAD_STATE
```

```
03209     *
03210     * @throws arcs::common_interface::AuboException
03211     *
03212     * @par Python function prototype
03213     * resumeMoveJoint(self: pyaubo_sdk.MotionControl, arg0: List[float], arg1:
03214     * float, arg2: float, arg3: float) -> int
03215     *
03216     * @par Lua function prototype
03217     * resumeMoveJoint(q: table, a: number, v: number, duration: number) -> nil
03218     *
03219     * @par Lua example
03220     * resumeMoveJoint({-2.05177,-0.400292, 1.19625, 0.0285152, 1.57033,
03221     * -2.28774},1.3,1.0,0)
03222     *
03223     * @par JSON-RPC request example
03224     * {"jsonrpc":"2.0","method":"rob1.MotionControl.resumeMoveJoint","params":[[-2.05177,
03225     * -0.400292, 1.19625, 0.0285152, 1.57033, -2.28774],0.3,0.3,0],"id":1}
03226     *
03227     * @par JSON-RPC response example
03228     * {"id":1,"jsonrpc":"2.0","result":-1}
03229     * \endenglish
03230     */
03231    int resumeMoveJoint(const std::vector<double> &q, double a, double v,
03232                        double duration);
03233
03234    /**
03235     * @ingroup MotionControl
03236     * \chinese
03237     *
03238     *
03239     * @param pose
03240     * @param a   (      1mm,      1e-4
03241     * rad,          ,   rad/s^2.      ,   m/s^2)
03242     *      RobotConfig    getTcpMaxAccelerations()
03243     * @param v   (      1mm,      1e-4
03244     * rad,        ,   rad/s.      ,   m/s)
03245     *      RobotConfig    getTcpMaxSpeeds()
03246     * @param blend_radius        m    0.001 1
03247     * @param duration        s \n
03248     *
03249     *              duration
03250     * duration                    \n
03251     *   duration = 0
03252     *
03253     *   duration   0 a    v
03254     * @retval 0
03255     * @retval AUBO_BAD_STATE(1)
03256     * Normal ReducedMode Recovery
03257     * @retval AUBO_QUEUE_FULL(2)
03258     * @retval AUBO_BUSY(3)
03259     * @retval -AUBO_BAD_STATE(-1)
03260     *                  task_id
03261     *            Running
03262     * @retval -AUBO_TIMEOUT(-4)
03263     * @retval -AUBO_INVL_ARGUMENT(-5)      pose
03264     *
03265     * @retval AUBO_REQUEST_IGNORE(13)      pose
03266     *
03267     *
03268     * @throws arcs::common_interface::AuboException
03269     *
03270     * @par Python
03271     * moveLine(self: pyaubo_sdk.MotionControl, arg0: List[float], arg1: float,
03272     * arg2: float, arg3: float, arg4: float) -> int
03273     *
03274     * @par Lua
03275     * moveLine(pose: table, a: number, v: number, blend_radius: number,
03276     * duration: number) -> nil
03277     *
03278     * @par Lua
03279     * moveLine({0.58815,0.0532,0.62391,2.46,0.479,1.619},1.3,1.0,0)
03280     *
03281     * @par JSON-RPC
03282     * {"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}
03283     *
03284     * @par JSON-RPC
03285     * {"id":1,"jsonrpc":"2.0","result":0}
03286     *
03287     * \endchinese
03288     * \english
03289     * Add linear motion
03290     *
03291     * @param pose Target pose
03292     * @param a Acceleration (if the position change is less than 1mm and the
03293     * posture change is greater than 1e-4 rad, this acceleration is treated as
03294     * angular acceleration in rad/s^2; otherwise, as linear acceleration in
```

```
03295     * m/s^2). The maximum value can be obtained via
03296     * RobotConfig::getTcpMaxAccelerations().
03297     * @param v Velocity (if the position change is less than 1mm and the
03298     * posture change is greater than 1e-4 rad, this velocity is treated as
03299     * angular velocity in rad/s; otherwise, as linear velocity in m/s). The
03300     * maximum value can be obtained via RobotConfig::getTcpMaxSpeeds().
03301     * @param blend_radius Blend radius, unit: m, value between 0.001 and 1
03302     * @param duration Execution time, unit: s \n
03303     * Normally, when speed and acceleration are given, the trajectory duration
03304     * can be determined. If you want to extend the trajectory duration, set the
03305     * duration parameter. Duration can extend the trajectory time, but cannot
03306     * shorten it.\n When duration = 0, it means the execution time is not
03307     * specified, and the time will be calculated based on speed and
03308     * acceleration. If duration is not 0, a and v values will be ignored.
03309     * @retval 0 Success
03310     * @retval AUBO_BAD_STATE(1) The current safety mode is not Normal,
03311     * ReducedMode, or Recovery
03312     * @retval AUBO_QUEUE_FULL(2) Trajectory queue is full
03313     * @retval AUBO_BUSY(3) The previous instruction is being executed
03314     * @retval -AUBO_BAD_STATE(-1) Possible reasons include but are not limited
03315     * to: thread has been detached, thread terminated, task_id not found, or
03316     * the current robot mode is not Running
03317     * @retval -AUBO_TIMEOUT(-4) Interface call timeout
03318     * @retval -AUBO_INVL_ARGUMENT(-5) The length of the pose array is less than
03319     * the degrees of freedom of the current robot arm
03320     * @retval AUBO_REQUEST_IGNORE(13) The length of the pose array is too short
03321     * and there is no need to stop at that point
03322     *
03323     * @throws arcs::common_interface::AuboException
03324     *
03325     * @par Python function prototype
03326     * moveLine(self: pyaubo_sdk.MotionControl, arg0: List[float], arg1: float,
03327     * arg2: float, arg3: float, arg4: float) -> int
03328     *
03329     * @par Lua function prototype
03330     * moveLine(pose: table, a: number, v: number, blend_radius: number,
03331     * duration: number) -> nil
03332     *
03333     * @par Lua example
03334     * moveLine({0.58815,0.0532,0.62391,2.46,0.479,1.619},1.3,1.0,0)
03335     *
03336     * @par JSON-RPC request example
03337     * {"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}
03338     *
03339     * @par JSON-RPC response example
03340     * {"id":1,"jsonrpc":"2.0","result":0}
03341     *
03342     * \endenglish
03343     */
03344    int moveLine(const std::vector<double> &pose, double a, double v,
03345                 double blend_radius, double duration);
03346
03347    /**
03348     * @ingroup MotionControl
03349     * \chinese
03350     *
03351     *
03352     * @param group_name
03353     * @param pose
03354     * @param a
03355     * @param v
03356     * @param blend_radius
03357     * @param duration
03358     * @return
03359     * \endchinese
03360     * \english
03361     * Linear motion synchronized with external axes
03362     *
03363     * @param group_name Name of the external axis group
03364     * @param pose Target pose
03365     * @param a Acceleration
03366     * @param v Velocity
03367     * @param blend_radius Blend radius
03368     * @param duration Execution time
03369     * @return
03370     * \endenglish
03371     */
03372    int moveLineWithAxisGroup(const std::vector<double> &pose, double a,
03373                              double v, double blend_radius, double duration,
03374                              const std::string &group_name,
03375                              const std::vector<double> &extq);
03376
03377    /**
03378     * @ingroup MotionControl
03379     * \chinese
03380     *
```

```
03381     *
03382     * @param pose
03383     * @param a
03384     * @param v
03385     * @param blend_radius
03386     * @return    0
03387     * AUBO_BAD_STATE
03388     * AUBO_BUSY
03389     * -AUBO_INVL_ARGUMENT
03390     * -AUBO_BAD_STATE
03391     *
03392     * @throws arcs::common_interface::AuboException
03393     *
03394     * @par Lua
03395     * moveProcess(pose: table, a: number, v: number, blend_radius: number,
03396     * duration: number) -> nil
03397     *
03398     * @par Lua
03399     * moveProcess({0.58815,0.0532,0.62391,2.46,0.479,1.619},1.2,0.25,0)
03400     *
03401     * @par JSON-RPC
03402     * {"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}
03403     *
03404     * @par JSON-RPC
03405     * {"id":1,"jsonrpc":"2.0","result":0}
03406     * \endchinese
03407     * \english
03408     * Add process motion
03409     *
03410     * @param pose
03411     * @param a
03412     * @param v
03413     * @param blend_radius
03414     * @return Returns 0 on success; otherwise returns an error code:
03415     * AUBO_BAD_STATE
03416     * AUBO_BUSY
03417     * -AUBO_INVL_ARGUMENT
03418     * -AUBO_BAD_STATE
03419     *
03420     * @throws arcs::common_interface::AuboException
03421     *
03422     * @par Lua function prototype
03423     * moveProcess(pose: table, a: number, v: number, blend_radius: number,
03424     * duration: number) -> nil
03425     *
03426     * @par Lua example
03427     * moveProcess({0.58815,0.0532,0.62391,2.46,0.479,1.619},1.2,0.25,0)
03428     *
03429     * @par JSON-RPC request example
03430     * {"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}
03431     *
03432     * @par JSON-RPC response example
03433     * {"id":1,"jsonrpc":"2.0","result":0}
03434     * \endenglish
03435     */
03436    int moveProcess(const std::vector<double> &pose, double a, double v,
03437                    double blend_radius);
03438
03439    /**
03440     * @ingroup MotionControl
03441     * \chinese
03442     *
03443     *
03444     * @param pose
03445     * @param a    ,    m/s^2
03446     * @param v    ,    m/s
03447     * @param duration       s \n
03448     *
03449     *              duration
03450     * duration                  \n
03451     *   duration = 0
03452     *
03453     *   duration   0 a    v
03454     * @return    0
03455     * -AUBO_INVL_ARGUMENT
03456     * -AUBO_BAD_STATE
03457     *
03458     * @throws arcs::common_interface::AuboException
03459     *
03460     * @par Python
03461     * resumeMoveLine(self: pyaubo_sdk.MotionControl, arg0: List[float], arg1:
03462     * float, arg2: float, arg3: float) -> int
03463     *
03464     * @par Lua
03465     * resumeMoveLine(pose: table, a: number, v: number,duration: number) -> nil
```

```
03466     *
03467     * @par Lua
03468     * resumeMoveLine({0.58815,0.0532,0.62391,2.46,0.479,1.619},1.2,0.25,0)
03469     *
03470     * @par JSON-RPC
03471     * {"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}
03472     *
03473     * @par JSON-RPC
03474     * {"id":1,"jsonrpc":"2.0","result":0}
03475     *
03476     * \endchinese
03477     * \english
03478     * Move to the pause point using linear motion.
03479     *
03480     * @param pose Target pose
03481     * @param a Acceleration, unit: m/s^2
03482     * @param v Velocity, unit: m/s
03483     * @param duration Execution time, unit: s \n
03484     * Normally, when speed and acceleration are given, the trajectory duration
03485     * can be determined. If you want to extend the trajectory duration, set the
03486     * duration parameter. Duration can extend the trajectory time, but cannot
03487     * shorten it.\n When duration = 0, it means the execution time is not
03488     * specified, and the time will be calculated based on speed and
03489     * acceleration. If duration is not 0, a and v values will be ignored.
03490     * @return Returns 0 on success; otherwise returns error code
03491     * -AUBO_INVL_ARGUMENT
03492     * -AUBO_BAD_STATE
03493     *
03494     * @throws arcs::common_interface::AuboException
03495     *
03496     * @par Python function prototype
03497     * resumeMoveLine(self: pyaubo_sdk.MotionControl, arg0: List[float], arg1:
03498     * float, arg2: float, arg3: float) -> int
03499     *
03500     * @par Lua function prototype
03501     * resumeMoveLine(pose: table, a: number, v: number, duration: number) ->
03502     * nil
03503     *
03504     * @par Lua example
03505     * resumeMoveLine({0.58815,0.0532,0.62391,2.46,0.479,1.619},1.2,0.25,0)
03506     *
03507     * @par JSON-RPC request example
03508     * {"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}
03509     *
03510     * @par JSON-RPC response example
03511     * {"id":1,"jsonrpc":"2.0","result":0}
03512     *
03513     * \endenglish
03514     */
03515    int resumeMoveLine(const std::vector<double> &pose, double a, double v,
03516                       double duration);
03517
03518    /**
03519     * @ingroup MotionControl
03520     * \chinese
03521     *
03522     *
03523     * @todo
03524     *
03525     * @param via_pose
03526     * @param end_pose
03527     * @param a    ( via_pose         1mm,      1e-4
03528     * rad,           ,   rad/s^2.      ,   m/s^2)
03529     * @param v   ( via_pose         1mm,      1e-4
03530     * rad,          ,    rad / s.     ,    m / s)
03531     * @param blend_radius      : m
03532     * @param duration      : s \n
03533     *
03534     *              duration
03535     * duration                  \n
03536     *   duration = 0
03537     *
03538     *   duration   0 a   v
03539     * @return    0
03540     * AUBO_BAD_STATE
03541     * AUBO_BUSY
03542     * -AUBO_INVL_ARGUMENT
03543     * -AUBO_BAD_STATE
03544     *
03545     * @throws arcs::common_interface::AuboException
03546     *
03547     * @par Python
03548     * moveCircle(self: pyaubo_sdk.MotionControl, arg0: List[float], arg1:
03549     * List[float], arg2: float, arg3: float, arg4: float, arg5: float) -> int
03550     *
```

```
03551     * @par Lua
03552     * moveCircle(via_pose: table, end_pose: table, a: number, v: number,
03553     * blend_radius: number, duration: number) -> nil
03554     *
03555     * @par Lua
03556     *
    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)
03557     *
03558     * @par JSON-RPC
03559     * {"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}
03560     *
03561     * @par JSON-RPC
03562     * {"id":1,"jsonrpc":"2.0","result":0}
03563     * \endchinese
03564     * \english
03565     * Add circular arc motion
03566     *
03567     * @todo Can be composed of multiple arcs to form a circular motion
03568     *
03569     * @param via_pose The pose of the via point during the arc motion
03570     * @param end_pose The pose of the end point of the arc motion
03571     * @param a Acceleration (if the position change between via_pose and the
03572     * previous waypoint is less than 1mm and the posture change is greater than
03573     * 1e-4 rad, this acceleration is treated as angular acceleration in
03574     * rad/s^2; otherwise, as linear acceleration in m/s^2)
03575     * @param v Velocity (if the position change between via_pose and the
03576     * previous waypoint is less than 1mm and the posture change is greater than
03577     * 1e-4 rad, this velocity is treated as angular velocity in rad/s;
03578     * otherwise, as linear velocity in m/s)
03579     * @param blend_radius Blend radius, unit: m
03580     * @param duration Execution time, unit: s \n
03581     * Normally, when speed and acceleration are given, the trajectory duration
03582     * can be determined. If you want to extend the trajectory duration, set the
03583     * duration parameter. Duration can extend the trajectory time, but cannot
03584     * shorten it.\n When duration = 0, it means the execution time is not
03585     * specified, and the time will be calculated based on speed and
03586     * acceleration. If duration is not 0, a and v values will be ignored.
03587     * @return Returns 0 on success; otherwise returns error code
03588     * AUBO_BAD_STATE
03589     * AUBO_BUSY
03590     * -AUBO_INVL_ARGUMENT
03591     * -AUBO_BAD_STATE
03592     *
03593     * @throws arcs::common_interface::AuboException
03594     *
03595     * @par Python function prototype
03596     * moveCircle(self: pyaubo_sdk.MotionControl, arg0: List[float], arg1:
03597     * List[float], arg2: float, arg3: float, arg4: float, arg5: float) -> int
03598     *
03599     * @par Lua function prototype
03600     * moveCircle(via_pose: table, end_pose: table, a: number, v: number,
03601     * blend_radius: number, duration: number) -> nil
03602     *
03603     * @par Lua example
03604     *
    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)
03605     *
03606     * @par JSON-RPC request example
03607     * {"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}
03608     *
03609     * @par JSON-RPC response example
03610     * {"id":1,"jsonrpc":"2.0","result":0}
03611     * \endenglish
03612     */
03613    int moveCircle(const std::vector<double> &via_pose,
03614                   const std::vector<double> &end_pose, double a, double v,
03615                   double blend_radius, double duration);
03616
03617    /**
03618     * @ingroup MotionControl
03619     * \chinese
03620     * moveCircle
03621     *
03622     * @param group_name
03623     * @param via_pose
03624     * @param end_pose
03625     * @param a
03626     * @param v
03627     * @param blend_radius
03628     * @param duration
03629     * @return
03630     * \endchinese
03631     * \english
03632     * moveCircle with external axes synchronized motion
03633     *
```

```
03634     * @param group_name Name of the external axis group
03635     * @param via_pose The pose of the via point during the arc motion
03636     * @param end_pose The pose of the end point of the arc motion
03637     * @param a Acceleration
03638     * @param v Velocity
03639     * @param blend_radius Blend radius
03640     * @param duration Execution time
03641     * @return
03642     * \endenglish
03643     */
03644    int moveCircleWithAxisGroup(const std::vector<double> &via_pose,
03645                                const std::vector<double> &end_pose, double a,
03646                                double v, double blend_radius, double duration,
03647                                const std::string &group_name,
03648                                const std::vector<double> &extq);
03649
03650    /**
03651     * @ingroup MotionControl
03652     * \chinese
03653     *
03654     *
03655     * @param mode    \n
03656     * 0:              \n
03657     * 1:    ,     ,          \n
03658     * 2:          ,
03659     * @return    0
03660     * AUBO_BAD_STATE
03661     * AUBO_BUSY
03662     * -AUBO_INVL_ARGUMENT
03663     * -AUBO_BAD_STATE
03664     *
03665     * @throws arcs::common_interface::AuboException
03666     *
03667     * @par Python
03668     * setCirclePathMode(self: pyaubo_sdk.MotionControl, arg0: int) -> int
03669     *
03670     * @par Lua
03671     * setCirclePathMode(mode: number) -> nil
03672     *
03673     * @par Lua
03674     * setCirclePathMode(1)
03675     *
03676     * @par JSON-RPC
03677     * {"jsonrpc":"2.0","method":"rob1.MotionControl.setCirclePathMode","params":[0],"id":1}
03678     *
03679     * @par JSON-RPC
03680     * {"id":1,"jsonrpc":"2.0","result":0}
03681     *
03682     * \endchinese
03683     * \english
03684     * Set circle path mode
03685     *
03686     * @param mode Mode \n
03687     * 0: Tool orientation remains unchanged relative to the arc path point
03688     * coordinate system \n 1: Orientation changes linearly, rotating around a
03689     * fixed axis in space, from the start orientation to the target orientation
03690     * \n 2: Orientation changes from the start orientation, through the via
03691     * point orientation, to the target orientation
03692     * @return Returns 0 on success; otherwise returns an error code
03693     * AUBO_BAD_STATE
03694     * AUBO_BUSY
03695     * -AUBO_INVL_ARGUMENT
03696     * -AUBO_BAD_STATE
03697     *
03698     * @throws arcs::common_interface::AuboException
03699     *
03700     * @par Python function prototype
03701     * setCirclePathMode(self: pyaubo_sdk.MotionControl, arg0: int) -> int
03702     *
03703     * @par Lua function prototype
03704     * setCirclePathMode(mode: number) -> nil
03705     *
03706     * @par Lua example
03707     * setCirclePathMode(1)
03708     *
03709     * @par JSON-RPC request example
03710     * {"jsonrpc":"2.0","method":"rob1.MotionControl.setCirclePathMode","params":[0],"id":1}
03711     *
03712     * @par JSON-RPC response example
03713     * {"id":1,"jsonrpc":"2.0","result":0}
03714     *
03715     * \endenglish
03716     */
03717    int setCirclePathMode(int mode);
03718
03719    /**
03720     * @ingroup MotionControl
```

```
03721     * \chinese
03722     *
03723     *
03724     * @param param
03725     * @return    0
03726     * AUBO_BAD_STATE
03727     * AUBO_BUSY
03728     * -AUBO_INVL_ARGUMENT
03729     * -AUBO_BAD_STATE
03730     *
03731     * @throws arcs::common_interface::AuboException
03732     *
03733     * @par Python
03734     * moveCircle2(self: pyaubo_sdk.MotionControl, arg0:
03735     * arcs::common_interface::CircleParameters) -> int
03736     *
03737     * @par Lua
03738     * moveCircle2(param: table) -> nil
03739     *
03740     * @par Lua
03741     * moveCircle2({0.440164,-
0.00249391,0.398658,2.45651,0,1.570},{0.440164,0.166256,0.297599,2.45651,0,1.5708},1.2,0.25,0,0,0,0,0,0)
03742     *
03743     * @par JSON-RPC
03744     * {"jsonrpc":"2.0","method":"rob1.MotionControl.moveCircle2","params":[{"pose_via":[0.440164,-
0.00249391,0.398658,2.45651,0,1.570],
03745     * "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,
03746     * "spiral":0,"direction":0,"loop_times":0}],"id":1}
03747     *
03748     * @par JSON-RPC
03749     * {"id":1,"jsonrpc":"2.0","result":0}
03750     * \endchinese
03751     * \english
03752     * Advanced arc or circular motion
03753     *
03754     * @param param Circular motion parameters
03755     * @return Returns 0 on success; otherwise returns an error code:
03756     * AUBO_BAD_STATE
03757     * AUBO_BUSY
03758     * -AUBO_INVL_ARGUMENT
03759     * -AUBO_BAD_STATE
03760     *
03761     * @throws arcs::common_interface::AuboException
03762     *
03763     * @par Python function prototype
03764     * moveCircle2(self: pyaubo_sdk.MotionControl, arg0:
03765     * arcs::common_interface::CircleParameters) -> int
03766     *
03767     * @par Lua function prototype
03768     * moveCircle2(param: table) -> nil
03769     *
03770     * @par Lua example
03771     * moveCircle2({0.440164,-
0.00249391,0.398658,2.45651,0,1.570},{0.440164,0.166256,0.297599,2.45651,0,1.5708},1.2,0.25,0,0,0,0,0,0)
03772     *
03773     * @par JSON-RPC request example
03774     * {"jsonrpc":"2.0","method":"rob1.MotionControl.moveCircle2","params":[{"pose_via":[0.440164,-
0.00249391,0.398658,2.45651,0,1.570],
03775     * "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,
03776     * "spiral":0,"direction":0,"loop_times":0}],"id":1}
03777     *
03778     * @par JSON-RPC response example
03779     * {"id":1,"jsonrpc":"2.0","result":0}
03780     * \endenglish
03781     */
03782    int moveCircle2(const CircleParameters &param);
03783
03784    /**
03785     * @ingroup MotionControl
03786     * \chinese
03787     *
03788     *
03789     * @param name
03790     * @param type      \n
03791     *   1: toppra       \n
03792     *   2: cubic_spline(    ) \n
03793     *   3:  B
03794     * @param size
03795     * @return      AUBO_OK(0)
03796     *
03797     * @throws arcs::common_interface::AuboException
03798     *
03799     * @par Python
03800     * pathBufferAlloc(self: pyaubo_sdk.MotionControl, arg0: str, arg1: int,
03801     * arg2: int) -> int
03802     *
03803     * @par Lua
```

```
03804     * pathBufferAlloc(name: string, type: number, size: number) -> nil
03805     *
03806     * @par Lua
03807     * pathBufferAlloc("traje_name",5,100)
03808     *
03809     * @par JSON-RPC
03810     * {"jsonrpc":"2.0","method":"rob1.MotionControl.pathBufferAlloc","params":["rec",2,3],"id":1}
03811     *
03812     * @par JSON-RPC
03813     * {"id":1,"jsonrpc":"2.0","result":0}
03814     *
03815     * \endchinese
03816     * \english
03817     * Create a new path point buffer
03818     *
03819     * @param name Name of the path
03820     * @param type Type of the path \n
03821     *   1: toppra time-optimal path planning \n
03822     *   2: cubic_spline (recorded trajectory) \n
03823     *   3: Joint B-spline interpolation, at least three points
03824     * @param size Buffer size
03825     * @return Returns AUBO_OK(0) on success
03826     *
03827     * @throws arcs::common_interface::AuboException
03828     *
03829     * @par Python function prototype
03830     * pathBufferAlloc(self: pyaubo_sdk.MotionControl, arg0: str, arg1: int,
03831     * arg2: int) -> int
03832     *
03833     * @par Lua function prototype
03834     * pathBufferAlloc(name: string, type: number, size: number) -> nil
03835     *
03836     * @par Lua exmaple
03837     * pathBufferAlloc("traje_name",5,100)
03838     *
03839     * @par JSON-RPC request example
03840     * {"jsonrpc":"2.0","method":"rob1.MotionControl.pathBufferAlloc","params":["rec",2,3],"id":1}
03841     *
03842     * @par JSON-RPC response example
03843     * {"id":1,"jsonrpc":"2.0","result":0}
03844     *
03845     * \endenglish
03846     */
03847    int pathBufferAlloc(const std::string &name, int type, int size);
03848
03849    /**
03850     * @ingroup MotionControl
03851     * \chinese
03852     *
03853     *
03854     * @param name
03855     * @param waypoints
03856     * @return    0
03857     * -AUBO_INVL_ARGUMENT
03858     * -AUBO_BAD_STATE
03859     *
03860     * @throws arcs::common_interface::AuboException
03861     *
03862     * @par Python
03863     * pathBufferAppend(self: pyaubo_sdk.MotionControl, arg0: str, arg1:
03864     * List[List[float]]) -> int
03865     *
03866     * @par Lua
03867     * pathBufferAppend(name: string, waypoints: table) -> nil
03868     *
03869     * @par Lua
03870     * 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}})
03871     *
03872     * @par JSON-RPC
03873     * {"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}
03874     *
03875     * @par JSON-RPC
03876     * {"id":1,"jsonrpc":"2.0","result":0}
03877     * \endchinese
03878     * \english
03879     * Add waypoints to the path buffer
03880     *
03881     * @param name Name of the path buffer
03882     * @param waypoints Waypoints
03883     * @return Returns 0 on success; otherwise returns an error code
03884     * -AUBO_INVL_ARGUMENT
03885     * -AUBO_BAD_STATE
03886     *
03887     * @throws arcs::common_interface::AuboException
03888     *
```

```
03889     * @par Python function prototype
03890     * pathBufferAppend(self: pyaubo_sdk.MotionControl, arg0: str, arg1:
03891     * List[List[float]]) -> int
03892     *
03893     * @par Lua function prototype
03894     * pathBufferAppend(name: string, waypoints: table) -> nil
03895     *
03896     * @par Lua example
03897     * 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}})
03898     *
03899     * @par JSON-RPC request example
03900     * {"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}
03901     *
03902     * @par JSON-RPC response example
03903     * {"id":1,"jsonrpc":"2.0","result":0}
03904     * \endenglish
03905     */
03906    int pathBufferAppend(const std::string &name,
03907                         const std::vector<std::vector<double» &waypoints);
03908
03909    /**
03910     * @ingroup MotionControl
03911     * \chinese
03912     *
03913     *
03914     * @param name   pathBufferAlloc
03915     * @param a      ,          ,   rad/s^2
03916     * @param v      ,          ,   rad/s
03917     * @param t    \n
03918     * pathBufferAlloc
03919     *   pathBufferAlloc     :\n
03920     *    1 ,       \n
03921     *    2 ,       \n
03922     *    3 :t      \n
03923     *       t=0,             (    )\n
03924     *       t!=0, t          *     (points *
03925     * interval,interval >= 0.01)
03926     * @return    0
03927     * -AUBO_INVL_ARGUMENT
03928     * -AUBO_BAD_STATE
03929     *
03930     * @throws arcs::common_interface::AuboException
03931     *
03932     * @par Python
03933     * pathBufferEval(self: pyaubo_sdk.MotionControl, arg0: str, arg1:
03934     * List[float], arg2: List[float], arg3: float) -> int
03935     *
03936     * @par Lua
03937     * pathBufferEval(name: string, a: table, v: table, t: number) -> nil
03938     *
03939     * @par Lua
03940     * pathBufferEval("traje_name",{1,1,1,1,1,1},{1,1,1,1,1,1},0.02)
03941     *
03942     * @par JSON-RPC
03943     * {"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}
03944     *
03945     * @par JSON-RPC
03946     * {"id":1,"jsonrpc":"2.0","result":0}
03947     * \endchinese
03948     * \english
03949     * Perform computation and optimization (time-consuming operations). If the
03950     * input parameters are the same, the calculation will not be repeated.
03951     *
03952     * @param name Name of the path point buffer created by pathBufferAlloc
03953     * @param a Joint acceleration limits, must match the robot DOF, unit:
03954     * rad/s^2
03955     * @param v Joint velocity limits, must match the robot DOF, unit: rad/s
03956     * @param t Time \n
03957     * When allocating memory with pathBufferAlloc, the type must be specified.
03958     * According to the type in pathBufferAlloc:\n
03959     * Type 1: t means motion duration\n
03960     * Type 2: t means sampling interval\n
03961     * Type 3: t means motion duration\n
03962     *     If t=0, the internal default time is used (recommended)\n
03963     *     If t!=0, t should be set to number_of_points *
03964     * interval_between_points (interval >= 0.01)
03965     * @return Returns 0 on success; otherwise returns an error code
03966     * -AUBO_INVL_ARGUMENT
03967     * -AUBO_BAD_STATE
03968     *
03969     * @throws arcs::common_interface::AuboException
03970     *
03971     * @par Python function prototype
03972     * pathBufferEval(self: pyaubo_sdk.MotionControl, arg0: str, arg1:
03973     * List[float], arg2: List[float], arg3: float) -> int
```

```
03974     *
03975     * @par Lua function prototype
03976     * pathBufferEval(name: string, a: table, v: table, t: number) -> nil
03977     *
03978     * @par Lua example
03979     * pathBufferEval("traje_name",{1,1,1,1,1,1},{1,1,1,1,1,1},0.02)
03980     *
03981     * @par JSON-RPC request example
03982     * {"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}
03983     *
03984     * @par JSON-RPC response example
03985     * {"id":1,"jsonrpc":"2.0","result":0}
03986     * \endenglish
03987     */
03988    int pathBufferEval(const std::string &name, const std::vector<double> &a,
03989                       const std::vector<double> &v, double t);
03990
03991    /**
03992     * @ingroup MotionControl
03993     * \chinese
03994     *     buffer
03995     *
03996     * buffer        : \n
03997     * 1 buffer          \n
03998     * 2   buffer      buffer    \n
03999     * 3     pathBufferEval
04000     *
04001     * @param name buffer
04002     * @return     true;     false
04003     *
04004     * @throws arcs::common_interface::AuboException
04005     *
04006     * @par Python
04007     * pathBufferValid(self: pyaubo_sdk.MotionControl, arg0: str) -> bool
04008     *
04009     * @par Lua
04010     * pathBufferValid(name: string) -> boolean
04011     *
04012     * @par Lua
04013     * buffer_status = pathBufferValid("traje_name")
04014     *
04015     * @par JSON-RPC
04016     * {"jsonrpc":"2.0","method":"rob1.MotionControl.pathBufferValid","params":["rec"],"id":1}
04017     *
04018     * @par JSON-RPC
04019     * {"id":1,"jsonrpc":"2.0","result":false}
04020     * \endchinese
04021     * \english
04022     * Whether the buffer with the specified name is valid
04023     *
04024     * The buffer must meet three conditions to be valid: \n
04025     * 1. The buffer exists and memory has been allocated \n
04026     * 2. The number of points passed into the buffer must be greater than or
04027     * equal to the buffer size \n
04028     * 3. pathBufferEval must be executed once
04029     *
04030     * @param name Name of the buffer
04031     * @return Returns true if valid; false otherwise
04032     *
04033     * @throws arcs::common_interface::AuboException
04034     *
04035     * @par Python function prototype
04036     * pathBufferValid(self: pyaubo_sdk.MotionControl, arg0: str) -> bool
04037     *
04038     * @par Lua function prototype
04039     * pathBufferValid(name: string) -> boolean
04040     *
04041     * @par Lua example
04042     * buffer_status = pathBufferValid("traje_name")
04043     *
04044     * @par JSON-RPC request example
04045     * {"jsonrpc":"2.0","method":"rob1.MotionControl.pathBufferValid","params":["rec"],"id":1}
04046     *
04047     * @par JSON-RPC response example
04048     * {"id":1,"jsonrpc":"2.0","result":false}
04049     * \endenglish
04050     */
04051    bool pathBufferValid(const std::string &name);
04052
04053    /**
04054     * @ingroup MotionControl
04055     * \chinese
04056     *
04057     *
04058     * @param name
04059     * @return     0
04060     * -AUBO_INVL_ARGUMENT
```

```
04061     * -AUBO_BAD_STATE
04062     *
04063     * @throws arcs::common_interface::AuboException
04064     *
04065     * @par Python
04066     * pathBufferFree(self: pyaubo_sdk.MotionControl, arg0: str) -> int
04067     *
04068     * @par Lua
04069     * pathBufferFree(name: string) -> nil
04070     *
04071     * @par Lua
04072     * pathBufferFree("traje_name")
04073     *
04074     * @par JSON-RPC
04075     * {"jsonrpc":"2.0","method":"rob1.MotionControl.pathBufferFree","params":["rec"],"id":1}
04076     *
04077     * @par JSON-RPC
04078     * {"id":1,"jsonrpc":"2.0","result":-5}
04079     *
04080     * \endchinese
04081     * \english
04082     * Release path buffer
04083     *
04084     * @param name Name of the path buffer
04085     * @return Returns 0 on success; otherwise returns an error code
04086     * -AUBO_INVL_ARGUMENT
04087     * -AUBO_BAD_STATE
04088     *
04089     * @throws arcs::common_interface::AuboException
04090     *
04091     * @par Python function prototype
04092     * pathBufferFree(self: pyaubo_sdk.MotionControl, arg0: str) -> int
04093     *
04094     * @par Lua function prototype
04095     * pathBufferFree(name: string) -> nil
04096     *
04097     * @par Lua example
04098     * pathBufferFree("traje_name")
04099     *
04100     * @par JSON-RPC request example
04101     * {"jsonrpc":"2.0","method":"rob1.MotionControl.pathBufferFree","params":["rec"],"id":1}
04102     *
04103     * @par JSON-RPC response example
04104     * {"id":1,"jsonrpc":"2.0","result":-5}
04105     *
04106     * \endenglish
04107     */
04108    int pathBufferFree(const std::string &name);
04109
04110    /**
04111     * @ingroup MotionControl
04112     * \chinese
04113     *
04114     *
04115     * @brief pathBufferFilter
04116     *
04117     * @param name
04118     * @param order      (  2)
04119     * @param fd             (  3-20)
04120     * @param fs         (  20-500)
04121     *
04122     * @return    0
04123     *
04124     * @throws arcs::common_interface::AuboException
04125     *
04126     * @par Python
04127     * pathBufferFilter(self: pyaubo_sdk.MotionControl, arg0: str, arg1: int,
04128     * arg2: float, arg3: float) -> int
04129     *
04130     * @par Lua
04131     * pathBufferFilter(name: string, order: number, fd: number, fs:number) ->
04132     * nil
04133     *
04134     * @par Lua
04135     * pathBufferFilter("traje_name",2,5,125)
04136     *
04137     * \endchinese
04138     * \english
04139     * Joint space path filter
04140     *
04141     * @brief pathBufferFilter
04142     *
04143     * @param name Name of the path buffer
04144     * @param order Filter order (usually 2)
04145     * @param fd Cutoff frequency, the smaller the smoother but with more delay
04146     * (usually 3-20)
04147     * @param fs Sampling frequency of discrete data (usually 20-500)
```

```
04148     *
04149     * @return Returns 0 on success
04150     *
04151     * @throws arcs::common_interface::AuboException
04152     *
04153     * @par Python function prototype
04154     * pathBufferFilter(self: pyaubo_sdk.MotionControl, arg0: str, arg1: int,
04155     * arg2: float, arg3: float) -> int
04156     *
04157     * @par Lua function prototype
04158     * pathBufferFilter(name: string, order: number, fd: number, fs:number) ->
04159     * nil
04160     *
04161     * @par Lua example
04162     * pathBufferFilter("traje_name",2,5,125)
04163     *
04164     * \endenglish
04165     */
04166    int pathBufferFilter(const std::string &name, int order, double fd,
04167                         double fs);
04168
04169    /**
04170     * @ingroup MotionControl
04171     * \chinese
04172     *
04173     *
04174     * @return
04175     *
04176     * @throws arcs::common_interface::AuboException
04177     *
04178     * @par Python
04179     * pathBufferList(self: pyaubo_sdk.MotionControl) -> List[str]
04180     *
04181     * @par Lua
04182     * pathBufferList() -> table
04183     *
04184     * @par Lua
04185     * Buffer_table = pathBufferList()
04186     *
04187     * @par JSON-RPC
04188     * {"jsonrpc":"2.0","method":"rob1.MotionControl.pathBufferList","params":[],"id":1}
04189     *
04190     * @par JSON-RPC
04191     * {"id":1,"jsonrpc":"2.0","result":[]}
04192     *
04193     * \endchinese
04194     * \english
04195     * List all cached path names
04196     *
04197     * @return Returns all cached path names
04198     *
04199     * @throws arcs::common_interface::AuboException
04200     *
04201     * @par Python function prototype
04202     * pathBufferList(self: pyaubo_sdk.MotionControl) -> List[str]
04203     *
04204     * @par Lua function prototype
04205     * pathBufferList() -> table
04206     *
04207     * @par Lua example
04208     * Buffer_table = pathBufferList()
04209     *
04210     * @par JSON-RPC request example
04211     * {"jsonrpc":"2.0","method":"rob1.MotionControl.pathBufferList","params":[],"id":1}
04212     *
04213     * @par JSON-RPC response example
04214     * {"id":1,"jsonrpc":"2.0","result":[]}
04215     *
04216     * \endenglish
04217     */
04218    std::vector<std::string> pathBufferList();
04219
04220    /**
04221     * @ingroup MotionControl
04222     * \chinese
04223     *
04224     *
04225     * @param name
04226     * @return    0
04227     * -AUBO_INVL_ARGUMENT
04228     * -AUBO_BAD_STATE
04229     *
04230     * @throws arcs::common_interface::AuboException
04231     *
04232     * @par Python
04233     * movePathBuffer(self: pyaubo_sdk.MotionControl, arg0: str) -> int
04234     *
```

```
04235     * @par Lua
04236     * movePathBuffer(name: string) -> nil
04237     *
04238     * @par Lua
04239     * movePathBuffer("traje_name")
04240     *
04241     * @par JSON-RPC
04242     * {"jsonrpc":"2.0","method":"rob1.MotionControl.movePathBuffer","params":["rec"],"id":1}
04243     *
04244     * @par JSON-RPC
04245     * {"id":1,"jsonrpc":"2.0","result":0}
04246     * \endchinese
04247     * \english
04248     * Execute the cached path
04249     *
04250     * @param name Name of the cached path
04251     * @return Returns 0 on success; otherwise returns an error code
04252     * -AUBO_INVL_ARGUMENT
04253     * -AUBO_BAD_STATE
04254     *
04255     * @throws arcs::common_interface::AuboException
04256     *
04257     * @par Python function prototype
04258     * movePathBuffer(self: pyaubo_sdk.MotionControl, arg0: str) -> int
04259     *
04260     * @par Lua function prototype
04261     * movePathBuffer(name: string) -> nil
04262     *
04263     * @par Lua example
04264     * movePathBuffer("traje_name")
04265     *
04266     * @par JSON-RPC request example
04267     * {"jsonrpc":"2.0","method":"rob1.MotionControl.movePathBuffer","params":["rec"],"id":1}
04268     *
04269     * @par JSON-RPC response example
04270     * {"id":1,"jsonrpc":"2.0","result":0}
04271     * \endenglish
04272     */
04273    int movePathBuffer(const std::string &name);
04274
04275    /**
04276     * @ingroup MotionControl
04277     * \chinese
04278     *
04279     *
04280     * @param pose       (       ,    :
04281     *                          )
04282     * p1:             ,
04283     * p2:             ,
04284     * p3:             ,
04285     * @param v
04286     * @param a
04287     * @param main_pipe_radius
04288     * @param sub_pipe_radius
04289     * @param normal_distance
04290     * @param normal_alpha
04291     *
04292     * @return    0
04293     * AUBO_BUSY
04294     * AUBO_BAD_STATE
04295     * -AUBO_INVL_ARGUMENT
04296     * -AUBO_BAD_STATE
04297     *
04298     * @throws arcs::common_interface::AuboException
04299     *
04300     * @par Python
04301     * moveIntersection(self: pyaubo_sdk.MotionControl, arg0: List[float], arg1:
04302     * float, arg2: float, arg3: float, arg4: float, arg5: float, arg6: float)
04303     * -> int
04304     *
04305     * @par Lua
04306     * moveIntersection(poses: table, a: number, v: number,
04307     * main_pipe_radius: number, sub_pipe_radius: number, normal_distance:
04308     * number, normal_alpha: number) -> nil \endchinese \english Intersection
04309     * interface
04310     *
04311     * @param poses Consists of three taught poses (first, move to the starting
04312     * point. The starting point requirement: the intersection of the plane
04313     * passing through the main cylinder axis and parallel to the sub-cylinder
04314     * axis with the sub-cylinder at the bottom) p1: Intersection of the plane
04315     * passing through the sub-cylinder axis and parallel to the main cylinder
04316     * axis with the sub-cylinder at the left top p2: Intersection of the plane
04317     * passing through the sub-cylinder axis and parallel to the main cylinder
04318     * axis with the main cylinder at the left bottom p3: Intersection of the
04319     * plane passing through the sub-cylinder axis and parallel to the main
04320     * cylinder axis with the main cylinder at the right bottom
04321     * @param v Velocity
```

```
04322     * @param a Acceleration
04323     * @param main_pipe_radius Main cylinder radius
04324     * @param sub_pipe_radius  Sub cylinder radius
04325     * @param normal_distance  Distance between the two cylinder axes
04326     * @param normal_alpha     Angle between the two cylinder axes
04327     *
04328     * @return Returns 0 on success; otherwise returns an error code
04329     * AUBO_BUSY
04330     * AUBO_BAD_STATE
04331     * -AUBO_INVL_ARGUMENT
04332     * -AUBO_BAD_STATE
04333     *
04334     * @throws arcs::common_interface::AuboException
04335     *
04336     * @par Python function prototype
04337     * moveIntersection(self: pyaubo_sdk.MotionControl, arg0: List[float], arg1:
04338     * float, arg2: float, arg3: float, arg4: float, arg5: float, arg6: float)
04339     * -> int
04340     *
04341     * @par Lua function prototype
04342     * moveIntersection(poses: table, a: number, v: number,
04343     * main_pipe_radius: number, sub_pipe_radius: number, normal_distance:
04344     * number, normal_alpha: number) -> nil \endenglish
04345     */
04346    int moveIntersection(const std::vector<std::vector<double» &poses,
04347                         double a, double v, double main_pipe_radius,
04348                         double sub_pipe_radius, double normal_distance,
04349                         double normal_alpha);
04350    /**
04351     * @ingroup MotionControl
04352     * \chinese
04353     *
04354     *
04355     * @param acc       : rad/s^2
04356     * @return    0
04357     * AUBO_BUSY
04358     * AUBO_BAD_STATE
04359     * -AUBO_INVL_ARGUMENT
04360     * -AUBO_BAD_STATE
04361     *
04362     * @throws arcs::common_interface::AuboException
04363     *
04364     * @par Python
04365     * stopJoint(self: pyaubo_sdk.MotionControl, arg0: float) -> int
04366     *
04367     * @par Lua
04368     * stopJoint(acc: number) -> nil
04369     *
04370     * @par Lua
04371     * stopJoint(2)
04372     *
04373     * @par JSON-RPC
04374     * {"jsonrpc":"2.0","method":"rob1.MotionControl.stopJoint","params":[31],"id":1}
04375     *
04376     * @par JSON-RPC
04377     * {"id":1,"jsonrpc":"2.0","result":0}
04378     *
04379     * \endchinese
04380     * \english
04381     * Stop motion in joint space
04382     *
04383     * @param acc Joint acceleration, unit: rad/s^2
04384     * @return Returns 0 on success; otherwise returns an error code
04385     * AUBO_BUSY
04386     * AUBO_BAD_STATE
04387     * -AUBO_INVL_ARGUMENT
04388     * -AUBO_BAD_STATE
04389     *
04390     * @throws arcs::common_interface::AuboException
04391     *
04392     * @par Python function prototype
04393     * stopJoint(self: pyaubo_sdk.MotionControl, arg0: float) -> int
04394     *
04395     * @par Lua function prototype
04396     * stopJoint(acc: number) -> nil
04397     *
04398     * @par Lua example
04399     * stopJoint(2)
04400     *
04401     * @par JSON-RPC request example
04402     * {"jsonrpc":"2.0","method":"rob1.MotionControl.stopJoint","params":[31],"id":1}
04403     *
04404     * @par JSON-RPC response example
04405     * {"id":1,"jsonrpc":"2.0","result":0}
04406     *
04407     * \endenglish
04408     */
```

```
04409   int stopJoint(double acc);
04410
04411   /**
04412    * @ingroup MotionControl
04413    * \chinese
04414    *       (         , resumeSpeedJoint               )
04415    *
04416    * @param acc       : rad/s^2
04417    * @return    0
04418    * AUBO_BUSY
04419    * AUBO_BAD_STATE
04420    * -AUBO_INVL_ARGUMENT
04421    * -AUBO_BAD_STATE
04422    *
04423    * @throws arcs::common_interface::AuboException
04424    *
04425    * @par Python
04426    * resumeStopJoint(self: pyaubo_sdk.MotionControl, arg0: float) -> int
04427    *
04428    * @par Lua
04429    * resumeStopJoint(acc: number) -> nil
04430    *
04431    * @par Lua
04432    * resumeStopJoint(31)
04433    *
04434    * @par JSON-RPC
04435    * {"jsonrpc":"2.0","method":"rob1.MotionControl.resumeStopJoint","params":[31],"id":1}
04436    *
04437    * @par JSON-RPC
04438    * {"id":1,"jsonrpc":"2.0","result":-1}
04439    *
04440    * \endchinese
04441    * \english
04442    * Stop motion in joint space (used after moving to a safe position via
04443    * resumeSpeedJoint following a collision during process execution)
04444    *
04445    * @param acc Joint acceleration, unit: rad/s^2
04446    * @return Returns 0 on success; otherwise returns an error code
04447    * AUBO_BUSY
04448    * AUBO_BAD_STATE
04449    * -AUBO_INVL_ARGUMENT
04450    * -AUBO_BAD_STATE
04451    *
04452    * @throws arcs::common_interface::AuboException
04453    *
04454    * @par Python function prototype
04455    * resumeStopJoint(self: pyaubo_sdk.MotionControl, arg0: float) -> int
04456    *
04457    * @par Lua function prototype
04458    * resumeStopJoint(acc: number) -> nil
04459    *
04460    * @par Lua example
04461    * resumeStopJoint(31)
04462    *
04463    * @par JSON-RPC request example
04464    * {"jsonrpc":"2.0","method":"rob1.MotionControl.resumeStopJoint","params":[31],"id":1}
04465    *
04466    * @par JSON-RPC response example
04467    * {"id":1,"jsonrpc":"2.0","result":-1}
04468    *
04469    * \endenglish
04470    */
04471   int resumeStopJoint(double acc);
04472
04473   /**
04474    * @ingroup MotionControl
04475    * \chinese
04476    *   moveLine/moveCircle
04477    *
04478    * @param acc     ,   : m/s^2
04479    * @param acc_rot
04480    * @return    0
04481    * AUBO_BUSY
04482    * AUBO_BAD_STATE
04483    * -AUBO_INVL_ARGUMENT
04484    * -AUBO_BAD_STATE
04485    *
04486    * @throws arcs::common_interface::AuboException
04487    *
04488    * @par Python
04489    * stopLine(self: pyaubo_sdk.MotionControl, arg0: float, arg1: float) -> int
04490    *
04491    * @par Lua
04492    * stopLine(acc: number, acc_rot: number) -> nil
04493    *
04494    * @par Lua
04495    * stopLine(10,10)
```

```
04496     *
04497     * @par JSON-RPC
04498     * {"jsonrpc":"2.0","method":"rob1.MotionControl.stopLine","params":[10,10],"id":1}
04499     *
04500     * @par JSON-RPC
04501     * {"id":1,"jsonrpc":"2.0","result":0}
04502     *
04503     * \endchinese
04504     * \english
04505     * Stop motions in Cartesian space such as moveLine/moveCircle.
04506     *
04507     * @param acc Tool acceleration, unit: m/s^2
04508     * @param acc_rot
04509     * @return Returns 0 on success; otherwise returns an error code
04510     * AUBO_BUSY
04511     * AUBO_BAD_STATE
04512     * -AUBO_INVL_ARGUMENT
04513     * -AUBO_BAD_STATE
04514     *
04515     * @throws arcs::common_interface::AuboException
04516     *
04517     * @par Python function prototype
04518     * stopLine(self: pyaubo_sdk.MotionControl, arg0: float, arg1: float) -> int
04519     *
04520     * @par Lua function prototype
04521     * stopLine(acc: number, acc_rot: number) -> nil
04522     *
04523     * @par Lua example
04524     * stopLine(10,10)
04525     *
04526     * @par JSON-RPC request example
04527     * {"jsonrpc":"2.0","method":"rob1.MotionControl.stopLine","params":[10,10],"id":1}
04528     *
04529     * @par JSON-RPC response example
04530     * {"id":1,"jsonrpc":"2.0","result":0}
04531     *
04532     * \endenglish
04533     */
04534    int stopLine(double acc, double acc_rot);
04535
04536    /**
04537     * @ingroup MotionControl
04538     * \chinese
04539     *       (         , resumeSpeedLine              )
04540     *
04541     * @param acc     ,   : m/s^2
04542     * @param acc_rot       : rad/s^2
04543     * @return    0
04544     * AUBO_BUSY
04545     * AUBO_BAD_STATE
04546     * -AUBO_INVL_ARGUMENT
04547     * -AUBO_BAD_STATE
04548     *
04549     * @throws arcs::common_interface::AuboException
04550     *
04551     * @par Python
04552     * resumeStopLine(self: pyaubo_sdk.MotionControl, arg0: float, arg1: float)
04553     * -> int
04554     *
04555     * @par Lua
04556     * resumeStopLine(acc: number, acc_rot: number) -> nil
04557     *
04558     * @par Lua
04559     * resumeStopLine(10,10)
04560     *
04561     * @par JSON-RPC
04562     * {"jsonrpc":"2.0","method":"rob1.MotionControl.resumeStopLine","params":[10,10],"id":1}
04563     *
04564     * @par JSON-RPC
04565     * {"id":1,"jsonrpc":"2.0","result":-1}
04566     *
04567     * \endchinese
04568     * \english
04569     * Stop motion in Cartesian space (used after moving to a safe position via
04570     * resumeSpeedLine following a collision during process execution)
04571     *
04572     * @param acc Position acceleration, unit: m/s^2
04573     * @param acc_rot Orientation acceleration, unit: rad/s^2
04574     * @return Returns 0 on success; otherwise returns an error code
04575     * AUBO_BUSY
04576     * AUBO_BAD_STATE
04577     * -AUBO_INVL_ARGUMENT
04578     * -AUBO_BAD_STATE
04579     *
04580     * @throws arcs::common_interface::AuboException
04581     *
04582     * @par Python function prototype
```

```
04583  * resumeStopLine(self: pyaubo_sdk.MotionControl, arg0: float, arg1: float)
04584  * -> int
04585  *
04586  * @par Lua function prototype
04587  * resumeStopLine(acc: number, acc_rot: number) -> nil
04588  *
04589  * @par Lua example
04590  * resumeStopLine(10,10)
04591  *
04592  * @par JSON-RPC request example
04593  * {"jsonrpc":"2.0","method":"rob1.MotionControl.resumeStopLine","params":[10,10],"id":1}
04594  *
04595  * @par JSON-RPC response example
04596  * {"id":1,"jsonrpc":"2.0","result":-1}
04597  *
04598  * \endenglish
04599  */
04600 int resumeStopLine(double acc, double acc_rot);
04601
04602 /**
04603  * @ingroup MotionControl
04604  * \chinese
04605  *
04606  * \warning  weaveStart  RuntimeMachine  start
04607  *
04608  *  weaveStart  weaveEnd
04609  * moveLine/moveCircle/moveProcess  params
04610  *
04611  * @param params Json
04612  *
04613  * @par  v0.29
04614  * - "type": <string>  ——  "SINE" |
04615  * "SPIRAL" | "TRIANGLE" | "TRAPEZOIDAL"
04616  * - "step": <number>  ——
04617  *  m
04618  * - "amplitude": [<number>,<number>]  ——  m
04619  * - "hold_distance": [<number>,<number>]  ——
04620  *  m
04621  * - "hold_time": [<number>,<number>]  ——
04622  *  s
04623  * - "angle": <number>  ——
04624  *  rad
04625  * - "direction": <integer>  ——
04626  *  0=  1=
04627  * - "movep_uniform_vel": <bool>  —— moveProcess
04628  *
04629  *
04630  * @par  v0.31
04631  * -  v0.29
04632  *
04633  * @par  v0.32  /
04634  * - "angle": [<number>,<number>]  ——  rad
04635  * - "frequency": <number>  ——  Hz  [0.1, 5]
04636  * - "ori_weave_range": [<number>,<number>,<number>]  ——
04637  * [x,y,z]  rad
04638  * - "ori_weave_frequency": [<number>,<number>,<number>] ——
04639  * [x,y,z]  m^-1  [0, 1]
04640  * - "adjust_cycle_num": <integer>  ——
04641  *  /
04642  * - "azimuth": <number>  ——  rad
04643  * - "ridge_height": <number>  ——  m
04644  *
04645  * \warning  **frequency = vel / step**  vel  vel
04646  *  m · s^-1
04647  * \warning **hold_time**  **SINE**
04648  * \warning "hold_distance"/"hold_time"
04649  *
04650  * \note
04651  * -  "amplitude"/"hold_distance"/"hold_time"
04652  * [left,right] "ori_weave_*"  [x,y,z]
04653  * -  rad
04654  *
04655  * @return  0;
04656  * - AUBO_BUSY
04657  * - AUBO_BAD_STATE
04658  * - -AUBO_INVL_ARGUMENT
04659  * - -AUBO_BAD_STATE
04660  *
04661  * @throws arcs::common_interface::AuboException
04662  *
04663  * @par  v0.29/v0.31
04664  * \code{.json}
04665  * params = {
04666  *  "type": "SINE",
04667  *  "step": 0.005,
04668  *  "amplitude": [0.01, 0.01],
04669  *  "hold_distance": [0.001, 0.001],
```

```
04670     *     "hold_time": [0, 0],
04671     *     "angle": 0,
04672     *     "direction": 0,
04673     *     "movep_uniform_vel": false
04674     * }
04675     * \endcode
04676     *
04677     * @par    v0.32
04678     * \code{.json}
04679     * params = {
04680     *     "type" : "SINE",  // "SINE" "SPIRAL" "TRIANGLE" "SAWTOOTH" "CRESCENT"
04681     *     "step" : 0.0,     //    frequency = vel / step vel: m · s^-1
04682     *     "frequency" : 0.0, // [0.1, 5] Hz
04683     *     "amplitude" : [0.01, 0.01],               // m
04684     *     "hold_distance" : [0.001, 0.001],          // m
04685     *     "hold_time" : [0, 0],                     // s   SINE
04686     *     "angle" : [0, 0],                        // rad
04687     *     "direction" : 0,
04688     *     "ori_weave_range" : [0.001, 0.001, 0.001],          // rad
04689     *     "ori_weave_frequency": [0.001, 0.001, 0.001],       // m^-1
04690     *     "adjust_cycle_num": 0,
04691     *     "azimuth" : 0,                           // rad
04692     *     "ridge_height" : 0                        // m
04693     * }
04694     * \endcode
04695     *
04696     * @par Python
04697     * \code{.py}
04698     * weaveStart(self: pyaubo_sdk.MotionControl, arg0: str) -> int
04699     * \endcode
04700     *
04701     * @par Python
04702     * \code{.py}
04703     * robot_name = rpc_cli.getRobotNames()[0]
04704     * robot_interface = rpc_cli.getRobotInterface(robot_name)
04705     * robot_interface.getMotionControl().weaveStart(params)
04706     * \endcode
04707     *
04708     * @par Lua
04709     * \code{.lua}
04710     * weaveStart(params: string) -> nil
04711     * \endcode
04712     *
04713     * @par Lua    v0.29/0.31
04714     * \code{.lua}
04715     *
weaveStart("{\"type\":\"SINE\",\"step\":0.005,\"amplitude\":[0.01,0.01],\"hold_distance\":[0.001,0.001],\"hold_time\":[0,0],\"angle\":0,\"direc
04716     * \endcode
04717     *
04718     * @par Lua    v0.32
04719     * \code{.lua}
04720     *
weaveStart("{\"type\":\"SINE\",\"step\":0.0,\"frequency\":2.0,\"amplitude\":[0.01,0.01],\"hold_distance\":[0.001,0.001],\"hold_time\":[0,0],\"a
04721     * \endcode
04722     *
04723     * @par JSON-RPC     v0.29/0.31
04724     * \code{.json}
04725     *
{"jsonrpc":"2.0","method":"rob1.MotionControl.weaveStart","params":["{\"type\":\"SINE\",\"step\":0.005,\"amplitude\":[0.01,0.01],\"hold_di
04726     * \endcode
04727     *
04728     * @par JSON-RPC     v0.32
04729     * \code{.json}
04730     *
{"jsonrpc":"2.0","method":"rob1.MotionControl.weaveStart","params":["{\"type\":\"SINE\",\"step\":0.0,\"frequency\":2.0,\"amplitude\":[0.01
04731     * \endcode
04732     *
04733     * @par JSON-RPC
04734     * \code{.json}
04735     * {"id":1,"jsonrpc":"2.0","result":0}
04736     * \endcode
04737     * \endchinese
04738     *
04739     * \english
04740     * Start weaving: between weaveStart and weaveEnd,
04741     * moveLine/moveCircle/moveProcess follows params.
04742     *
04743     * \warning Start RuntimeMachine before calling weaveStart.
04744     *
04745     * @param params Json string.
04746     *
04747     * @par Since v0.29
04748     * - "type": <string>                        — Waveform: "SINE" | "SPIRAL"
04749     * | "TRIANGLE" | "TRAPEZOIDAL" (case-sensitive).
04750     * - "step": <number>                          — Arc-length sampling step,
04751     * m.
04752     * - "amplitude": [<number>,<number>]          — Left/right weave amplitude,
```

```
04753     * m.
04754     * - "hold_distance": [<number>,<number>]      — Dwell distance at peak
04755     * amplitude along path, m.
04756     * - "hold_time": [<number>,<number>]          — Dwell time at peak
04757     * amplitude, s.
04758     * - "angle": <number>                         — Angle to normal plane, rad.
04759     * - "direction": <integer>                    — Initial direction: 0=above
04760     * path, 1=below path.
04761     * - "movep_uniform_vel": <bool>               — Uniform-velocity planning
04762     * (boolean value).
04763     *
04764     * @par Since v0.31
04765     * - Same as v0.29.
04766     *
04767     * @par New/changed in v0.32
04768     * - "frequency": <number>                     — Weave frequency, Hz (range
04769     * [0.1, 5]).
04770     * - "ori_weave_range": [<number>,<number>,<number>]     — Attitude weave
04771     * range [x,y,z], rad; often used with Archimedean spiral for force-guided
04772     * hole finding.
04773     * - "ori_weave_frequency": [<number>,<number>,<number>] — Attitude weave
04774     * frequency [x,y,z], m^-1 (per unit arc; range [0, 1]).
04775     * - "adjust_cycle_num": <integer>              — Number of transition
04776     * cycles.
04777     * - "azimuth": <number>                       — Waveform azimuth, rad.
04778     * - "ridge_height": <number>                  — Center raise height, m.
04779     *
04780     * \warning **hold_time** is supported for **SINE** only and may be
04781     * asymmetric. \warning You may relate **frequency = vel / step** when an
04782     * external path speed vel (m · s^-1) is available.
04783     *
04784     * \note
04785     * - Two-element arrays are [left,right]; three-element arrays are attitude
04786     * [x,y,z].
04787     * - All angles are in radians (rad).
04788     *
04789     * @return Returns 0 on success; otherwise returns an error code:
04790     * - AUBO_BUSY
04791     * - AUBO_BAD_STATE
04792     * - -AUBO_INVL_ARGUMENT
04793     * - -AUBO_BAD_STATE
04794     *
04795     * @throws arcs::common_interface::AuboException
04796     *
04797     * @par Example v0.29/v0.31
04798     * \code{.json}
04799     * params = {
04800     *     "type": "SINE",
04801     *     "step": 0.005,
04802     *     "amplitude": [0.01, 0.01],
04803     *     "hold_distance": [0.001, 0.001],
04804     *     "hold_time": [0, 0],
04805     *     "angle": 0,
04806     *     "direction": 0,
04807     *     "movep_uniform_vel": false
04808     * }
04809     * \endcode
04810     *
04811     * @par Example v0.32
04812     * \code{.json}
04813     * params = {
04814     *     "type" : "SINE",  // "SINE" "SPIRAL" "TRIANGLE" "TRAPEZOIDAL"
04815     *     "step" : 0.0,     // frequency = vel / step (vel: m · s^-1)
04816     *     "frequency" : 0.0, // [0.1, 5] Hz
04817     *     "amplitude" : [0.01, 0.01],          // m
04818     *     "hold_distance" : [0.001, 0.001],    // m
04819     *     "hold_time" : [0, 0],                // s (SINE only)
04820     *     "angle" : 0,                         // rad
04821     *     "direction" : 0,
04822     *     "ori_weave_range" : [0.001, 0.001, 0.001],      // rad
04823     *     "ori_weave_frequency": [0.001, 0.001, 0.001],   // m^-1
04824     *     "adjust_cycle_num": 0,
04825     *     "azimuth" : 0,                       // rad
04826     *     "ridge_height" : 0                   // m
04827     * }
04828     * \endcode
04829     *
04830     * @par Python function prototype
04831     * \code{.py}
04832     * weaveStart(self: pyaubo_sdk.MotionControl, arg0: str) -> int
04833     * \endcode
04834     *
04835     * @par Python example
04836     * \code{.py}
04837     * robot_name = rpc_cli.getRobotNames()[0]
04838     * robot_interface = rpc_cli.getRobotInterface(robot_name)
04839     * robot_interface.getMotionControl().weaveStart(params)
```

```
04840     * \endcode
04841     *
04842     * @par Lua function prototype
04843     * \code{.lua}
04844     * weaveStart(params: string) -> nil
04845     * \endcode
04846     *
04847     * @par Lua example (v0.29/0.31)
04848     * \code{.lua}
04849     *
      weaveStart("{\"type\":\"SINE\",\"step\":0.005,\"amplitude\":[0.01,0.01],\"hold_distance\":[0.001,0.001],\"hold_time\":[0,0],\"angle\":0,\"direc
04850     * \endcode
04851     *
04852     * @par Lua example (v0.32)
04853     * \code{.lua}
04854     *
      weaveStart("{\"type\":\"SINE\",\"step\":0.0,\"frequency\":2.0,\"amplitude\":[0.01,0.01],\"hold_distance\":[0.001,0.001],\"hold_time\":[0,0],\"a
04855     * \endcode
04856     *
04857     * @par JSON-RPC request example (v0.29/0.31)
04858     * \code{.json}
04859     *
      {"jsonrpc":"2.0","method":"rob1.MotionControl.weaveStart","params":["{\"type\":\"SINE\",\"step\":0.005,\"amplitude\":[0.01,0.01],\"hold_di
04860     * \endcode
04861     *
04862     * @par JSON-RPC request example (v0.32)
04863     * \code{.json}
04864     *
      {"jsonrpc":"2.0","method":"rob1.MotionControl.weaveStart","params":["{\"type\":\"SINE\",\"step\":0.0,\"frequency\":2.0,\"amplitude\":[0.01
04865     * \endcode
04866     *
04867     * @par JSON-RPC response example
04868     * \code{.json}
04869     * {"id":1,"jsonrpc":"2.0","result":0}
04870     * \endcode
04871     * \endenglish
04872     */
04873    int weaveStart(const std::string &params);
04874
04875    /**
04876     * @ingroup MotionControl
04877     * \chinese
04878     *
04879     *
04880     * @return    0
04881     * AUBO_BUSY
04882     * AUBO_BAD_STATE
04883     * -AUBO_BAD_STATE
04884     *
04885     * @throws arcs::common_interface::AuboException
04886     *
04887     * @par JSON-RPC
04888     * {"jsonrpc":"2.0","method":"rob1.MotionControl.weaveEnd","params":[],"id":1}
04889     *
04890     * @par JSON-RPC
04891     * {"id":1,"jsonrpc":"2.0","result":0}
04892     *
04893     * \endchinese
04894     * \english
04895     * End weaving
04896     *
04897     * @return Returns 0 on success; otherwise returns an error code
04898     * AUBO_BUSY
04899     * AUBO_BAD_STATE
04900     * -AUBO_BAD_STATE
04901     *
04902     * @throws arcs::common_interface::AuboException
04903     *
04904     * @par JSON-RPC request example
04905     * {"jsonrpc":"2.0","method":"rob1.MotionControl.weaveEnd","params":[],"id":1}
04906     *
04907     * @par JSON-RPC response example
04908     * {"id":1,"jsonrpc":"2.0","result":0}
04909     *
04910     * \endenglish
04911     */
04912    int weaveEnd();
04913
04914    /**
04915     * @ingroup MotionControl
04916     * \chinese
04917     *
04918     *
04919     * @param sample_time        : m/s^2
04920     * @return    0
04921     * AUBO_BUSY
04922     * AUBO_BAD_STATE
```

```
04923     * -AUBO_INVL_ARGUMENT
04924     * -AUBO_BAD_STATE
04925     *
04926     * @throws arcs::common_interface::AuboException
04927     * \endchinese
04928     * \english
04929     * Set the sampling interval for points on the future path
04930     *
04931     * @param sample_time Sampling interval, unit: m/s^2
04932     * @return Returns 0 on success; otherwise returns an error code
04933     * AUBO_BUSY
04934     * AUBO_BAD_STATE
04935     * -AUBO_INVL_ARGUMENT
04936     * -AUBO_BAD_STATE
04937     *
04938     * @throws arcs::common_interface::AuboException
04939     * \endenglish
04940     */
04941    int setFuturePointSamplePeriod(double sample_time);
04942
04943    /**
04944     * @ingroup MotionControl
04945     * \chinese
04946     *
04947     *
04948     * @return  (100ms * 10)
04949     *
04950     * @throws arcs::common_interface::AuboException
04951     *
04952     * @par JSON-RPC
04953     * {"jsonrpc":"2.0","method":"rob1.MotionControl.getFuturePathPointsJoint","params":[],"id":1}
04954     *
04955     * @par JSON-RPC
04956     * {"id":1,"jsonrpc":"2.0","result":[]}
04957     *
04958     * \endchinese
04959     * \english
04960     * Get trajectory points on the future path
04961     *
04962     * @return Waypoints (100ms * 10)
04963     *
04964     * @throws arcs::common_interface::AuboException
04965     *
04966     * @par JSON-RPC request example
04967     * {"jsonrpc":"2.0","method":"rob1.MotionControl.getFuturePathPointsJoint","params":[],"id":1}
04968     *
04969     * @par JSON-RPC response example
04970     * {"id":1,"jsonrpc":"2.0","result":[]}
04971     *
04972     * \endenglish
04973     */
04974    std::vector<std::vector<double» getFuturePathPointsJoint();
04975
04976    /**
04977     * @ingroup MotionControl
04978     * \chinese
04979     *
04980     *
04981     * @param encoder_id
04982     * @param tick_per_meter
04983     *    ===>
04984     *    ===>
04985     * @return
04986     *
04987     * @throws arcs::common_interface::AuboException
04988     *
04989     * @par Python
04990     * setConveyorTrackEncoder(self: pyaubo_sdk.MotionControl) -> int
04991     *
04992     * @par Lua
04993     * setConveyorTrackEncoder(encoder_id: bumber,tick_per_meter: number) -> int
04994     *
04995     * @par Lua
04996     * num = setConveyorTrackEncoder(1,40000)
04997     *
04998     * @par JSON-RPC
04999     * {"jsonrpc":"2.0","method":"rob1.MotionControl.setConveyorTrackEncoder","params":[1,40000],"id":1}
05000     *
05001     * @par JSON-RPC
05002     * {"id":1,"jsonrpc":"2.0","result":0}
05003     *
05004     * \endchinese
05005     * \english
05006     * Set conveyor encoder parameters
05007     *
05008     * @param encoder_id Reserved
05009     * @param tick_per_meter
```

```
05010     * Linear conveyor: pulses per meter
05011     * Circular conveyor: pulses per revolution
05012     * @return
05013     *
05014     * @throws arcs::common_interface::AuboException
05015     *
05016     * @par Python function prototype
05017     * setConveyorTrackEncoder(self: pyaubo_sdk.MotionControl) -> int
05018     *
05019     * @par Lua function prototype
05020     * setConveyorTrackEncoder(encoder_id: bumber,tick_per_meter: number) -> int
05021     *
05022     * @par Lua example
05023     * num = setConveyorTrackEncoder(1,40000)
05024     *
05025     * @par JSON-RPC request example
05026     * {"jsonrpc":"2.0","method":"rob1.MotionControl.setConveyorTrackEncoder","params":[1,40000],"id":1}
05027     *
05028     * @par JSON-RPC response example
05029     * {"id":1,"jsonrpc":"2.0","result":0}
05030     *
05031     * \endenglish
05032     */
05033    int setConveyorTrackEncoder(int encoder_id, int tick_per_meter);
05034    /**
05035     * @ingroup MotionControl
05036     * \chinese
05037     *
05038     *
05039     * @note
05040     *
05041     * @param encoder_id
05042     * 0-
05043     * @param rotate_tool
05044     * @return
05045     *
05046     * @throws arcs::common_interface::AuboException
05047     * \endchinese
05048     * \english
05049     * Circular conveyor tracking
05050     *
05051     * @note Not implemented yet
05052     *
05053     * @param encoder_id
05054     * 0 - integrated sensor
05055     * @param rotate_tool
05056     * @return
05057     *
05058     * @throws arcs::common_interface::AuboException
05059     * \endenglish
05060     */
05061    int conveyorTrackCircle(int encoder_id, const std::vector<double> &center,
05062                            bool rotate_tool);
05063
05064    /**
05065     * @ingroup MotionControl
05066     * \chinese
05067     *
05068     *
05069     * @param encoder_id
05070     * @param direction
05071     * @return
05072     *
05073     * @throws arcs::common_interface::AuboException
05074     *
05075     * @par Python
05076     * conveyorTrackLine(self: pyaubo_sdk.MotionControl) -> int
05077     *
05078     * @par Lua
05079     * conveyorTrackLine -> int
05080     *
05081     * @par JSON-RPC
05082     * {"jsonrpc":"2.0","method":"rob1.MotionControl.conveyorTrackLine","params":[1,[1.0,0.0,0.0,0.0,0.0,0.0,0.0]],"id":1}
05083     *
05084     * @par JSON-RPC
05085     * {"id":1,"jsonrpc":"2.0","result":0}
05086     *
05087     * \endchinese
05088     * \english
05089     * Linear conveyor tracking
05090     *
05091     * @param encoder_id Reserved
05092     * @param direction The movement direction of the conveyor relative to the
05093     * robot base coordinate system
05094     * @return
05095     *
05096     * @throws arcs::common_interface::AuboException
```

```
05097     *
05098     * @par Python function prototype
05099     * conveyorTrackLine(self: pyaubo_sdk.MotionControl) -> int
05100     *
05101     * @par Lua function prototype
05102     * conveyorTrackLine -> int
05103     *
05104     * @par JSON-RPC request example
05105     * {"jsonrpc":"2.0","method":"rob1.MotionControl.conveyorTrackLine","params":[1,[1.0,0.0,0.0,0.0,0.0,0.0]],"id":1}
05106     *
05107     * @par JSON-RPC response example
05108     * {"id":1,"jsonrpc":"2.0","result":0}
05109     *
05110     * \endenglish
05111     */
05112    int conveyorTrackLine(int encoder_id, const std::vector<double> &direction);
05113
05114    /**
05115     * @ingroup MotionControl
05116     * \chinese
05117     *
05118     *
05119     * @param encoder_id
05120     * @param a
05121     * @return
05122     *
05123     * @throws arcs::common_interface::AuboException
05124     *
05125     * @par Python
05126     * conveyorTrackStop(self: pyaubo_sdk.MotionControl) -> int
05127     *
05128     * @par Lua
05129     * conveyorTrackStop -> int
05130     *
05131     * @par JSON-RPC
05132     * {"jsonrpc":"2.0","method":"rob1.MotionControl.conveyorTrackStop","params":[0,1.0],"id":1}
05133     *
05134     * @par JSON-RPC
05135     * {"id":1,"jsonrpc":"2.0","result":0}
05136     * \endchinese
05137     * \english
05138     * Stop conveyor tracking
05139     *
05140     * @param encoder_id Reserved
05141     * @param a Reserved
05142     * @return
05143     *
05144     * @throws arcs::common_interface::AuboException
05145     *
05146     * @par Python function prototype
05147     * conveyorTrackStop(self: pyaubo_sdk.MotionControl) -> int
05148     *
05149     * @par Lua function prototype
05150     * conveyorTrackStop -> int
05151     *
05152     * @par JSON-RPC request example
05153     * {"jsonrpc":"2.0","method":"rob1.MotionControl.conveyorTrackStop","params":[0,1.0],"id":1}
05154     *
05155     * @par JSON-RPC response example
05156     * {"id":1,"jsonrpc":"2.0","result":0}
05157     * \endenglish
05158     */
05159    int conveyorTrackStop(int encoder_id, double a);
05160
05161    /**
05162     * @ingroup MotionControl
05163     * \chinese
05164     *
05165     *                    true
05166     *            false
05167     *
05168     * @return
05169     *
05170     * @throws arcs::common_interface::AuboException
05171     *
05172     * @param encoder_id
05173     *
05174     * @par Python
05175     * conveyorTrackSwitch(self: pyaubo_sdk.MotionControl) -> bool
05176     *
05177     * @par Lua
05178     * conveyorTrackSwitch() -> boolean
05179     *
05180     * @par JSON-RPC
05181     * {"jsonrpc":"2.0","method":"rob1.MotionControl.conveyorTrackSwitch","params":[0],"id":1}
05182     *
05183     * @par JSON-RPC
```

```
05184     * {"id":1,"jsonrpc":"2.0","result":true}
05185     *
05186     * \endchinese
05187     * \english
05188     * Switch conveyor tracking item.
05189     * If the current item is being tracked, it will be dequeued and no longer
05190     * tracked, returning true. If no item is currently being tracked, returns
05191     * false.
05192     *
05193     * @return
05194     *
05195     * @throws arcs::common_interface::AuboException
05196     *
05197     * @param encoder_id Reserved
05198     *
05199     * @par Python function prototype
05200     * conveyorTrackSwitch(self: pyaubo_sdk.MotionControl) -> bool
05201     *
05202     * @par Lua function prototype
05203     * conveyorTrackSwitch() -> boolean
05204     *
05205     * @par JSON-RPC request example
05206     * {"jsonrpc":"2.0","method":"rob1.MotionControl.conveyorTrackSwitch","params":[0],"id":1}
05207     *
05208     * @par JSON-RPC response example
05209     * {"id":1,"jsonrpc":"2.0","result":true}
05210     *
05211     * \endenglish
05212     */
05213    bool conveyorTrackSwitch(int encoder_id);
05214
05215    /**
05216     * @ingroup MotionControl
05217     * \chinese
05218     *
05219     * @param encoder_id
05220     * @return
05221     *           true
05222     *
05223     *                    true
05224     *           false
05225     *
05226     * @throws arcs::common_interface::AuboException
05227     *
05228     * @par Python
05229     * hasItemOnConveyorToTrack(self: pyaubo_sdk.MotionControl) -> bool
05230     *
05231     * @par Lua
05232     * hasItemOnConveyorToTrack() -> boolean
05233     *
05234     * @par JSON-RPC
05235     * {"jsonrpc":"2.0","method":"rob1.MotionControl.hasItemOnConveyorToTrack","params":[0],"id":1}
05236     *
05237     * @par JSON-RPC
05238     * {"id":1,"jsonrpc":"2.0","result":{true}}
05239     * \endchinese
05240     * \english
05241     * Whether there is an item on the conveyor that can be tracked
05242     * @param encoder_id Reserved
05243     * @return
05244     * If the first item in the queue is in tracking state, returns true.
05245     * If the first item is not in tracking state, checks the rest of the queue
05246     * for items within the start window. Items outside the start window are
05247     * dequeued until an item is found within the window, which is then set to
05248     * tracking state and returns true. If no item in the queue is within the
05249     * start window, returns false.
05250     *
05251     * @throws arcs::common_interface::AuboException
05252     *
05253     * @par Python function prototype
05254     * hasItemOnConveyorToTrack(self: pyaubo_sdk.MotionControl) -> bool
05255     *
05256     * @par Lua function prototype
05257     * hasItemOnConveyorToTrack() -> boolean
05258     *
05259     * @par JSON-RPC request example
05260     * {"jsonrpc":"2.0","method":"rob1.MotionControl.hasItemOnConveyorToTrack","params":[0],"id":1}
05261     *
05262     * @par JSON-RPC response example
05263     * {"id":1,"jsonrpc":"2.0","result":{true}}
05264     * \endenglish
05265     */
05266    bool hasItemOnConveyorToTrack(int encoder_id);
05267
05268    /**
05269     * @ingroup MotionControl
05270     * \chinese
```

```
05271     *
05272     *
05273     * @param encoder_id
05274     * @param item_id  ID
05275     * @param offset
05276     * @return
05277     *
05278     * @throws arcs::common_interface::AuboException
05279     *
05280     * @par Python
05281     * conveyorTrackCreatItem(self: pyaubo_sdk.MotionControl, arg0: int,
05282     * arg1:int, arg2: List[float]) -> int
05283     *
05284     * @par Lua
05285     * conveyorTrackCreatItem(encoder_id: number, item_id: number, offset:
05286     * table) -> number
05287     *
05288     * @par JSON-RPC
05289     * {"jsonrpc":"2.0","method":"rob1.MotionControl.conveyorTrackCreatItem","params":[0,2,
05290     * {0.0,0.0,0.0,0.0,0.0,0.0}],"id":1}
05291     *
05292     * @par JSON-RPC
05293     * {"id":1,"jsonrpc":"2.0","result":0.0}
05294     * \endchinese
05295     * \english
05296     * Add an item to the conveyor queue
05297     *
05298     * @param encoder_id Reserved
05299     * @param item_id Item ID
05300     * @param offset Offset of the current item position relative to the
05301     * template item position
05302     * @return
05303     *
05304     * @throws arcs::common_interface::AuboException
05305     *
05306     * @par Python function prototype
05307     * conveyorTrackCreatItem(self: pyaubo_sdk.MotionControl, arg0: int,
05308     * arg1:int, arg2: List[float]) -> int
05309     *
05310     * @par Lua function prototype
05311     * conveyorTrackCreatItem(encoder_id: number, item_id: number, offset:
05312     * table) -> number
05313     *
05314     * @par JSON-RPC request example
05315     * {"jsonrpc":"2.0","method":"rob1.MotionControl.conveyorTrackCreatItem","params":[0,2,
05316     * {0.0,0.0,0.0,0.0,0.0,0.0}],"id":1}
05317     *
05318     * @par JSON-RPC response example
05319     * {"id":1,"jsonrpc":"2.0","result":0.0}
05320     * \endenglish
05321     */
05322    int conveyorTrackCreatItem(int encoder_id, int item_id,
05323                               const std::vector<double> &offset);
05324    /**
05325     * @ingroup MotionControl
05326     * \chinese
05327     *
05328     *
05329     * @param encoder_id
05330     * @param comp
05331     * @return
05332     *
05333     * @throws arcs::common_interface::AuboException
05334     *
05335     * @par Python
05336     * setConveyorTrackCompensate(self: pyaubo_sdk.MotionControl, arg0: int,
05337     * arg1: float) -> int
05338     *
05339     * @par Lua
05340     * setConveyorTrackCompensate(comp: number) -> number
05341     *
05342     * @par JSON-RPC
05343     * {"jsonrpc":"2.0","method":"rob1.MotionControl.setConveyorTrackCompensate","params":[0,
05344     * 0.1],"id":1}
05345     *
05346     * @par JSON-RPC
05347     * {"id":1,"jsonrpc":"2.0","result":0}
05348     * \endchinese
05349     * \english
05350     * Set the compensation value for conveyor tracking
05351     *
05352     * @param encoder_id Reserved
05353     * @param comp Conveyor compensation value
05354     * @return
05355     *
05356     * @throws arcs::common_interface::AuboException
05357     *
```

```
05358     * @par Python function prototype
05359     * setConveyorTrackCompensate(self: pyaubo_sdk.MotionControl, arg0: int,
05360     * arg1: float) -> int
05361     *
05362     * @par Lua function prototype
05363     * setConveyorTrackCompensate(comp: number) -> number
05364     *
05365     * @par JSON-RPC request example
05366     * {"jsonrpc":"2.0","method":"rob1.MotionControl.setConveyorTrackCompensate","params":[0,
05367     * 0.1],"id":1}
05368     *
05369     * @par JSON-RPC response example
05370     * {"id":1,"jsonrpc":"2.0","result":0}
05371     * \endenglish
05372     */
05373    int setConveyorTrackCompensate(int encoder_id, double comp);
05374
05375    /**
05376     * @ingroup MotionControl
05377     * \chinese
05378     *
05379     *
05380     * @param encoder_id
05381     * @return
05382     *
05383     * @throws arcs::common_interface::AuboException
05384     *
05385     * @par Python
05386     * isConveyorTrackSync(self: pyaubo_sdk.MotionControl, arg0: int) -> bool
05387     *
05388     * @par Lua
05389     * isConveyorTrackSync(encoder_id: number) -> bool
05390     *
05391     * @par JSON-RPC
05392     * {"jsonrpc":"2.0","method":"rob1.MotionControl.isConveyorTrackSync","params":[0],"id":1}
05393     *
05394     * @par JSON-RPC
05395     * {"id":1,"jsonrpc":"2.0","result":false}
05396     *
05397     * \endchinese
05398     * \english
05399     * Determine whether the conveyor and the robot arm have reached relative
05400     * rest
05401     *
05402     * @param encoder_id Reserved
05403     * @return
05404     *
05405     * @throws arcs::common_interface::AuboException
05406     *
05407     * @par Python function prototype
05408     * isConveyorTrackSync(self: pyaubo_sdk.MotionControl, arg0: int) -> bool
05409     *
05410     * @par Lua function prototype
05411     * isConveyorTrackSync(encoder_id: number) -> bool
05412     *
05413     * @par JSON-RPC request example
05414     * {"jsonrpc":"2.0","method":"rob1.MotionControl.isConveyorTrackSync","params":[0],"id":1}
05415     *
05416     * @par JSON-RPC response example
05417     * {"id":1,"jsonrpc":"2.0","result":false}
05418     *
05419     * \endenglish
05420     */
05421    bool isConveyorTrackSync(int encoder_id);
05422
05423    /**
05424     * @ingroup MotionControl
05425     * \chinese
05426     *
05427     *
05428     * @param encoder_id
05429     * @param limit
05430     *        ,
05431     * @return
05432     *
05433     * @throws arcs::common_interface::AuboException
05434     *
05435     * @par Python
05436     * setConveyorTrackLimit(self: pyaubo_sdk.MotionControl, arg0: int, arg1:
05437     * double) -> int
05438     *
05439     * @par Lua
05440     * setConveyorTrackLimit(encoder_id: number, limit: number) -> int
05441     *
05442     * @par JSON-RPC
05443     * {"jsonrpc":"2.0","method":"rob1.MotionControl.setConveyorTrackLimit","params":[0,
05444     * 1.5],"id":1}
```

```
05445     *
05446     * @par JSON-RPC
05447     * {"id":1,"jsonrpc":"2.0","result":0}
05448     * \endchinese
05449     * \english
05450     * Set the maximum distance limit for conveyor tracking
05451     *
05452     * @param encoder_id Reserved
05453     * @param limit
05454     * Maximum distance limit for conveyor tracking, unit: meter
05455     * @return
05456     *
05457     * @throws arcs::common_interface::AuboException
05458     *
05459     * @par Python function prototype
05460     * setConveyorTrackLimit(self: pyaubo_sdk.MotionControl, arg0: int, arg1:
05461     * double) -> int
05462     *
05463     * @par Lua function prototype
05464     * setConveyorTrackLimit(encoder_id: number, limit: number) -> int
05465     *
05466     * @par JSON-RPC request example
05467     * {"jsonrpc":"2.0","method":"rob1.MotionControl.setConveyorTrackLimit","params":[0,
05468     * 1.5],"id":1}
05469     *
05470     * @par JSON-RPC response example
05471     * {"id":1,"jsonrpc":"2.0","result":0}
05472     * \endenglish
05473     */
05474    int setConveyorTrackLimit(int encoder_id, double limit);
05475
05476    /**
05477     * @ingroup MotionControl
05478     * \chinese
05479     *
05480     *
05481     * @param encoder_id
05482     * @param min_window
05483     *        ,
05484     * @param max_window
05485     *        ,
05486     * @return
05487     *
05488     * @throws arcs::common_interface::AuboException
05489     *
05490     * @par Python
05491     * setConveyorTrackStartWindow(self: pyaubo_sdk.MotionControl, arg0: int,
05492     * arg1: double, arg2: double) -> int
05493     *
05494     * @par Lua
05495     * setConveyorTrackStartWindow(encoder_id: number, min_window: number,
05496     * max_window: number) -> int
05497     *
05498     * @par JSON-RPC
05499     * {"jsonrpc":"2.0","method":"rob1.MotionControl.setConveyorTrackStartWindow","params":[0,
05500     * 0.2, 1.0],"id":1}
05501     *
05502     * @par JSON-RPC
05503     * {"id":1,"jsonrpc":"2.0","result":0}
05504     *
05505     * \endchinese
05506     * \english
05507     * Set the start window for conveyor tracking
05508     *
05509     * @param encoder_id Reserved
05510     * @param min_window
05511     * Start position of the window, unit: meter
05512     * @param max_window
05513     * End position of the window, unit: meter
05514     * @return
05515     *
05516     * @throws arcs::common_interface::AuboException
05517     *
05518     * @par Python function prototype
05519     * setConveyorTrackStartWindow(self: pyaubo_sdk.MotionControl, arg0: int,
05520     * arg1: double, arg2: double) -> int
05521     *
05522     * @par Lua function prototype
05523     * setConveyorTrackStartWindow(encoder_id: number, min_window: number,
05524     * max_window: number) -> int
05525     *
05526     * @par JSON-RPC request example
05527     * {"jsonrpc":"2.0","method":"rob1.MotionControl.setConveyorTrackStartWindow","params":[0,
05528     * 0.2, 1.0],"id":1}
05529     *
05530     * @par JSON-RPC response example
05531     * {"id":1,"jsonrpc":"2.0","result":0}
```

```
05532     *
05533     * \endenglish
05534     */
05535    int setConveyorTrackStartWindow(int encoder_id, double window_min,
05536                                    double window_max);
05537
05538    /**
05539     * @ingroup MotionControl
05540     * \chinese
05541     *
05542     *
05543     * @param encoder_id
05544     * @param offset
05545     *              ,
05546     * @return
05547     *
05548     * @throws arcs::common_interface::AuboException
05549     *
05550     * @par Python
05551     * setConveyorTrackSensorOffset(self: pyaubo_sdk.MotionControl, arg0: int,
05552     * arg1: double) -> int
05553     *
05554     * @par Lua
05555     * setConveyorTrackSensorOffset(encoder_id: number, offset: number) -> int
05556     *
05557     * @par JSON-RPC
05558     * {"jsonrpc":"2.0","method":"rob1.MotionControl.setConveyorTrackSensorOffset","params":[0,
05559     * 0.2],"id":1}
05560     *
05561     * @par JSON-RPC
05562     * {"id":1,"jsonrpc":"2.0","result":0}
05563     *
05564     * \endchinese
05565     * \english
05566     * Set the distance from the conveyor teaching position to the sync switch
05567     *
05568     * @param encoder_id Reserved
05569     * @param offset
05570     * Distance from the conveyor teaching position to the sync switch, unit:
05571     * meter
05572     * @return
05573     *
05574     * @throws arcs::common_interface::AuboException
05575     *
05576     * @par Python function prototype
05577     * setConveyorTrackSensorOffset(self: pyaubo_sdk.MotionControl, arg0: int,
05578     * arg1: double) -> int
05579     *
05580     * @par Lua function prototype
05581     * setConveyorTrackSensorOffset(encoder_id: number, offset: number) -> int
05582     *
05583     * @par JSON-RPC request example
05584     * {"jsonrpc":"2.0","method":"rob1.MotionControl.setConveyorTrackSensorOffset","params":[0,
05585     * 0.2],"id":1}
05586     *
05587     * @par JSON-RPC response example
05588     * {"id":1,"jsonrpc":"2.0","result":0}
05589     *
05590     * \endenglish
05591     */
05592    int setConveyorTrackSensorOffset(int encoder_id, double offset);
05593
05594    /**
05595     * @ingroup MotionControl
05596     * \chinese
05597     *
05598     *
05599     * @param encoder_id
05600     * @param distance
05601     *                  ,
05602     * @param time
05603     *                  ,
05604     *
05605     * distance time    0
05606     *  distance time      distance
05607     * @return
05608     *
05609     * @throws arcs::common_interface::AuboException
05610     *
05611     * @par Python
05612     * setConveyorTrackSyncSeparation(self: pyaubo_sdk.MotionControl, arg0: int,
05613     * arg1: double, arg2: double) -> int
05614     *
05615     * @par Lua
05616     * setConveyorTrackSyncSeparation(encoder_id: number, distance: number,
05617     * time: number) -> int
05618     *
```

```
05619     * @par JSON-RPC
05620     * {”jsonrpc”:”2.0”,”method”:”rob1.MotionControl.setConveyorTrackSyncSeparation”,”params”:[0,
05621     * 0.05, 0.2],”id”:1}
05622     *
05623     * @par JSON-RPC
05624     * {”id”:1,”jsonrpc”:”2.0”,”result”:0}
05625     * \endchinese
05626     * \english
05627     * Set conveyor sync separation, used to filter out unwanted signals from
05628     * the sync switch.
05629     *
05630     * @param encoder_id Reserved
05631     * @param distance
05632     * The minimum distance to travel after a sync signal appears before
05633     * accepting a new sync signal as a valid object, unit: meter
05634     * @param time
05635     * The minimum time to elapse after a sync signal appears before accepting a
05636     * new sync signal as a valid object, unit: second
05637     *
05638     * If distance and time are both set greater than 0, the setting takes
05639     * effect. If both distance and time are set, distance takes precedence.
05640     * @return
05641     *
05642     * @throws arcs::common_interface::AuboException
05643     *
05644     * @par Python function prototype
05645     * setConveyorTrackSyncSeparation(self: pyaubo_sdk.MotionControl, arg0: int,
05646     * arg1: double, arg2: double) -> int
05647     *
05648     * @par Lua function prototype
05649     * setConveyorTrackSyncSeparation(encoder_id: number, distance: number,
05650     * time: number) -> int
05651     *
05652     * @par JSON-RPC request example
05653     * {”jsonrpc”:”2.0”,”method”:”rob1.MotionControl.setConveyorTrackSyncSeparation”,”params”:[0,
05654     * 0.05, 0.2],”id”:1}
05655     *
05656     * @par JSON-RPC response example
05657     * {”id”:1,”jsonrpc”:”2.0”,”result”:0}
05658     * \endenglish
05659     */
05660    int setConveyorTrackSyncSeparation(int encoder_id, double distance,
05661                                       double time);
05662
05663    /**
05664     * @ingroup MotionControl
05665     * \chinese
05666     *
05667     *
05668     * @param encoder_id
05669     * @param threshold
05670     *           (   0.2),    /
05671     *
05672     *
05673     * @return
05674     *
05675     * @throws arcs::common_interface::AuboException
05676     *
05677     * @par Python
05678     * setConveyorTrackVelThreshold(self: pyaubo_sdk.MotionControl, arg0: int,
05679     * arg1: double) -> int
05680     *
05681     * @par Lua
05682     * setConveyorTrackVelThreshold(encoder_id: number, threshold: number) ->
05683     * int
05684     *
05685     * @par JSON-RPC
05686     * {”jsonrpc”:”2.0”,”method”:”rob1.MotionControl.setConveyorTrackVelThreshold”,”params”:[0,
05687     * 0.2],”id”:1}
05688     *
05689     * @par JSON-RPC
05690     * {”id”:1,”jsonrpc”:”2.0”,”result”:0}
05691     * \endchinese
05692     * \english
05693     * Set the conveyor velocity threshold used to suppress position jumps
05694     * caused by abnormal encoder spikes.
05695     *
05696     * @param encoder_id Reserved
05697     * @param threshold
05698     * The maximum allowable conveyor speed during operation, unit:
05699     * meter/second. Default: 0.2 m/s.
05700     *
05701     * @return
05702     * 0 on success; otherwise a non-zero error code.
05703     *
05704     * @throws arcs::common_interface::AuboException
05705     *
```

```
05706     * @par Python function prototype
05707     * setConveyorTrackVelThreshold(self: pyaubo_sdk.MotionControl, arg0: int,
05708     * arg1: double) -> int
05709     *
05710     * @par Lua function prototype
05711     * setConveyorTrackVelThreshold(encoder_id: number, threshold: number) ->
05712     * int
05713     *
05714     * @par JSON-RPC request example
05715     * {"jsonrpc":"2.0","method":"rob1.MotionControl.setConveyorTrackVelThreshold","params":[0,
05716     * 0.2],"id":1}
05717     *
05718     * @par JSON-RPC response example
05719     * {"id":1,"jsonrpc":"2.0","result":0}
05720     * \endenglish
05721     */
05722    int setConveyorTrackVelThreshold(int encoder_id, double threshold);
05723
05724    /**
05725     * \chinese
05726     *
05727     *
05728     * @param encoder_id
05729     * @return
05730     * true :
05731     * false :
05732     *
05733     * @throws arcs::common_interface::AuboException
05734     *
05735     * @par Python
05736     * isConveyorTrackExceed(self: pyaubo_sdk.MotionControl, arg0: int) -> bool
05737     *
05738     * @par Lua
05739     * isConveyorTrackExceed(encoder_id: number) -> bool
05740     *
05741     * @par JSON-RPC
05742     * {"jsonrpc":"2.0","method":"rob1.MotionControl.isConveyorTrackExceed","params":[0],"id":1}
05743     *
05744     * @par JSON-RPC
05745     * {"id":1,"jsonrpc":"2.0","result":false}
05746     * \endchinese
05747     * \english
05748     * Whether the workpiece on the conveyor has moved beyond the maximum limit
05749     *
05750     * @param encoder_id Reserved
05751     * @return
05752     * true : The movement distance exceeds the maximum limit
05753     * false : The movement distance does not exceed the maximum limit
05754     *
05755     * @throws arcs::common_interface::AuboException
05756     *
05757     * @par Python function prototype
05758     * isConveyorTrackExceed(self: pyaubo_sdk.MotionControl, arg0: int) -> bool
05759     *
05760     * @par Lua function prototype
05761     * isConveyorTrackExceed(encoder_id: number) -> bool
05762     *
05763     * @par JSON-RPC request example
05764     * {"jsonrpc":"2.0","method":"rob1.MotionControl.isConveyorTrackExceed","params":[0],"id":1}
05765     *
05766     * @par JSON-RPC response example
05767     * {"id":1,"jsonrpc":"2.0","result":false}
05768     * \endenglish
05769     */
05770    bool isConveyorTrackExceed(int encoder_id);
05771
05772    /**
05773     * @ingroup MotionControl
05774     * \chinese
05775     *
05776     *
05777     * @param encoder_id
05778     * @return
05779     *
05780     * @throws arcs::common_interface::AuboException
05781     *
05782     * @par Python
05783     * conveyorTrackClearItems(self: pyaubo_sdk.MotionControl, arg0: int) -> int
05784     *
05785     * @par Lua
05786     * conveyorTrackClearItems(encoder_id: number) -> int
05787     *
05788     * @par JSON-RPC
05789     * {"jsonrpc":"2.0","method":"rob1.MotionControl.conveyorTrackClearItems","params":[0],"id":1}
05790     *
05791     * @par JSON-RPC
05792     * {"id":1,"jsonrpc":"2.0","result":0}
```

```
05793     *
05794     * \endchinese
05795     * \english
05796     * Clear all items in the conveyor queue
05797     *
05798     * @param encoder_id Reserved
05799     * @return
05800     *
05801     * @throws arcs::common_interface::AuboException
05802     *
05803     * @par Python function prototype
05804     * conveyorTrackClearItems(self: pyaubo_sdk.MotionControl, arg0: int) -> int
05805     *
05806     * @par Lua function prototype
05807     * conveyorTrackClearItems(encoder_id: number) -> int
05808     *
05809     * @par JSON-RPC request example
05810     * {"jsonrpc":"2.0","method":"rob1.MotionControl.conveyorTrackClearItems","params":[0],"id":1}
05811     *
05812     * @par JSON-RPC response example
05813     * {"id":1,"jsonrpc":"2.0","result":0}
05814     *
05815     * \endenglish
05816     */
05817    int conveyorTrackClearItems(int encoder_id);
05818
05819    /**
05820     * @ingroup MotionControl
05821     * \chinese
05822     *
05823     *
05824     * @param encoder_id
05825     * @return
05826     *
05827     * @throws arcs::common_interface::AuboException
05828     *
05829     * @par Python
05830     * getConveyorTrackQueue(self: pyaubo_sdk.MotionControl, arg0: int) ->
05831     * List[int]
05832     * @par Lua
05833     * getConveyorTrackQueue(encoder_id: number) -> table
05834     *
05835     * @par JSON-RPC
05836     * {"jsonrpc":"2.0","method":"rob1.MotionControl.getConveyorTrackQueue","params":[0],"id":1}
05837     *
05838     * @par JSON-RPC
05839     * {"id":1,"jsonrpc":"2.0","result":{[-500,-200,150,-50]}}
05840     * \endchinese
05841     * \english
05842     * Get encoder values of the conveyor queue
05843     *
05844     * @param encoder_id Reserved
05845     * @return
05846     *
05847     * @throws arcs::common_interface::AuboException
05848     *
05849     * @par Python function prototype
05850     * getConveyorTrackQueue(self: pyaubo_sdk.MotionControl, arg0: int) ->
05851     * List[int]
05852     * @par Lua function prototype
05853     * getConveyorTrackQueue(encoder_id: number) -> table
05854     *
05855     * @par JSON-RPC request example
05856     * {"jsonrpc":"2.0","method":"rob1.MotionControl.getConveyorTrackQueue","params":[0],"id":1}
05857     *
05858     * @par JSON-RPC response example
05859     * {"id":1,"jsonrpc":"2.0","result":{[-500,-200,150,-50]}}
05860     * \endenglish
05861     */
05862    std::vector<int> getConveyorTrackQueue(int encoder_id);
05863    /**
05864     *  \chinese
05865     *          id
05866     *
05867     * @param encoder_id
05868     * @return
05869     *    id  next item  -1
05870     *
05871     * @throws arcs::common_interface::AuboException
05872     *
05873     * @par Python
05874     *  getConveyorTrackNextItem(self: pyaubo_sdk.MotionControl, arg0: int) ->
05875     * int
05876     * @par Lua
05877     *  getConveyorTrackNextItem(encoder_id: number) -> int
05878     *
05879     * @par JSON-RPC
```

```
05880     * {"jsonrpc":"2.0","method":"rob1.MotionControl.getConveyorTrackNextItem","params":[0],"id":1}
05881     *
05882     * @par JSON-RPC
05883     * {"id":1,"jsonrpc":"2.0","result":{10}}
05884     * \endchinese
05885     * \english
05886     * Get the ID of the next item on the conveyor to be tracked
05887     *
05888     * @param encoder_id Reserved
05889     * @return
05890     * Returns the item ID; returns -1 if there is no next item
05891     *
05892     * @throws arcs::common_interface::AuboException
05893     *
05894     * @par Python function prototype
05895     *  getConveyorTrackNextItem(self: pyaubo_sdk.MotionControl, arg0: int) ->
05896     * int
05897     * @par Lua function prototype
05898     *  getConveyorTrackNextItem(encoder_id: number) -> int
05899     *
05900     * @par JSON-RPC Request Example
05901     * {"jsonrpc":"2.0","method":"rob1.MotionControl.getConveyorTrackNextItem","params":[0],"id":1}
05902     *
05903     * @par JSON-RPC Response Example
05904     * {"id":1,"jsonrpc":"2.0","result":{10}}
05905     * \endenglish
05906     */
05907
05908    int getConveyorTrackNextItem(int encoder_id);
05909
05910    /**
05911     * @ingroup MotionControl
05912     * \chinese
05913     *
05914     *
05915     * @param param
05916     * @param blend_radius
05917     * @param v
05918     * @param a
05919     * @param t
05920     * @return    0
05921     * AUBO_BUSY
05922     * AUBO_BAD_STATE
05923     * -AUBO_INVL_ARGUMENT
05924     * -AUBO_BAD_STATE
05925     *
05926     * @throws arcs::common_interface::AuboException
05927     *
05928     * @par JSON-RPC
05929     * {"jsonrpc":"2.0","method":"rob1.MotionControl.moveSpiral",
05930     * "params":[{"spiral":0.005,"helix":0.005,"angle":18.84,"plane":0,"frame":[0,0,0,0,0,0]},0,0.25,1.2,0],"id":1}
05931     *
05932     * @par JSON-RPC
05933     * {"id":1,"jsonrpc":"2.0","result":0}
05934     * \endchinese
05935     * \english
05936     * Spiral motion
05937     *
05938     * @param param Encapsulated parameters
05939     * @param blend_radius
05940     * @param v
05941     * @param a
05942     * @param t
05943     * @return Returns 0 on success; otherwise returns an error code
05944     * AUBO_BUSY
05945     * AUBO_BAD_STATE
05946     * -AUBO_INVL_ARGUMENT
05947     * -AUBO_BAD_STATE
05948     *
05949     * @throws arcs::common_interface::AuboException
05950     *
05951     * @par JSON-RPC request example
05952     * {"jsonrpc":"2.0","method":"rob1.MotionControl.moveSpiral",
05953     * "params":[{"spiral":0.005,"helix":0.005,"angle":18.84,"plane":0,"frame":[0,0,0,0,0,0]},0,0.25,1.2,0],"id":1}
05954     *
05955     * @par JSON-RPC response example
05956     * {"id":1,"jsonrpc":"2.0","result":0}
05957     * \endenglish
05958     */
05959    int moveSpiral(const SpiralParameters &param, double blend_radius, double v,
05960                   double a, double t);
05961
05962    /**
05963     * @ingroup MotionControl
05964     * \chinese
05965     *
05966     *
```

```
05967     * @return
05968     *
05969     * @throws arcs::common_interface::AuboException
05970     *
05971     * @par Python
05972     * getLookAheadSize(self: pyaubo_sdk.MotionControl) -> int
05973     *
05974     * @par Lua
05975     * getLookAheadSize() -> number
05976     *
05977     * @par JSON-RPC
05978     * {"jsonrpc":"2.0","method":"rob1.MotionControl.getLookAheadSize","params":[],"id":1}
05979     *
05980     * @par JSON-RPC
05981     * {"id":1,"jsonrpc":"2.0","result":1}
05982     *
05983     * \endchinese
05984     * \english
05985     * Get look-ahead segment size
05986     *
05987     * @return Returns the look-ahead segment size
05988     *
05989     * @throws arcs::common_interface::AuboException
05990     *
05991     * @par Python function prototype
05992     * getLookAheadSize(self: pyaubo_sdk.MotionControl) -> int
05993     *
05994     * @par Lua function prototype
05995     * getLookAheadSize() -> number
05996     *
05997     * @par JSON-RPC request example
05998     * {"jsonrpc":"2.0","method":"rob1.MotionControl.getLookAheadSize","params":[],"id":1}
05999     *
06000     * @par JSON-RPC response example
06001     * {"id":1,"jsonrpc":"2.0","result":1}
06002     *
06003     * \endenglish
06004     */
06005    int getLookAheadSize();
06006
06007    /**
06008     * @ingroup MotionControl
06009     * \chinese
06010     *
06011     * 1.          ,   , ,    ,           ,      ;
06012     * 2.        ,       ,    ,              .
06013     *
06014     * @return    0
06015     *
06016     * @throws arcs::common_interface::AuboException
06017     *
06018     * @par Python
06019     * setLookAheadSize(self: pyaubo_sdk.MotionControl, arg0: int) -> int
06020     *
06021     * @par Lua
06022     * setLookAheadSize(eqradius: number) -> number
06023     *
06024     * @par JSON-RPC
06025     * {"jsonrpc":"2.0","method":"rob1.MotionControl.setLookAheadSize","params":[1],"id":1}
06026     *
06027     * @par JSON-RPC
06028     * {"id":1,"jsonrpc":"2.0","result":0}
06029     * \endchinese
06030     * \english
06031     * Set look-ahead segment size
06032     * 1. For tasks requiring high speed smoothness, such as CNC machining,
06033     * gluing, welding, etc., a longer look-ahead segment size can provide
06034     * better speed planning and smoother motion.
06035     * 2. For fast-response tasks such as picking, a shorter look-ahead segment
06036     * size is preferred to improve response speed, but may cause large speed
06037     * fluctuations due to insufficiently timely path feeding.
06038     *
06039     * @return Returns 0 on success
06040     *
06041     * @throws arcs::common_interface::AuboException
06042     *
06043     * @par Python function prototype
06044     * setLookAheadSize(self: pyaubo_sdk.MotionControl, arg0: int) -> int
06045     *
06046     * @par Lua function prototype
06047     * setLookAheadSize(eqradius: number) -> number
06048     *
06049     * @par JSON-RPC request example
06050     * {"jsonrpc":"2.0","method":"rob1.MotionControl.setLookAheadSize","params":[1],"id":1}
06051     *
06052     * @par JSON-RPC response example
06053     * {"id":1,"jsonrpc":"2.0","result":0}
```

```
06054     * \endenglish
06055     */
06056    int setLookAheadSize(int size);
06057
06058    /**
06059     * @ingroup MotionControl
06060     * \chinese
06061     *
06062     *
06063     * @param params Json
06064     * {
06065     *   "frequency": <num>,
06066     *   "amplitude": {<num>,<num>}
06067     * }
06068     *
06069     * @return    0
06070     *
06071     * @throws arcs::common_interface::AuboException
06072     * \endchinese
06073     * \english
06074     * Update frequency and amplitude during weaving process
06075     *
06076     * @param params Json string used to define weaving parameters
06077     * {
06078     *   "frequency": <num>,
06079     *   "amplitude": {<num>,<num>}
06080     * }
06081     *
06082     * @return Returns 0 on success
06083     *
06084     * @throws arcs::common_interface::AuboException
06085     * \endenglish
06086     */
06087    int weaveUpdateParameters(const std::string &params);
06088
06089    /**
06090     * @ingroup MotionControl
06091     * \chinese
06092     *
06093     *
06094     * @param stiffness          [0 -> 1]
06095     * @return
06096     *
06097     * @throws arcs::common_interface::AuboException
06098     *
06099     * @par Python
06100     * enableJointSoftServo(self: pyaubo_sdk.RobotConfig, arg0: List[float]) ->
06101     * int
06102     *
06103     * @par Lua
06104     * enableJointSoftServo(stiffness: table) -> int
06105     *
06106     * @par JSON-RPC
06107     * {"jsonrpc":"2.0","method":"rob1.RobotConfig.enableJointSoftServo","params":[[0.0,0.0,0.0,0.0,0.0,0.0]],"id":1}
06108     *
06109     * @par JSON-RPC
06110     * {"id":1,"jsonrpc":"2.0","result":0}
06111     * \endchinese
06112     * \english
06113     * Set joint current loop stiffness coefficient
06114     *
06115     * @param stiffness Stiffness coefficient for each joint, as a percentage [0
06116     * -> 1]. The larger the value, the stiffer the joint.
06117     * @return
06118     *
06119     * @throws arcs::common_interface::AuboException
06120     *
06121     * @par Python function prototype
06122     * enableJointSoftServo(self: pyaubo_sdk.RobotConfig, arg0: List[float]) ->
06123     * int
06124     *
06125     * @par Lua function prototype
06126     * enableJointSoftServo(stiffness: table) -> int
06127     *
06128     * @par JSON-RPC request example
06129     * {"jsonrpc":"2.0","method":"rob1.RobotConfig.enableJointSoftServo","params":[[0.0,0.0,0.0,0.0,0.0,0.0]],"id":1}
06130     *
06131     * @par JSON-RPC response example
06132     * {"id":1,"jsonrpc":"2.0","result":0}
06133     * \endenglish
06134     */
06135    int enableJointSoftServo(const std::vector<double> &stiffness);
06136
06137    /**
06138     * @ingroup MotionControl
06139     * \chinese
06140     *
```

```
06141     *
06142     * @return  0
06143     *
06144     * @throws arcs::common_interface::AuboException
06145     *
06146     * @par Python
06147     * disableJointSoftServo(self: pyaubo_sdk.MotionControl) -> int
06148     *
06149     * @par Lua
06150     * disableJointSoftServo() -> number
06151     *
06152     * @par JSON-RPC
06153     * {"jsonrpc":"2.0","method":"rob1.MotionControl.disableJointSoftServo","params":[],"id":1}
06154     *
06155     * @par JSON-RPC
06156     * {"id":1,"jsonrpc":"2.0","result":1}
06157     * \endchinese
06158     * \english
06159     * Disable joint current loop stiffness coefficient
06160     *
06161     * @return Returns 0 on success, otherwise error code
06162     *
06163     * @throws arcs::common_interface::AuboException
06164     *
06165     * @par Python function prototype
06166     * disableJointSoftServo(self: pyaubo_sdk.MotionControl) -> int
06167     *
06168     * @par Lua function prototype
06169     * disableJointSoftServo() -> number
06170     *
06171     * @par JSON-RPC request example
06172     * {"jsonrpc":"2.0","method":"rob1.MotionControl.disableJointSoftServo","params":[],"id":1}
06173     *
06174     * @par JSON-RPC response example
06175     * {"id":1,"jsonrpc":"2.0","result":1}
06176     * \endenglish
06177     */
06178    int disableJointSoftServo();
06179
06180    /**
06181     * @ingroup MotionControl
06182     * \chinese
06183     *
06184     *
06185     * @return     true     false
06186     *
06187     * @throws arcs::common_interface::AuboException
06188     *
06189     * @par Python
06190     * isJointSoftServoEnabled(self: pyaubo_sdk.MotionControl) -> bool
06191     *
06192     * @par Lua
06193     * isJointSoftServoEnabled() -> boolean
06194     *
06195     * @par JSON-RPC
06196     * {"jsonrpc":"2.0","method":"rob1.MotionControl.isJointSoftServoEnabled","params":[],"id":1}
06197     *
06198     * @par JSON-RPC
06199     * {"id":1,"jsonrpc":"2.0","result":1}
06200     * \endchinese
06201     * \english
06202     * Determine whether the joint current loop stiffness coefficient is
06203     * enabled.
06204     *
06205     * @return Returns true if enabled, otherwise returns false.
06206     *
06207     * @throws arcs::common_interface::AuboException
06208     *
06209     * @par Python function prototype
06210     * isJointSoftServoEnabled(self: pyaubo_sdk.MotionControl) -> bool
06211     *
06212     * @par Lua function prototype
06213     * isJointSoftServoEnabled() -> boolean
06214     *
06215     * @par JSON-RPC request example
06216     * {"jsonrpc":"2.0","method":"rob1.MotionControl.isJointSoftServoEnabled","params":[],"id":1}
06217     *
06218     * @par JSON-RPC response example
06219     * {"id":1,"jsonrpc":"2.0","result":1}
06220     * \endenglish
06221     */
06222    bool isJointSoftServoEnabled();
06223
06224    /**
06225     * @ingroup MotionControl
06226     * \chinese
06227     *
```

```
06228     *
06229     * @param omega      ,            . 0
06230     * \n
06231     * @param zeta       ,            . 0
06232     * \n
06233     * @param level
06234     *     (1~3),              .  0
06235     * \n
06236     * @retval 0
06237     * @retval AUBO_BAD_STATE(1)
06238     * Normal ReducedMode Recovery
06239     * @retval AUBO_QUEUE_FULL(2)
06240     * @retval AUBO_BUSY(3)
06241     * @retval -AUBO_BAD_STATE(-1)
06242     *                 task_id
06243     *           Running
06244     * @retval -AUBO_TIMEOUT(-4)
06245     *
06246     * @throws arcs::common_interface::AuboException
06247     *
06248     * @par Python
06249     * enableVibrationSuppress(self: pyaubo_sdk.MotionControl, arg0:
06250     * list[float], arg1: list[float], arg2: int) -> int
06251     *
06252     * @par Lua
06253     * enableVibrationSuppress(omega: table, zeta: table, level: number) -> nil
06254     *
06255     * @par JSON-RPC
06256     * {"json-
    rpc":"2.0","method":"rob1.MotionControl.enableVibrationSuppress","params":[[0.5,0.5,0.5,0.5,0.5,0.5],[0.5,0.5,0.5,0.5,0.5,0.5],2],"id":1}
06257     *
06258     * @par JSON-RPC
06259     * {"id":1,"jsonrpc":"2.0","result":0}
06260     *
06261     * \endchinese
06262     */
06263    int enableVibrationSuppress(const std::vector<double> &omega,
06264                                const std::vector<double> &zeta, int level);
06265    /**
06266     * @ingroup MotionControl
06267     * \chinese
06268     *
06269     *
06270     * @return     true     false
06271     *
06272     * @throws arcs::common_interface::AuboException
06273     *
06274     * @par Python
06275     * disbaleVibrationSuppress(self: pyaubo_sdk.MotionControl) -> int
06276     *
06277     * @par Lua
06278     * disbaleVibrationSuppress() -> number
06279     *
06280     * @par JSON-RPC
06281     * {"jsonrpc":"2.0","method":"rob1.MotionControl.disbaleVibrationSuppress","params":[],"id":1}
06282     *
06283     * @par JSON-RPC
06284     * {"id":1,"jsonrpc":"2.0","result":1}
06285     * \endchinese
06286     */
06287    int disbaleVibrationSuppress();
06288
06289    /**
06290     * @ingroup MotionControl
06291     * \chinese
06292     *
06293     *
06294     * @param enable
06295     *
06296     * @throws arcs::common_interface::AuboException
06297     *
06298     * @par Lua
06299     * setTimeOptimalEnable() -> bool
06300     *
06301     * @par Lua
06302     * setTimeOptimalEnable(true)
06303     *
06304     * @par JSON-RPC
06305     * {"jsonrpc":"2.0","method":"rob1.MotionControl.setTimeOptimalEnable","params":[true],"id":1}
06306     *
06307     * @par JSON-RPC
06308     * {"id":1,"jsonrpc":"2.0","result":0}
06309     * \endchinese
06310     * \english
06311     *
06312     * @param enable
06313     *
```

```
06314     * @throws arcs::common_interface::AuboException
06315     *
06316     * @par Lua function prototype
06317     * setTimeOptimalEnable() -> number
06318     *
06319     * @par Lua example
06320     * setTimeOptimalEnable(true)
06321     *
06322     * @par JSON-RPC request example
06323     * {"jsonrpc":"2.0","method":"rob1.MotionControl.setTimeOptimalEnable","params":[true],"id":1}
06324     *
06325     * @par JSON-RPC response example
06326     * {"id":1,"jsonrpc":"2.0","result":0}
06327     * \endenglish
06328     */
06329    int setTimeOptimalEnable(bool enable);
06330
06331    /**
06332     * @ingroup MotionControl
06333     * \chinese
06334     *
06335     *   true -
06336     *   false -
06337     *
06338     * @return
06339     *
06340     * @throws arcs::common_interface::AuboException
06341     *
06342     * @par Python
06343     * isTimeOptimalEnabled(self: pyaubo_sdk.MotionControl) -> float
06344     *
06345     * @par Lua
06346     * isTimeOptimalEnabled() -> number
06347     *
06348     * @par Lua
06349     * num = isTimeOptimalEnabled()
06350     *
06351     * @par JSON-RPC
06352     * {"jsonrpc":"2.0","method":"rob1.MotionControl.isTimeOptimalEnabled","params":[],"id":1}
06353     *
06354     * @par JSON-RPC
06355     * {"id":1,"jsonrpc":"2.0","result":1.0}
06356     * \endchinese
06357     * \english
06358     * Get the time optimal algorithm state:
06359     *   true - enable
06360     *   false - disable
06361     *
06362     * @return Returns the time optimal algorithm state
06363     *
06364     * @throws arcs::common_interface::AuboException
06365     *
06366     * @par Python function prototype
06367     * isTimeOptimalEnabled(self: pyaubo_sdk.MotionControl) -> bool
06368     *
06369     * @par Lua function prototype
06370     * isTimeOptimalEnabled() -> boolean
06371     *
06372     * @par Lua example
06373     * flag = isTimeOptimalEnabled()
06374     *
06375     * @par JSON-RPC request example
06376     * {"jsonrpc":"2.0","method":"rob1.MotionControl.isTimeOptimalEnabled","params":[],"id":1}
06377     *
06378     * @par JSON-RPC response example
06379     * {"id":1,"jsonrpc":"2.0","result":true}
06380     * \endenglish
06381     */
06382    bool isTimeOptimalEnabled();
06383
06384    /**
06385     * @ingroup MotionControl
06386     * \chinese
06387     *
06388     *   true -
06389     *   false -
06390     *
06391     * @return
06392     *
06393     * @throws arcs::common_interface::AuboException
06394     *
06395     * @par Python
06396     * isTimeOptimalEnabled(self: pyaubo_sdk.MotionControl) -> bool
06397     *
06398     * @par Lua
06399     * isSupportedTimeOptimal() -> boolean
06400     *
```

```
06401     * @par Lua
06402     * num = isSupportedTimeOptimal()
06403     *
06404     * @par JSON-RPC
06405     * {"jsonrpc":"2.0","method":"rob1.MotionControl.isSupportedTimeOptimal","params":[],"id":1}
06406     *
06407     * @par JSON-RPC
06408     * {"id":1,"jsonrpc":"2.0","result":1.0}
06409     * \endchinese
06410     * \english
06411     * Check whether the time optimal algorithm is supported
06412     *   true - supported
06413     *   false - not supported
06414     *
06415     * @return Return whether the time-optimal algorithm is supported
06416     *
06417     * @throws arcs::common_interface::AuboException
06418     *
06419     * @par Python function prototype
06420     * isSupportedTimeOptimal(self: pyaubo_sdk.MotionControl) -> bool
06421     *
06422     * @par Lua function prototype
06423     * isSupportedTimeOptimal() -> boolean
06424     *
06425     * @par Lua example
06426     * flag = isSupportedTimeOptimal()
06427     *
06428     * @par JSON-RPC request example
06429     * {"jsonrpc":"2.0","method":"rob1.MotionControl.isSupportedTimeOptimal","params":[],"id":1}
06430     *
06431     * @par JSON-RPC response example
06432     * {"id":1,"jsonrpc":"2.0","result":true}
06433     * \endenglish
06434     */
06435    bool isSupportedTimeOptimal();
06436
06437    /**
06438     * @ingroup MotionControl
06439     * \chinese
06440     *  TCP
06441     *
06442     * @param v TCP       m/s  /
06443     *
06444     * @throws arcs::common_interface::AuboException
06445     *
06446     * @par Lua
06447     * setTcpMaxLinearVelocity(number) -> number
06448     *
06449     * @par Lua
06450     * setTcpMaxLinearVelocity(0.5)  --   TCP     0.5 /
06451     *
06452     * @par Python
06453     * setTcpMaxLinearVelocity(v: float) -> int
06454     *
06455     * @par Python
06456     * setTcpMaxLinearVelocity(0.5) --   TCP     0.5 /
06457     *
06458     * @par JSON-RPC
06459     * {"jsonrpc":"2.0","method":"rob1.MotionControl.setTcpMaxLinearVelocity","params":[0.5],"id":1}
06460     *
06461     * @par JSON-RPC
06462     * {"id":1,"jsonrpc":"2.0","result":0}
06463     * \endchinese
06464     * \english
06465     * Set TCP maximum linear velocity
06466     *
06467     * @param v TCP maximum linear velocity value, unit is m/s
06468     *
06469     * @throws arcs::common_interface::AuboException
06470     *
06471     * @par Lua function prototype
06472     * setTcpMaxLinearVelocity(number) -> number
06473     *
06474     * @par Lua example
06475     * setTcpMaxLinearVelocity(0.5)  -- Set TCP maximum linear velocity to 0.5
06476     * meters per second
06477     *
06478     * @par Python function prototype
06479     * setTcpMaxLinearVelocity(v: float) -> int
06480     *
06481     * @par Python example
06482     * setTcpMaxLinearVelocity(0.5)  -- Set TCP maximum linear velocity to 0.5
06483     *
06484     * @par JSON-RPC request example
06485     * {"jsonrpc":"2.0","method":"rob1.MotionControl.setTcpMaxLinearVelocity","params":[0.5],"id":1}
06486     *
06487     * @par JSON-RPC response example
```

```
06488     * {"id":1,"jsonrpc":"2.0","result":0}
06489     * \endenglish
06490     */
06491    int setTcpMaxLinearVelocity(double v);
06492
06493    /**
06494     * @ingroup MotionControl
06495     * \chinese
06496     *  TCP
06497     *
06498     * @return  TCP      m/s /
06499     *
06500     * @throws arcs::common_interface::AuboException
06501     *
06502     * @par Lua
06503     * getTcpMaxLinearVelocity() -> number
06504     *
06505     * @par Lua
06506     * local v = getTcpMaxLinearVelocity()  --  TCP
06507     *
06508     * @par Python
06509     * getTcpMaxLinearVelocity() -> float
06510     *
06511     * @par Python
06512     * v = getTcpMaxLinearVelocity()  #  TCP
06513     *
06514     * @par JSON-RPC
06515     * {"jsonrpc":"2.0","method":"rob1.MotionControl.getTcpMaxLinearVelocity","params":[],"id":1}
06516     *
06517     * @par JSON-RPC
06518     * {"id":1,"jsonrpc":"2.0","result":0.5}
06519     * \endchinese
06520     * \english
06521     * Get TCP maximum linear velocity
06522     *
06523     * @return Current TCP maximum linear velocity value, unit is m/s
06524     *
06525     * @throws arcs::common_interface::AuboException
06526     *
06527     * @par Lua function prototype
06528     * getTcpMaxLinearVelocity() -> number
06529     *
06530     * @par Lua example
06531     * local v = getTcpMaxLinearVelocity()  -- Get TCP maximum linear velocity
06532     *
06533     * @par Python function prototype
06534     * getTcpMaxLinearVelocity() -> float
06535     *
06536     * @par Python example
06537     * v = getTcpMaxLinearVelocity()  # Get TCP maximum linear velocity
06538     *
06539     * @par JSON-RPC request example
06540     * {"jsonrpc":"2.0","method":"rob1.MotionControl.getTcpMaxLinearVelocity","params":[],"id":1}
06541     *
06542     * @par JSON-RPC response example
06543     * {"id":1,"jsonrpc":"2.0","result":0.5}
06544     * \endenglish
06545     */
06546    double getTcpMaxLinearVelocity();
06547
06548    /**
06549     * @ingroup MotionControl
06550     * \chinese
06551     *   TCP
06552     *
06553     *     setTcpMaxLinearVelocity      TCP
06554     *
06555     * @return     0
06556     *
06557     * @throws arcs::common_interface::AuboException
06558     *
06559     * @par Lua
06560     * resetTcpMaxLinearVelocity() -> number
06561     *
06562     * @par Lua
06563     * resetTcpMaxLinearVelocity()  --    TCP
06564     *
06565     * @par Python
06566     * resetTcpMaxLinearVelocity() -> int
06567     *
06568     * @par Python
06569     * resetTcpMaxLinearVelocity()  --    TCP
06570     *
06571     * @par JSON-RPC
06572     * {"jsonrpc":"2.0","method":"rob1.MotionControl.resetTcpMaxLinearVelocity","params":[],"id":1}
06573     *
06574     * @par JSON-RPC
```

```
06575     * {"id":1,"jsonrpc":"2.0","result":0}
06576     * \endchinese
06577     *
06578     * \english
06579     * Reset TCP maximum linear velocity
06580     *
06581     * Clear the TCP maximum linear velocity override set by
06582     * setTcpMaxLinearVelocity
06583     *
06584     * @return Returns 0 on success; error code on failure.
06585     *
06586     * @throws arcs::common_interface::AuboException
06587     *
06588     * @par Lua function prototype
06589     * resetTcpMaxLinearVelocity() -> number
06590     *
06591     * @par Lua example
06592     * resetTcpMaxLinearVelocity()  -- Reset TCP maximum linear velocity to
06593     * the value in the safety configuration
06594     *
06595     * @par Python function prototype
06596     * resetTcpMaxLinearVelocity() -> int
06597     *
06598     * @par Python example
06599     * resetTcpMaxLinearVelocity()  -- Reset TCP maximum linear velocity
06600     *
06601     * @par JSON-RPC request example
06602     * {"jsonrpc":"2.0","method":"rob1.MotionControl.resetTcpMaxLinearVelocity","params":[],"id":1}
06603     *
06604     * @par JSON-RPC response example
06605     * {"id":1,"jsonrpc":"2.0","result":0}
06606     * \endenglish
06607     */
06608    int resetTcpMaxLinearVelocity();
06609
06610    /**
06611     * @ingroup MotionControl
06612     * \chinese
06613     *
06614     *
06615     *                    blending
06616     *
06617     *
06618     *
06619     *
06620     *
06621     * @return     0
06622     *
06623     * @throws arcs::common_interface::AuboException
06624     *
06625     * @par Lua
06626     * setEndPath() -> number
06627     *
06628     * @par Lua
06629     * setEndPath()  --
06630     *
06631     * @par Python
06632     * setEndPath() -> int
06633     *
06634     * @par Python
06635     * setEndPath()  #
06636     *
06637     * @par JSON-RPC
06638     * {"jsonrpc":"2.0","method":"rob1.MotionControl.setEndPath","params":[],"id":1}
06639     *
06640     * @par JSON-RPC
06641     * {"id":1,"jsonrpc":"2.0","result":0}
06642     * \endchinese
06643     *
06644     * \english
06645     * Set end path (terminate blending at current trajectory segment)
06646     *
06647     * Explicitly specifies the currently executing trajectory segment as the
06648     * boundary at which trajectory blending is terminated. The current
06649     * trajectory segment will be executed completely, but no velocity or
06650     * position blending will be applied with subsequent trajectory segments.
06651     *
06652     * This interface does not trigger an emergency stop and does not interrupt
06653     * the execution of the current trajectory segment. It only affects the
06654     * blending behavior between trajectory segments.
06655     *
06656     * @return Returns 0 on success; error code on failure.
06657     *
06658     * @throws arcs::common_interface::AuboException
06659     *
06660     * @par Lua function prototype
06661     * setEndPath() -> number
```

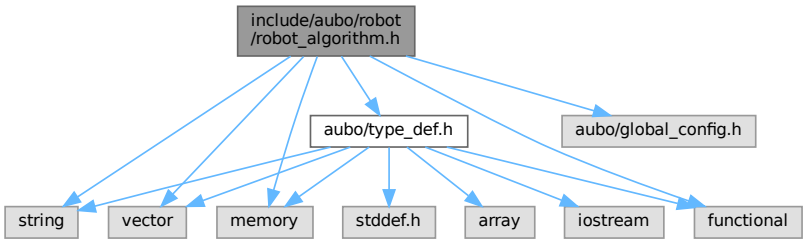
```
06662     *
06663     * @par Lua example
06664     * setEndPath()  -- Terminate trajectory blending at the current segment
06665     *
06666     * @par Python function prototype
06667     * setEndPath() -> int
06668     *
06669     * @par Python example
06670     * setEndPath()  # Terminate trajectory blending at the current segment
06671     *
06672     * @par JSON-RPC request example
06673     * {"jsonrpc":"2.0","method":"rob1.MotionControl.setEndPath","params":[],"id":1}
06674     *
06675     * @par JSON-RPC response example
06676     * {"id":1,"jsonrpc":"2.0","result":0}
06677     * \endenglish
06678     */
06679    int setEndPath();
06680
06681    /**
06682     * @ingroup MotionControl
06683     * \chinese
06684     *
06685     *
06686     *            Weave
06687     *           Y
06688     *
06689     *
06690     *
06691     *
06692     *              0
06693     *
06694     *
06695     *
06696     * @return          WeaveDirection
06697     * - 0 : NONE
06698     * - 1 : POSITIVE          Y
06699     * - 2 : NEGATIVE          Y
06700     *
06701     * @throws arcs::common_interface::AuboException
06702     *
06703     * @par Lua
06704     * getWeaveDirection() -> number
06705     *
06706     * @par Lua
06707     * local dir = getWeaveDirection()
06708     * if dir == 1 then
06709     *     --
06710     * elseif dir == 2 then
06711     *     --
06712     * end
06713     *
06714     * @par Python
06715     * getWeaveDirection() -> int
06716     *
06717     * @par Python
06718     * dir = getWeaveDirection()
06719     * if dir == 1:
06720     *     # positive direction
06721     *     pass
06722     * elif dir == 2:
06723     *     # negative direction
06724     *     pass
06725     *
06726     * @par JSON-RPC
06727     * {"jsonrpc":"2.0","method":"rob1.MotionControl.getWeaveDirection","params":[],"id":1}
06728     *
06729     * @par JSON-RPC
06730     * {"id":1,"jsonrpc":"2.0","result":1}
06731     *
06732     * \endchinese
06733     *
06734     * \english
06735     * Get current weave trajectory direction
06736     *
06737     * Retrieves the directional state of the current trajectory point during
06738     * weave motion, indicating whether the point is located on the positive or
06739     * negative side of the weave coordinate system Y-axis.
06740     *
06741     * When no weave trajectory is active, this interface returns 0.
06742     *
06743     * This is a state query interface and does not affect trajectory execution
06744     * or robot motion.
06745     *
06746     * @return Returns current weave direction corresponding to WeaveDirection
06747     * enum:
06748     * - 0 : NONE      No active weave trajectory
```

```
06749     * - 1 : POSITIVE  Positive Y-axis side
06750     * - 2 : NEGATIVE  Negative Y-axis side
06751     *
06752     * @throws arcs::common_interface::AuboException
06753     *
06754     * @par Lua function prototype
06755     * getWeaveDirection() -> number
06756     *
06757     * @par Lua example
06758     * local dir = getWeaveDirection()
06759     *
06760     * @par Python function prototype
06761     * getWeaveDirection() -> int
06762     *
06763     * @par Python example
06764     * dir = getWeaveDirection()
06765     *
06766     * @par JSON-RPC request example
06767     * {"jsonrpc":"2.0","method":"rob1.MotionControl.getWeaveDirection","params":[],"id":1}
06768     *
06769     * @par JSON-RPC response example
06770     * {"id":1,"jsonrpc":"2.0","result":1}
06771     *
06772     * \endenglish
06773     */
06774     int getWeaveDirection();
06775
06776 protected:
06777     void *d_;
06778 };
06779 using MotionControlPtr = std::shared_ptr<MotionControl>;
06780 } // namespace common_interface
06781 } // namespace arcs
06782
06783 #endif // AUBO_SDK_MOTION_CONTROL_INTERFACE_H
```

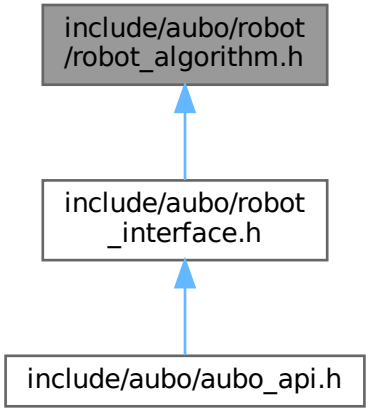
12.26 include/aubo/robot/robot_algorithm.h File Reference

```
#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     ResultWithErrno 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":[[0.7137448715395925,0.08416057568819092,0.6707994191515292,2.4599818776908724,0.4789772388601265,1.6189
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     *
{"jsonrpc":"2.0","method":"rob1.RobotAlgorithm.forwardKinematics1","params":[[3.083688522170976,1.2273215976885394,1.098072739631141,0
1.870715392248607,-0.39708546603119627],[0.0,
00593     * 0.13201,0.03879,0,0,0]],"id":1}
00594     *
00595     * @par JSON-RPC response example
00596     *
{"id":1,"jsonrpc":"2.0","result":[[0.7137636726659518,0.0837705432006433,0.6710022027216355,2.459981877690872,0.4789772388601267,1.618963
00597     *
00598     * @since 0.24.1
00599     * \endenglish
00600     */
00601    ResultWithErrno forwardKinematics1(const std::vector<double> &q,
00602                                       const std::vector<double> &tcp_offset);
00603
00604    /**
00605     * @ingroup RobotAlgorithm
00606     * \chinese
00607     *    (   TCP    )
00608     *
00609     * @param q
00610     * @return
00611     *
00612     * 0 -
00613     * -1 -      (          )
00614     * -5 -       (   )
00615     *
00616     * @throws arcs::common_interface::AuboException
00617     *
00618     * @par Lua
00619     * forwardToolKinematics(q: table) -> table, number
00620     *
00621     * @par Lua
00622     * pose, fk_result = forwardToolKinematics({3.083,1.227,1.098,0.670,-1.870,-0.397})
00623     *
00624     * @par JSON-RPC
00625     *
{"jsonrpc":"2.0","method":"rob1.RobotAlgorithm.forwardToolKinematics","params":[[3.083688522170976,1.2273215976885394,1.0980727396311
1.870715392248607,-0.39708546603119627]],"id":1}
00626     *
00627     * @par JSON-RPC
00628     *
{"id":1,"jsonrpc":"2.0","result":[[0.5881351149440136,0.05323734739426938,0.623922550656701,2.4599818776908724,0.4789772388601265,1.61896
00629     * \endchinese
00630     * \english
00631     * Forward kinematics (ignoring TCP offset)
00632     *
00633     * @param q Joint angles
00634     * @return Flange center pose and whether the result is valid
00635     * The first parameter of the return value is the forward kinematics result, the second is the error code. Error codes are
as follows:
00636     * 0 - Success
00637     * -1 - The status of the robot is incorrect (not initialized yet, you can try calling it again)
00638     * -5 - The input joint angles is invalid (dimension error)
00639     *
00640     * @par Lua function prototype
00641     * forwardToolKinematics(q: table) -> table, number
00642     *
00643     * @par Lua example
00644     * pose, fk_result = forwardToolKinematics({3.083,1.227,1.098,0.670,-1.870,-0.397})
00645     *
00646     * @throws arcs::common_interface::AuboException
00647     *
00648     * @par JSON-RPC request example
00649     *
{"jsonrpc":"2.0","method":"rob1.RobotAlgorithm.forwardToolKinematics","params":[[3.083688522170976,1.2273215976885394,1.0980727396311
1.870715392248607,-0.39708546603119627]],"id":1}
00650     *
00651     * @par JSON-RPC response example
00652     *
{"id":1,"jsonrpc":"2.0","result":[[0.5881351149440136,0.05323734739426938,0.623922550656701,2.4599818776908724,0.4789772388601265,1.61896
00653     * \endenglish
00654     */
00655    ResultWithErrno forwardToolKinematics(const std::vector<double> &q);
00656
00657    /**
00658     * @ingroup RobotAlgorithm
00659     * \chinese
00660     *    ,     TCP  (     setTcpOffset     )
00661     *
00662     *
00663     * @param q
00664     * @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 -      (          )
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],[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],[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     *
00875     *   qnear        TCP             qnear
00876     *     TCP   (    setTcpOffset    )
00877     *
00878     * @param qnear
00879     * @param pose TCP       TCP   (    setTcpOffset    )
00880     * @return
00881     *
00882     * 0 -
00883     * -1 -      (          )
00884     * -5 -       TCP   (   )
00885     * -23 -
00886     * -24 -
00887     * -25 -
00888     * -26 -
00889     * -27 -
00890     * -28 -
00891     *    0,            qnear
00892     * @throws arcs::common_interface::AuboException
00893     *
00894     * @par Python
00895     * inverseKinematics2(self: pyaubo_sdk.RobotAlgorithm, arg0: List[float],
00896     * arg1: List[float]) -> Tuple[List[float], int]
00897     *
00898     * @par Lua
00899     * inverseKinematics2(qnear: table, pose: table) -> table, int
```

```
00900     *
00901     * @par Lua
00902     * joint,ik_result = inverseKinematics2({0,0,0,0,0,0},{0.81665,-0.20419,0.43873,-3.135,0.004,1.569})
00903     *
00904     * @par JSON-RPC
00905     * {"json-
rpc":"2.0","method":"rob1.RobotAlgorithm.inverseKinematics2","params":[[0,0,0,0,0,0],[0.71374,0.08417,0.6708,2.46,0.479,1.619]],"id":1}
00906     *
00907     * @par JSON-RPC
00908     * {"id":1,"jsonrpc":"2.0","result":[[3.083688522170976,1.2273215976885394,1.098072739631141,0.6705738810610149,-
1.870715392248607,-0.39708546603119627],0]}
00909     * \endchinese
00910     * \english
00911     * Inverse kinematics with the same robot configuration and minimum joint change
00912     * Calculate the inverse kinematics solution for the target TCP pose according to the robot configuration of qnear, and
select the solution with the minimum joint change relative to qnear among solutions with the same configuration.
00913     * This interface is based on the activated TCP offset, which is the latest parameter set by setTcpOffset.
00914     *
00915     * @param qnear Reference joint angles
00916     * @param pose TCP pose, based on the activated TCP offset, which is the latest parameter set by setTcpOffset
00917     * @return Joint angles and whether the result is valid
00918     * The first parameter of the return value is the inverse kinematics result, the second is the error code. Error codes are
as follows:
00919     * 0 - Success
00920     * -1 - The status of the robot is incorrect (not initialized yet, you can try calling it again)
00921     * -5 - The input reference joint angles or tcp pose is invalid (dimension error)
00922     * -23 - Inverse kinematics calculation does not converge, calculation error
00923     * -24 - Inverse kinematics calculation exceeds robot limits
00924     * -25 - Inverse kinematics input configuration error
00925     * -26 - Inverse kinematics Jacobian calculation failed
00926     * -27 - Analytical solution exists but none satisfy the same-configuration minimum-joint-change selection criteria
00927     * -28 - Unknown error in inverse kinematics
00928     * If the error code is not 0, the first parameter of the return value is the input reference joint angles qnear
00929     * @throws arcs::common_interface::AuboException
00930     *
00931     * @par Python function prototype
00932     * inverseKinematics2(self: pyaubo_sdk.RobotAlgorithm, arg0: List[float],
00933     * arg1: List[float]) -> Tuple[List[float], int]
00934     *
00935     * @par Lua function prototype
00936     * inverseKinematics2(qnear: table, pose: table) -> table, int
00937     *
00938     * @par Lua example
00939     * joint,ik_result = inverseKinematics2({0,0,0,0,0,0},{0.81665,-0.20419,0.43873,-3.135,0.004,1.569})
00940     *
00941     * @par JSON-RPC request example
00942     * {"json-
rpc":"2.0","method":"rob1.RobotAlgorithm.inverseKinematics2","params":[[0,0,0,0,0,0],[0.71374,0.08417,0.6708,2.46,0.479,1.619]],"id":1}
00943     *
00944     * @par JSON-RPC response example
00945     * {"id":1,"jsonrpc":"2.0","result":[[3.083688522170976,1.2273215976885394,1.098072739631141,0.6705738810610149,-
1.870715392248607,-0.39708546603119627],0]}
00946     * \endenglish
00947     */
00948    ResultWithErrno inverseKinematics2(const std::vector<double> &qnear,
00949                                       const std::vector<double> &pose);
00950
00951    /**
00952     * @ingroup RobotAlgorithm
00953     * \chinese
00954     *      ,      TCP
00955     *
00956     * @param pose TCP
00957     * @return
00958     *      inverseKinematics
00959     *
00960     * @throws arcs::common_interface::AuboException
00961     *
00962     * @par JSON-RPC
00963     * {"json-
rpc":"2.0","method":"rob1.RobotAlgorithm.inverseKinematicsAll","params":[[0.71374,0.08417,0.6708,2.46,0.479,1.619]],"id":1}
00964     *
00965     * @par JSON-RPC
00966     * {"id":1,"jsonrpc":"2.0","result":[[[3.083688522170976,1.2273215976885394,1.098072739631141,0.6705738810610149,-
1.870715392248607,-0.39708546603119627],
00967     * [3.081056801097411,0.17985038037652645,-1.0991717292664145,-0.4806460200109001,-1.869182975312333,-
0.402066016835411],
00968     * [0.4090095277807992,-
0.1623365054641728,1.081775890307679,0.26993250263224805,0.9738255833642309,0.000572556627720845],
00969     * [0.4116449425067969,-1.1931664523907126,-1.0822709833775688,-
0.8665964106161371,0.9732141569888207,0.006484919654891586]],0]}
00970     * \endchinese
00971     * \english
00972     * Solve all inverse kinematics solutions based on the activated TCP offset
00973     *
00974     * @param pose TCP pose
00975     * @return Joint angles and whether the result is valid
```

```
00976     * The returned error code is the same as inverseKinematics
00977     *
00978     * @throws arcs::common_interface::AuboException
00979     *
00980     * @par JSON-RPC request example
00981     * {"jsonrpc":"2.0","method":"rob1.RobotAlgorithm.inverseKinematicsAll","params":[[0.71374,0.08417,0.6708,2.46,0.479,1.619]],"id":1}
00982     *
00983     * @par JSON-RPC response example
00984     * {"id":1,"jsonrpc":"2.0","result":[[[3.083688522170976,1.2273215976885394,1.098072739631141,0.6705738810610149,-1.870715392248607,-0.39708546603119627],
00985     * [3.081056801097411,0.17985038037652645,-1.0991717292664145,-0.4806460200109001,-1.869182975312333,-0.402066016835411],
00986     * [0.4090095277807992,-0.1623365054641728,1.081775890307679,0.26993250263224805,0.9738255833642309,0.000572556627720845],
00987     * [0.4116449425067969,-1.1931664523907126,-1.0822709833775688,-0.8665964106161371,0.9732141569888207,0.006484919654891586]],0]}
00988     * \endenglish
00989     */
00990    ResultWithErrno1 inverseKinematicsAll(const std::vector<double> &pose);
00991
00992    /**
00993     * @ingroup RobotAlgorithm
00994     * \chinese
00995     *      ,      TCP
00996     *
00997     * @param pose TCP
00998     * @param tcp_offset TCP
00999     * @return              inverseKinematicsAll
01000     *      inverseKinematics
01001     *
01002     * @throws arcs::common_interface::AuboException
01003     *
01004     * @par JSON-RPC
01005     * {"jsonrpc":"2.0","method":"rob1.RobotAlgorithm.inverseKinematicsAll1","params":[[0.71374,0.08417,0.6708,2.46,0.479,1.619],[0.0,
01006     * 0.13201,0.03879,0,0,0]],"id":1}
01007     *
01008     * @par JSON-RPC
01009     * {"id":1,"jsonrpc":"2.0","result":[[[3.084454549595208,1.2278265883747776,1.0986586440159576,0.6708221281915528,-1.8712459848518375,-0.3965111476861782],
01010     * [3.0818224058231602,0.17980369843203092,-1.0997576631122077,-0.48102131527371267,-1.8697135490338517,-0.40149459722060593],
01011     * [0.40972960018231047,-0.16226026285489026,1.0823403816496,0.2700204411869427,0.9734251963887868,0.0012903686498106507],
01012     * [0.41236549588802296,-1.193621392918341,-1.0828346680836718,-0.8671097369314354,0.972815367289568,0.007206851371073478]],0]}
01013     * \endchinese
01014     * \english
01015     * Solve all inverse kinematics solutions based on the provided TCP offset
01016     *
01017     * @param pose TCP pose
01018     * @param tcp_offset TCP offset
01019     * @return Joint angles and whether the result is valid, same as inverseKinematicsAll
01020     * The returned error code is the same as inverseKinematics
01021     *
01022     * @throws arcs::common_interface::AuboException
01023     *
01024     * @par JSON-RPC request example
01025     * {"jsonrpc":"2.0","method":"rob1.RobotAlgorithm.inverseKinematicsAll1","params":[[0.71374,0.08417,0.6708,2.46,0.479,1.619],[0.0,
01026     * 0.13201,0.03879,0,0,0]],"id":1}
01027     *
01028     * @par JSON-RPC response example
01029     * {"id":1,"jsonrpc":"2.0","result":[[[3.084454549595208,1.2278265883747776,1.0986586440159576,0.6708221281915528,-1.8712459848518375,-0.3965111476861782],
01030     * [3.0818224058231602,0.17980369843203092,-1.0997576631122077,-0.48102131527371267,-1.8697135490338517,-0.40149459722060593],
01031     * [0.40972960018231047,-0.16226026285489026,1.0823403816496,0.2700204411869427,0.9734251963887868,0.0012903686498106507],
01032     * [0.41236549588802296,-1.193621392918341,-1.0828346680836718,-0.8671097369314354,0.972815367289568,0.007206851371073478]],0]}
01033     * \endenglish
01034     */
01035    ResultWithErrno1 inverseKinematicsAll1(
01036        const std::vector<double> &pose, const std::vector<double> &tcp_offset);
01037
01038    /**
01039     * @ingroup RobotAlgorithm
01040     * \chinese
01041     *    (   TCP    )
01042     *
01043     * @param qnear
01044     * @param pose
01045     * @return
01046     *
01047     * 0 -
```

```
01048     * -1 -      (          )
01049     * -5 -          (   )
01050     *
01051     * @throws arcs::common_interface::AuboException
01052     *
01053     * @par Lua
01054     * inverseToolKinematics(qnear: table, pose: table) -> table, int
01055     *
01056     * @par Lua
01057     * joint, ik_result = inverseToolKinematics({0,0,0,0,0,0},{0.58815,0.0532,0.62391,2.46,0.479,1.619})
01058     *
01059     * @par JSON-RPC
01060     * {"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}
01061     *
01062     * @par JSON-RPC
01063     * {"id":1,"jsonrpc":"2.0","result":[[3.083609363838651,1.22736129158332,1.098095443698268,0.6705395395487186,-1.8706605026855632,-0.39714507002376465],0]}
01064     * \endchinese
01065     * \english
01066     * Inverse kinematics (ignoring TCP offset)
01067     *
01068     * @param qnear Reference joint angles
01069     * @param pose Flange center pose
01070     * @return Joint angles and whether the result is valid
01071     * The first parameter of the return value is the inverse kinematics result, the second is the error code. Error codes are as follows:
01072     * 0 - Success
01073     * -1 - The status of the robot is incorrect (not initialized yet, you can try calling it again)
01074     * -5 - The input reference joint angles or pose is invalid (dimension error)
01075     * @throws arcs::common_interface::AuboException
01076     *
01077     * @par Lua function prototype
01078     * inverseToolKinematics(qnear: table, pose: table) -> table, int
01079     *
01080     * @par Lua example
01081     * joint, ik_result = inverseToolKinematics({0,0,0,0,0,0},{0.58815,0.0532,0.62391,2.46,0.479,1.619})
01082     *
01083     * @par JSON-RPC request example
01084     * {"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}
01085     *
01086     * @par JSON-RPC response example
01087     * {"id":1,"jsonrpc":"2.0","result":[[3.083609363838651,1.22736129158332,1.098095443698268,0.6705395395487186,-1.8706605026855632,-0.39714507002376465],0]}
01088     * \endenglish
01089     */
01090    ResultWithErrno inverseToolKinematics(const std::vector<double> &qnear,
01091                                          const std::vector<double> &pose);
01092
01093    /**
01094     * @ingroup RobotAlgorithm
01095     * \chinese
01096     *     (   TCP    )
01097     *
01098     * @param qnear
01099     * @param pose
01100     * @return
01101     *
01102     * @throws arcs::common_interface::AuboException
01103     *
01104     * @par JSON-RPC
01105     * {"jsonrpc":"2.0","method":"rob1.RobotAlgorithm.inverseToolKinematicsAll","params":[[0.58815,0.0532,0.62391,2.46,0.479,1.619]],"id":1}
01106     *
01107     * @par JSON-RPC
01108     * {"id":1,"jsonrpc":"2.0","result":[[[3.083609363838651,1.22736129158332,1.098095443698268,0.6705395395487186,-1.8706605026855632,-0.39714507002376465],
01109     * [3.0809781797426523,0.17987122696706134,-1.0991932793263717,-0.4807053707530958,-1.8691282890274434,-0.40212516672751814],
01110     * [0.40892195618737215,-0.16235398607358653,1.081812753177426,0.27003586475871766,0.9738744130114284,0.00048462518316674287],
01111     * [0.41155633414333076,-1.1932173012004512,-1.082306542045813,-0.8665312056504818,0.9732632365861417,0.006395831160177175]],0]}
01112     * \endchinese
01113     * \english
01114     * Inverse kinematics (ignoring TCP offset)
01115     *
01116     * @param qnear Reference joint angles
01117     * @param pose Flange center pose
01118     * @return Joint angles and whether the result is valid
01119     *
01120     * @throws arcs::common_interface::AuboException
01121     *
01122     * @par JSON-RPC request example
01123     * {"jsonrpc":"2.0","method":"rob1.RobotAlgorithm.inverseToolKinematicsAll","params":[[0.58815,0.0532,0.62391,2.46,0.479,1.619]],"id":1}
```

```
01124     *
01125     * @par JSON-RPC response example
01126     * {"id":1,"jsonrpc":"2.0","result":[[[3.083609363838651,1.22736129158332,1.098095443698268,0.6705395395487186,-
1.8706605026855632,-0.39714507002376465],
01127     * [3.0809781797426523,0.17987122696706134,-1.0991932793263717,-0.4807053707530958,-1.8691282890274434,-
0.40212516672751814],
01128     * [0.40892195618737215,-
0.16235398607358653,1.081812753177426,0.27003586475871766,0.9738744130114284,0.00048462518316674287],
01129     * [0.41155633414333076,-1.1932173012004512,-1.082306542045813,-
0.8665312056504818,0.9732632365861417,0.006395831160177175]],0]}
01130     * \endenglish
01131     */
01132    ResultWithErrno1 inverseToolKinematicsAll(const std::vector<double> &pose);
01133
01134    /**
01135     * @ingroup RobotAlgorithm
01136     * \chinese
01137     *
01138     *
01139     *
01140     * -    LEFT(   ) / RIGHT(   )
01141     * -    UP(   ) / DOWN(   )
01142     * -    FLIP(   ) / NOFLIP(    )
01143     *
01144     *       8    L/R + U/D + F/N
01145     *                    LUF  0    LEFT/UP/FLIP
01146     *
01147     * @param q       6    6      (rad)
01148     * @return           ResultWithErrno3  std::tuple<int, int>
01149     *        -   int      (RobotConfiguration)
01150     *          -1 (NONE)   -
01151     *           0 (LUF)    - LEFT+UP+FLIP
01152     *           1 (LUN)    - LEFT+UP+NOFLIP
01153     *           2 (LDF)    - LEFT+DOWN+FLIP
01154     *           3 (LDN)    - LEFT+DOWN+NOFLIP
01155     *           4 (RUF)    - RIGHT+UP+FLIP
01156     *           5 (RUN)    - RIGHT+UP+NOFLIP
01157     *           6 (RDF)    - RIGHT+DOWN+FLIP
01158     *           7 (RDN)    - RIGHT+DOWN+NOFLIP
01159     *        -   int
01160     *          0  -
01161     *         -1  -
01162     *         -5  -             6
01163     *
01164     * @throws arcs::common_interface::AuboException
01165     *
01166     * @par Python
01167     * getRobotConfiguration(self: pyaubo_sdk.RobotAlgorithm, arg0: List[float]) -> Tuple[int, int]
01168     *
01169     * @par Lua
01170     * getRobotConfiguration(q: table) -> number, number
01171     *
01172     * @par Lua
01173     * --   6      rad
01174     * local joint_angles = {3.083,1.227,1.098,0.670,-1.870,-0.397}
01175     * local config_val, err_code = getRobotConfiguration(joint_angles)
01176     * if err_code == 0 then
01177     *     print("      ", config_val)  --     4   RUF
01178     * else
01179     *     print("        ", err_code)
01180     * end
01181     *
01182     * @par JSON-RPC
01183     *
{"jsonrpc":"2.0","method":"rob1.RobotAlgorithm.getRobotConfiguration","params":[[3.083688522170976,1.2273215976885394,1.0980727396311
1.870715392248607,-0.39708546603119627]],"id":1}
01184     *
01185     * @par JSON-RPC
01186     * {"id":1,"jsonrpc":"2.0","result":[4,0]}
01187     * \endchinese
01188     * \english
01189     * Calculate and return the corresponding robot configuration based on input joint angles
01190     *
01191     * The robot configuration consists of three dimensions of states, with the following definitions for each dimension:
01192     * - Shoulder direction: LEFT (Shoulder to the left) / RIGHT (Shoulder to the right)
01193     * - Elbow direction: UP (Elbow up) / DOWN (Elbow down)
01194     * - Wrist state: FLIP (Wrist flipped) / NOFLIP (Wrist not flipped)
01195     *
01196     * The combination of the three dimensions forms 8 basic configurations (L/R + U/D + F/N). This interface parses the
current
01197     * configuration type from the input joint angles and returns its integer value corresponding to the RobotConfiguration
enumeration
01198     * and an error code (Note: The interface returns combined configuration values, e.g., LUF corresponds to 0, not the
individual
01199     * enumeration values of LEFT/UP/FLIP).
01200     *
01201     * @param q Input joint angle array, 6 elements for 6-axis robot, unit: radians (rad)
```

```
01202     * @return Robot configuration result and error code (type: ResultWithErrno3, i.e., std::tuple<int, int>):
01203     *         - First int: Integer value corresponding to the RobotConfiguration enumeration, value range and meanings:
01204     *           -1 (NONE)   - Invalid configuration
01205     *            0 (LUF)    - LEFT+UP+FLIP (Left shoulder, elbow up, wrist flipped)
01206     *            1 (LUN)    - LEFT+UP+NOFLIP (Left shoulder, elbow up, wrist not flipped)
01207     *            2 (LDF)    - LEFT+DOWN+FLIP (Left shoulder, elbow down, wrist flipped)
01208     *            3 (LDN)    - LEFT+DOWN+NOFLIP (Left shoulder, elbow down, wrist not flipped)
01209     *            4 (RUF)    - RIGHT+UP+FLIP (Right shoulder, elbow up, wrist flipped)
01210     *            5 (RUN)    - RIGHT+UP+NOFLIP (Right shoulder, elbow up, wrist not flipped)
01211     *            6 (RDF)    - RIGHT+DOWN+FLIP (Right shoulder, elbow down, wrist flipped)
01212     *            7 (RDN)    - RIGHT+DOWN+NOFLIP (Right shoulder, elbow down, wrist not flipped)
01213     *         - Second int: Error code with the following meanings:
01214     *           0  - Success: Configuration calculation completed, valid configuration value returned
01215     *          -1  - Abnormal robot state: Not initialized completely, try reinitializing before calling
01216     *          -5  - Invalid input parameters: Incorrect dimension of joint angle array (not 6 elements) or values out of
      reasonable range
01217     *
01218     * @throws arcs::common_interface::AuboException Thrown when input parameters are illegal (e.g., empty array, wrong
      number of elements)
01219     *
01220     * @par Python function prototype
01221     * getRobotConfiguration(self: pyaubo_sdk.RobotAlgorithm, arg0: List[float]) -> Tuple[int, int]
01222     *
01223     * @par Lua function prototype
01224     * getRobotConfiguration(q: table) -> number, number
01225     *
01226     * @par Lua example
01227     * -- Input 6-axis joint angles (unit: rad) to get configuration and error code
01228     * local joint_angles = {3.083,1.227,1.098,0.670,-1.870,-0.397}
01229     * local config_val, err_code = getRobotConfiguration(joint_angles)
01230     * if err_code == 0 then
01231     *     print("Robot configuration value: ", config_val)  -- Example output: 4 (corresponding to RUF, Right shoulder,
      elbow up, wrist flipped)
01232     * else
01233     *     print("Failed to get configuration, error code: ", err_code)
01234     * end
01235     *
01236     * @par JSON-RPC request example
01237     *
      {"jsonrpc":"2.0","method":"rob1.RobotAlgorithm.getRobotConfiguration","params":[[3.083688522170976,1.2273215976885394,1.0980727396311
      1.870715392248607,-0.39708546603119627]],"id":1}
01238     *
01239     * @par JSON-RPC response example
01240     * {"id":1,"jsonrpc":"2.0","result":[4,0]}
01241     * \endenglish
01242     */
01243    ResultWithErrno3 getRobotConfiguration(const std::vector<double>& q);
01244
01245    /**
01246     * \chinese
01247     *  movej
01248     *
01249     * @param q1 movej
01250     * @param r1  q1
01251     * @param q2 movej
01252     * @param r2  q2
01253     * @param d
01254     * @return q1~q2          (x,y,z,rx,ry,rz)
01255     *
01256     * @throws arcs::common_interface::AuboException
01257     *
01258     * @par Python
01259     * pathMovej(self: pyaubo_sdk.RobotAlgorithm, arg0: List[float], arg1:
01260     * float, arg2: List[float], arg3: float, arg4: float) -> List[List[float]]
01261     *
01262     * @par Lua
01263     * pathMovej(q1: table, r1: number, q2: table, r2: number, d: number) ->
01264     * table, number
01265     *
01266     * @par Lua
01267     * 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)
01268     *
01269     * \endchinese
01270     * \english
01271     * Solve the trajectory points between movej
01272     *
01273     * @param q1 Start point of movej
01274     * @param r1 Blend radius at q1
01275     * @param q2 End point of movej
01276     * @param r2 Blend radius at q2
01277     * @param d Sampling distance
01278     * @return Discrete trajectory points (x, y, z, rx, ry, rz) between q1 and q2 in Cartesian space
01279     *
01280     * @throws arcs::common_interface::AuboException
01281     *
01282     * @par Python function prototype
```

```
01283     * pathMovej(self: pyaubo_sdk.RobotAlgorithm, arg0: List[float], arg1:
01284     * float, arg2: List[float], arg3: float, arg4: float) -> List[List[float]]
01285     *
01286     * @par Lua function prototype
01287     * pathMovej(q1: table, r1: number, q2: table, r2: number, d: number) ->
01288     * table, number
01289     *
01290     * @par Lua example
01291     * 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)
01292     *
01293     * \endenglish
01294     */
01295    std::vector<std::vector<double» pathMovej(const std::vector<double> &q1,
01296                                              double r1,
01297                                              const std::vector<double> &q2,
01298                                              double r2, double d);
01299    /**
01300     * @ingroup RobotAlgorithm
01301     * \chinese
01302     *
01303     *
01304     * @param q
01305     * @param base_or_end
01306     *        true:   base
01307     *        false:
01308     * @return
01309     *
01310     *         0.28.1-rc.21 0.29.0-alpha.25
01311     *        30082
01312     * 0 -
01313     * -23 -
01314     * -24 -
01315     * -25 -
01316     * -26 -
01317     * -27 -
01318     * -28 -
01319     *    0,             qnear
01320     *
01321     * @throws arcs::common_interface::AuboException
01322     *
01323     * @par Python
01324     * calJacobian(self: pyaubo_sdk.RobotAlgorithm, arg0: List[float],
01325     * arg1: bool) -> Tuple[List[float], int]
01326     *
01327     * @par Lua
01328     * calJacobian(q: table, base_or_end: boolean) -> table
01329     *
01330     * @par Lua
01331     * calJ_result = calJacobian({0.58815,0.0532,0.62391,2.46,0.479,1.619},true)
01332     *
01333     * @par JSON-RPC
01334     * {"json-
rpc":"2.0","method":"rob1.RobotAlgorithm.calcJacobian","params":[[0.58815,0.0532,0.62391,2.46,0.479,1.619],true],"id":1}
01335     *
01336     * @par JSON-RPC
01337     * {"id":1,"jsonrpc":"2.0","result":[[0.20822779551242535,-
0.5409416184208162,0.2019786999613013,0.061264982268770196,-0.026269884327316487,
01338     * 0.10131708699859962,0.26388933410019777,-0.36074292664199115,0.1346954733416397,0.04085636647597124,-
0.07244204452918337,0.0708466286633346,
01339     * 0.0,0.10401808481666497,-0.12571344758923886,-
0.07741290545882097,0.18818543519232858,0.04628646442706299,0.0,0.5548228314607867,
01340     * -0.5548228314607868,0.5548228314607868,-0.7901273140338193,0.37230961532208007,0.0,-
0.8319685244586092,0.8319685244586091,-0.8319685244586091,
01341     * -0.5269197820578843,-0.8184088260676008,1.0,3.749399456654644e-33,-6.512048180336603e-18,1.0956823467534067e-
16,-0.31313634553301894,
01342     * 0.43771285536682175],0]}
01343     * \endchinese
01344     * \english
01345     * Calculate the Jacobian matrix at the robot end-effector
01346     *
01347     * @param q Joint angles
01348     * @param base_or_end Reference frame: base (or end-effector)
01349     *        true: described in base frame
01350     *        false: described in end-effector frame
01351     * @return Whether the Jacobian matrix is valid
01352     * The first parameter of the return value is the Jacobian matrix for this configuration, the second is the error code.
01353     * The error code list was updated after versions 0.28.1-rc.21 and 0.29.0-alpha.25.
01354     * Previously, inverse kinematics errors returned 30082. The updated error codes are:
01355     * 0 - Success
01356     * -23 - Inverse kinematics calculation does not converge, calculation error
01357     * -24 - Inverse kinematics calculation exceeds robot limits
01358     * -25 - Inverse kinematics input configuration error
01359     * -26 - Inverse kinematics Jacobian calculation failed
01360     * -27 - Analytical solution exists but none satisfy the selection criteria
01361     * -28 - Unknown error in inverse kinematics
01362     * If the error code is not 0, the first parameter of the return value is the input reference joint angles qnear
```

```
01363     *
01364     * @throws arcs::common_interface::AuboException
01365     *
01366     * @par Python function prototype
01367     * calJacobian(self: pyaubo_sdk.RobotAlgorithm, arg0: List[float],
01368     * arg1: bool) -> Tuple[List[float], int]
01369     *
01370     * @par Lua function prototype
01371     * calJacobian(q: table, base_or_end: boolean) -> table
01372     *
01373     * @par Lua example
01374     * calJ_result = calJacobian({0.58815,0.0532,0.62391,2.46,0.479,1.619},true)
01375     *
01376     * @par JSON-RPC request example
01377     * {"json-
rpc":"2.0","method":"rob1.RobotAlgorithm.calcJacobian","params":[[0.58815,0.0532,0.62391,2.46,0.479,1.619],true],"id":1}
01378     *
01379     * @par JSON-RPC response example
01380     * {"id":1,"jsonrpc":"2.0","result":[[0.20822779551242535,-
0.5409416184208162,0.2019786999613013,0.061264982268770196,-0.026269884327316487,
01381     * 0.10131708699859962,0.26388933410019777,-0.36074292664199115,0.1346954733416397,0.04085636647597124,-
0.07244204452918337,0.0708466286633346,
01382     * 0.0,0.10401808481666497,-0.12571344758923886,-
0.07741290545882097,0.18818543519232858,0.04628646442706299,0.0,0.5548228314607867,
01383     * -0.5548228314607868,0.5548228314607868,-0.7901273140338193,0.37230961532208007,0.0,-
0.8319685244586092,0.8319685244586091,-0.8319685244586091,
01384     * -0.5269197820578843,-0.8184088260676008,1.0,3.749399456654644e-33,-6.512048180336603e-18,1.0956823467534067e-
16,-0.31313634553301894,
01385     * 0.43771285536682175],0]}
01386     * \endenglish
01387     */
01388    ResultWithErrno calcJacobian(const std::vector<double> &q,
01389                                 bool base_or_end);
01390
01391    /**
01392     * @ingroup RobotAlgorithm
01393     * \chinese
01394     *
01395     *
01396     * @param type
01397     * 0-movej   movej
01398     * 1-movej   movel
01399     * 2-movel   movej
01400     * 3-movel   movel
01401     * @param q_start
01402     * @param q_via
01403     * @param q_to
01404     * @param r  q_via
01405     * @param d
01406     * @return q_via          (x,y,z)
01407     *
01408     * @throws arcs::common_interface::AuboException
01409     *
01410     * @par Python
01411     * pathBlend3Points(self: pyaubo_sdk.RobotAlgorithm, arg0: int, arg1:
01412     * List[float], arg2: List[float], arg3: List[float], arg4: float, arg5:
01413     * float) -> List[List[float]]
01414     *
01415     * @par Lua
01416     * pathBlend3Points(type: number, q_start: table, q_via: table, q_to: table,
01417     * r: number, d: number) -> table, number
01418     *
01419     * @par Lua
01420     * 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},
01421     * {0.3234,-0.5405,1.5403,0.5881,1.2962,0.7435},0.25,0.02)
01422     *
01423     * \endchinese
01424     * \english
01425     * Solve the blended trajectory points
01426     *
01427     * @param type
01428     * 0-movej and movej
01429     * 1-movej and movel
01430     * 2-movel and movej
01431     * 3-movel and movel
01432     * @param q_start Start point before blending
01433     * @param q_via Blending point
01434     * @param q_to End point after blending
01435     * @param r Blend radius at q_via
01436     * @param d Sampling distance
01437     * @return Discrete trajectory points (x, y, z) of the blend segment at q_via in Cartesian space
01438     *
01439     * @throws arcs::common_interface::AuboException
01440     *
01441     * @par Python function prototype
01442     * pathBlend3Points(self: pyaubo_sdk.RobotAlgorithm, arg0: int, arg1:
01443     * List[float], arg2: List[float], arg3: List[float], arg4: float, arg5:
```

```
01444     * float) -> List[List[float]]
01445     *
01446     * @par Lua function prototype
01447     * pathBlend3Points(type: number, q_start: table, q_via: table, q_to: table,
01448     * r: number, d: number) -> table, number
01449     *
01450     * @par Lua example
01451     * 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},
01452     * {0.3234,-0.5405,1.5403,0.5881,1.2962,0.7435},0.25,0.02)
01453     *
01454     * \endenglish
01455     */
01456    std::vector<std::vector<double» pathBlend3Points(
01457        int type, const std::vector<double> &q_start,
01458        const std::vector<double> &q_via, const std::vector<double> &q_to,
01459        double r, double d);
01460
01461    /**
01462     * @ingroup RobotAlgorithm
01463     * \chinese
01464     *
01465     *      pathBufferAppend
01466     *      buffer      movePathBuffer
01467     * @param name
01468     * @param traj_conf
01469     * traj_conf.move_axis:
01470     *                          traj_conf.move_axis=LoadIdentifyMoveAxis::Joint_4_6;
01471     * traj_conf.init_joint:
01472     *      ,     5          traj_conf.init_joint[4]      0.3(rad)   1.57(rad)
01473     * traj_conf.lower_joint_bound, traj_conf.upper_joint_bound:
01474     *     ,    config.move_axis     ,    upper_joint_bound 2 lower_joint_bound -2
01475     * config.max_velocity, config.max_acceleration:
01476     *         ,    config.move_axis     ,           ,    max_velocity=3,max_acceleration=5
01477     *
01478     * @return    0
01479     * AUBO_BUSY
01480     * AUBO_BAD_STATE
01481     * -AUBO_INVL_ARGUMENT
01482     * -AUBO_BAD_STATE
01483     *
01484     * @throws arcs::common_interface::AuboException
01485     * \endchinese
01486     * \english
01487     * Generate excitation trajectory for payload identification
01488     * This interface internally calls pathBufferAppend
01489     * The offline trajectory is stored in the buffer, and can be executed later via movePathBuffer
01490     * @param name Trajectory name
01491     * @param traj_conf Joint trajectory constraints
01492     * traj_conf.move_axis: Moving axes
01493     * 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;
01494     * traj_conf.init_joint:
01495     * Initial joint angles. To avoid singularity issues near joint 5 zero position, set abs(traj_conf.init_joint[4]) >= 0.3(rad),
    preferably close to 1.57(rad). Other joints can be set arbitrarily.
01496     * traj_conf.lower_joint_bound, traj_conf.upper_joint_bound:
01497     * Joint angle limits, dimensions should match config.move_axis. Recommended: upper_joint_bound=2,
    lower_joint_bound=-2
01498     * config.max_velocity, config.max_acceleration:
01499     * Joint velocity and acceleration limits, dimensions should match config.move_axis. For safety and driver performance,
    recommended: max_velocity=3, max_acceleration=5
01500     *
01501     * @return Returns 0 on success; error code on failure
01502     * AUBO_BUSY
01503     * AUBO_BAD_STATE
01504     * -AUBO_INVL_ARGUMENT
01505     * -AUBO_BAD_STATE
01506     *
01507     * @throws arcs::common_interface::AuboException
01508     * \endenglish
01509     */
01510    int generatePayloadIdentifyTraj(const std::string &name,
01511                                    const TrajConfig &traj_conf);
01512    /**
01513     * @ingroup RobotAlgorithm
01514     * \chinese
01515     *
01516     *
01517     * @return    0;      1;      <0;
01518     *
01519     * @throws arcs::common_interface::AuboException
01520     *
01521     * @par JSON-RPC
01522     * {"jsonrpc":"2.0","method":"rob1.RobotAlgorithm.payloadIdentifyTrajGenFinished","params":[],"id":1}
01523     *
01524     * @par JSON-RPC
01525     * {"id":1,"jsonrpc":"2.0","result":0}
01526     *
```

```
01527     * \endchinese
01528     * \english
01529     * Whether payload identification trajectory generation is finished
01530     *
01531     * @return 0 if finished; 1 if in progress; <0 if failed;
01532     *
01533     * @throws arcs::common_interface::AuboException
01534     *
01535     * @par JSON-RPC request example
01536     * {"jsonrpc":"2.0","method":"rob1.RobotAlgorithm.payloadIdentifyTrajGenFinished","params":[],"id":1}
01537     *
01538     * @par JSON-RPC response example
01539     * {"id":1,"jsonrpc":"2.0","result":0}
01540     *
01541     * \endenglish
01542     */
01543    int payloadIdentifyTrajGenFinished();
01544
01545    /**
01546     * @ingroup RobotAlgorithm
01547     * \chinese
01548     *    moveS
01549     *
01550     * @brief pathMoveS
01551     * @param qs
01552     * @param d
01553     * @return
01554     *
01555     * @throws arcs::common_interface::AuboException
01556     * \endchinese
01557     * \english
01558     * Solve the trajectory points for moveS
01559     *
01560     * @brief pathMoveS
01561     * @param qs Spline trajectory generation point set
01562     * @param d Sampling distance
01563     * @return
01564     *
01565     * @throws arcs::common_interface::AuboException
01566     * \endenglish
01567     */
01568    std::vector<std::vector<double» pathMoveS(
01569        const std::vector<std::vector<double» &qs, double d);
01570
01571    /**
01572     * @ingroup RobotAlgorithm
01573     * \chinese
01574     *
01575     *
01576     * @param q
01577     * @param qd
01578     * @param target_q
01579     * @param target_qd
01580     * @param target_qdd
01581     * @param tool_offset   TCP
01582     * @param omega
01583     * @param zeta
01584     * @return
01585     *
01586     * @throws arcs::common_interface::AuboException
01587     *
01588     * @par Python
01589     * calibVibrationParams(self: pyaubo_sdk.RobotAlgorithm, arg0:
01590     * List[List[float]], arg1: List[List[float]], arg2: List[List[float]], arg3: List[List[float]],
01591     * arg4: List[List[float]], arg5: List[float]) -> list[list[float]],int
01592     *
01593     * @par Lua
01594     * calibVibrationParams(q: table,qd: table, target_q: table, target_qd: table,
01595     * target_qdd: table, tool_offset: table, omega: table, zeta: table) -> table,number
01596     * \endchinese
01597     */
01598    ResultWithErrno1 calibVibrationParams(const std::vector<std::vector<double» &q,
01599                          const std::vector<std::vector<double» &qd,
01600                          const std::vector<std::vector<double» &target_q,
01601                          const std::vector<std::vector<double» &target_qd,
01602                          const std::vector<std::vector<double» &target_qdd,
01603                          const std::vector<double> &tool_offset);
01604
01605    /**
01606     * @ingroup RobotAlgorithm
01607     * \chinese
01608     *        1
01609     *
01610     * @param record_cache_name
01611     * @param tool_offset   TCP
01612     * @return
01613     *
```

```
01614     * @throws arcs::common_interface::AuboException
01615     *
01616     * @par Python
01617     * calibVibrationParams(self: pyaubo_sdk.RobotAlgorithm, arg0:
01618     * string, arg1: List[float]) -> list[list[float]],int
01619     *
01620     * @par Lua
01621     * calibVibrationParams(record_cache_name: string, tool_offset: table) -> table,number
01622     * \endchinese
01623     */
01624    ResultWithErrno1 calibVibrationParams1(const std::string &record_cache_name, const std::vector<double>
        &tool_offset);
01625
01626    /**
01627     * @ingroup RobotAlgorithm
01628     * \chinese
01629     *
01630     *
01631     * @param param1
01632     * @param param2
01633     * @param threshold     0~1
01634     * @return >0       =0     <0
01635     *
01636     * @par Lua
01637     * needVibrationRecalib(param1: table, param2: table, threshold: number) -> number
01638     *
01639     * @par JSON-RPC
01640     * {"jsonrpc":"2.0","method":"VibrationController.needVibrationRecalib",
01641     *  "params":[param1_obj, param2_obj, 0.1],"id":1}
01642     * \endchinese
01643     */
01644    int needVibrationRecalib(const VibrationRecalibrationParameter &param1,
01645                             const VibrationRecalibrationParameter &param2,
01646                             double threshold);
01647
01648    /**
01649     * @ingroup RobotAlgorithm
01650     * \chinese
01651     *
01652     *
01653     *
01654     *
01655     * @param name
01656     * @param link_name         link
01657     * `world` `base_link` `shoulder_Link` `upperArm_Link`
01658     * `foreArm_Link` `wrist1_Link` `wrist2_Link` `wrist3_Link`
01659     * `end_effector`
01660     * @param sizes
01661     * `[length, width, height]`    m
01662     * @param poses                 link
01663     *     `[x, y, z, rx, ry, rz]`
01664     * @return 0       0
01665     *
01666     * @par Lua
01667     * addCollisionBox(name, link_name, sizes, poses) -> integer
01668     *
01669     * @par Lua
01670     * --      world
01671     * addCollisionBox("env_box", "world",
01672     *                 {{0.4, 0.4, 0.2}},
01673     *                 {{0.6, 0.0, 0.1, 0.0, 0.0, 0.0}})
01674     *
01675     * --      end_effector
01676     * addCollisionBox("tool_box", "end_effector",
01677     *                 {{0.1, 0.08, 0.06}},
01678     *                 {{0.0, 0.0, 0.05, 0.0, 0.0, 0.0}})
01679     *
01680     * @par Python
01681     * addCollisionBox(name: str, link_name: str, sizes: list[list[float]],
01682     * poses: list[list[float]]) -> int
01683     *
01684     * @par Python
01685     * #      world
01686     * robot_algorithm.addCollisionBox(
01687     *     "env_box", "world",
01688     *     [[0.4, 0.4, 0.2]],
01689     *     [[0.6, 0.0, 0.1, 0.0, 0.0, 0.0]]
01690     * )
01691     *
01692     * #      end_effector
01693     * robot_algorithm.addCollisionBox(
01694     *     "tool_box", "end_effector",
01695     *     [[0.1, 0.08, 0.06]],
01696     *     [[0.0, 0.0, 0.05, 0.0, 0.0, 0.0]]
01697     * )
01698     *
01699     * @par JSON-RPC
```

```
01700     * {"jsonrpc":"2.0","method":"rob1.RobotAlgorithm.addCollisionBox",
01701     * "params":["env_box","world",[[0.4,0.4,0.2]],
01702     * [[0.6,0.0,0.1,0.0,0.0,0.0]]],"id":1}
01703     *
01704     * @par JSON-RPC
01705     * {"id":1,"jsonrpc":"2.0","result":0}
01706     * \endchinese
01707     *
01708     * \english
01709     * Add a box collision object
01710     *
01711     * This interface adds a named group of box collision objects into the
01712     * current robot algorithm scene.
01713     *
01714     * @param name Unique identifier of the collision object
01715     * @param link_name Name of the reference link the collision object is
01716     * attached to. Common names include: `world`, `base_link`,
01717     * `shoulder_Link`, `upperArm_Link`, `foreArm_Link`, `wrist1_Link`,
01718     * `wrist2_Link`, `wrist3_Link`, and `end_effector`
01719     * @param sizes List of box sizes. Each element is in the format
01720     * `[length, width, height]`, unit: m
01721     * @param poses List of collision object poses. Each element is a pose
01722     * relative to the reference link in the format `[x, y, z, rx, ry, rz]`
01723     * @return 0 on success, non-zero on failure
01724     *
01725     * @par Lua function prototype
01726     * addCollisionBox(name, link_name, sizes, poses) -> integer
01727     *
01728     * @par Lua example
01729     * -- Add an environment collision box attached to world
01730     * addCollisionBox("env_box", "world",
01731     *                 {{0.4, 0.4, 0.2}},
01732     *                 {{0.6, 0.0, 0.1, 0.0, 0.0, 0.0}})
01733     *
01734     * -- Add a tool collision box attached to end_effector
01735     * addCollisionBox("tool_box", "end_effector",
01736     *                 {{0.1, 0.08, 0.06}},
01737     *                 {{0.0, 0.0, 0.05, 0.0, 0.0, 0.0}})
01738     *
01739     * @par Python function prototype
01740     * addCollisionBox(name: str, link_name: str, sizes: list[list[float]],
01741     * poses: list[list[float]]) -> int
01742     *
01743     * @par Python example
01744     * # Add an environment collision box attached to world
01745     * robot_algorithm.addCollisionBox(
01746     *     "env_box", "world",
01747     *     [[0.4, 0.4, 0.2]],
01748     *     [[0.6, 0.0, 0.1, 0.0, 0.0, 0.0]]
01749     * )
01750     *
01751     * # Add a tool collision box attached to end_effector
01752     * robot_algorithm.addCollisionBox(
01753     *     "tool_box", "end_effector",
01754     *     [[0.1, 0.08, 0.06]],
01755     *     [[0.0, 0.0, 0.05, 0.0, 0.0, 0.0]]
01756     * )
01757     *
01758     * @par JSON-RPC request example
01759     * {"jsonrpc":"2.0","method":"rob1.RobotAlgorithm.addCollisionBox",
01760     * "params":["env_box","world",[[0.4,0.4,0.2]],
01761     * [[0.6,0.0,0.1,0.0,0.0,0.0]]],"id":1}
01762     *
01763     * @par JSON-RPC response example
01764     * {"id":1,"jsonrpc":"2.0","result":0}
01765     * \endenglish
01766     */
01767    int addCollisionBox(const std::string &name,
01768                        const std::string &link_name,
01769                        const std::vector<std::vector<double>> &sizes,
01770                        const std::vector<std::vector<double>> &poses);
01771
01772    /**
01773     * @ingroup RobotAlgorithm
01774     * \chinese
01775     *
01776     *
01777     * @param name
01778     * @return 0      0
01779     *
01780     * @par Lua
01781     * removeCollisionObject(name) -> integer
01782     *
01783     * @par Lua
01784     * removeCollisionObject("tool_box")
01785     *
01786     * @par Python
```

```
01787     * removeCollisionObject(name: str) -> int
01788     *
01789     * @par Python
01790     * robot_algorithm.removeCollisionObject("tool_box")
01791     *
01792     * @par JSON-RPC
01793     * {"jsonrpc":"2.0","method":"rob1.RobotAlgorithm.removeCollisionObject",
01794     * "params":["tool_box"],"id":1}
01795     *
01796     * @par JSON-RPC
01797     * {"id":1,"jsonrpc":"2.0","result":0}
01798     * \endchinese
01799     *
01800     * \english
01801     * Remove a collision object by name
01802     *
01803     * @param name Name of the collision object to remove
01804     * @return 0 on success, non-zero on failure
01805     *
01806     * @par Lua function prototype
01807     * removeCollisionObject(name) -> integer
01808     *
01809     * @par Lua example
01810     * removeCollisionObject("tool_box")
01811     *
01812     * @par Python function prototype
01813     * removeCollisionObject(name: str) -> int
01814     *
01815     * @par Python example
01816     * robot_algorithm.removeCollisionObject("tool_box")
01817     *
01818     * @par JSON-RPC request example
01819     * {"jsonrpc":"2.0","method":"rob1.RobotAlgorithm.removeCollisionObject",
01820     * "params":["tool_box"],"id":1}
01821     *
01822     * @par JSON-RPC response example
01823     * {"id":1,"jsonrpc":"2.0","result":0}
01824     * \endenglish
01825     */
01826    int removeCollisionObject(const std::string &name);
01827
01828    /**
01829     * @ingroup RobotAlgorithm
01830     * \chinese
01831     *
01832     *
01833     *          SDK Lua JSON-RPC   ROS
01834     *
01835     * @param type
01836     *        0 -
01837     *        1 -
01838     * @param start
01839     * @param r1        m
01840     * @param end
01841     * @param r2        m
01842     * @param d        m
01843     * @return
01844     *
01845     * \endchinese
01846     *
01847     * \english
01848     * Validate the reachability of the robot's motion path from start point to
01849     * end point without external collision checks.
01850     *
01851     * This interface is kept for compatibility with existing SDK, Lua,
01852     * JSON-RPC, and ROS service calls.
01853     *
01854     * @param type Path type identifier, used to specify the data types of
01855     * start and end:
01856     *    0 - Start: Joint angles, End: Joint angles
01857     *    1 - Start: Pose, End: Pose
01858     * @param start Start point data of the path
01859     * @param r1 Start point blending radius, unit: m (meters)
01860     * @param end End point data of the path
01861     * @param r2 End point blending radius, unit: m (meters)
01862     * @param d Path sampling interval, unit: m (meters). A smaller interval
01863     * improves validation accuracy but takes more time.
01864     * @return Path reachability result code
01865     *
01866     * \endenglish
01867     */
01868    int validatePath(int type, const std::vector<double> &start, double r1,
01869                     const std::vector<double> &end, double r2, double d);
01870
01871    /**
01872     * @ingroup RobotAlgorithm
01873     * \chinese
```

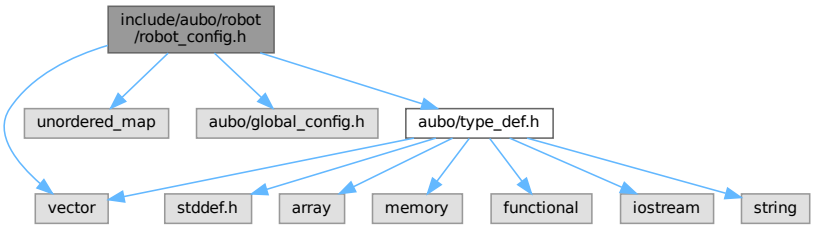
```
01874     *
01875     *
01876     *
01877     *          <->    <->
01878     *
01879     * @param type
01880     *        0 -
01881     *        1 -
01882     * @param start
01883     * @param r1         m
01884     * @param end
01885     * @param r2         m
01886     * @param d         m
01887     * @param check_external_collision
01888     *        false -
01889     *        true  -
01890     * @return
01891     *        0  -
01892     *       -18 -        /
01893     *       -21 -
01894     *       -22 -
01895     *       -24 -
01896     *       -27 -
01897     *
01898     * @throws arcs::common_interface::AuboException
01899     *
01900     * @par Lua
01901     * validatePath1(type, start, r1, end, r2, d, check_external_collision) -> number
01902     *
01903     * @par Lua
01904     * local result =
01905     *     validatePath1(0, start_joint, 0.01, end_joint, 0.01, 0.05, true)
01906     *
01907     * @par Python
01908     * validatePath1(type: int, start: list[float], r1: float, end: list[float],
01909     * r2: float, d: float, check_external_collision: bool) -> int
01910     *
01911     * @par Python
01912     * result = validatePath1(1, start_joint, 0.01, end_pose, 0.01, 0.05, True)
01913     *
01914     * @par JSON-RPC
01915     * {"jsonrpc":"2.0","method":"rob1.RobotAlgorithm.validatePath1","params":[0,
01916     * [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,
01917     * 0.05, true],"id":1}
01918     *
01919     * @par JSON-RPC
01920     * {"id":1,"jsonrpc":"2.0","result":0}
01921     * \endchinese
01922     *
01923     * \english
01924     * Validate the reachability of the robot's motion path from start point to
01925     * end point, with optional external collision checks.
01926     *
01927     * This interface verifies whether the specified path has unreachable
01928     * conditions such as out-of-limit, self-collision, collision with external
01929     * objects, singularity, etc., by sampling. It supports two path type
01930     * validations: Joint-Joint and Pose-Pose.
01931     *
01932     * @param type Path type identifier, used to specify the data types of
01933     * start and end:
01934     *    0 - Start: Joint angles, End: Joint angles
01935     *    1 - Start: Pose, End: Pose
01936     * @param start Start point data of the path
01937     * @param r1 Start point blending radius, unit: m (meters)
01938     * @param end End point data of the path
01939     * @param r2 End point blending radius, unit: m (meters)
01940     * @param d Path sampling interval, unit: m (meters). A smaller interval
01941     * improves validation accuracy but takes more time.
01942     * @param check_external_collision Whether to enable external collision
01943     * checks:
01944     *    false - Keep the original reachability and self-collision checks
01945     *    true  - Add collision checks against external objects
01946     *
01947     * @return Path reachability result code:
01948     *    0  - Reachable: Path is normal, movement is possible
01949     *    -18 - Joint/Cartesian space limits exceeded in the path
01950     *    -21 - Trajectory generation failed
01951     *    -22 - Self-collision of the robot body in the path
01952     *    -24 - Path passing through robot singular configuration
01953     *    -27 - The target point has a solution but exceeds joint limits
01954     *
01955     * @throws arcs::common_interface::AuboException Thrown when parameters are
01956     * invalid.
01957     *
01958     * @par Lua function prototype
01959     * validatePath1(type, start, r1, end, r2, d, check_external_collision) -> integer
01960     *
```

```
01961      * @par Lua example
01962      * result = validatePath1(0, start_joint, 0.01, end_joint, 0.01, 0.05, true)
01963      *
01964      * @par Python function prototype
01965      * validatePath1(type: int, start: list[float], r1: float, end: list[float],
01966      * r2: float, d: float, check_external_collision: bool) -> int
01967      *
01968      * @par Python example
01969      * result = validatePath1(1, start_joint, 0.01, end_pose, 0.01, 0.005, True)
01970      *
01971      * @par JSON-RPC request example
01972      * {"jsonrpc":"2.0","method":"rob1.RobotAlgorithm.validatePath1","params":[0,
01973      * [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,
01974      * 0.005, true],"id":1}
01975      *
01976      * @par JSON-RPC response example {"id":1,"jsonrpc":"2.0","result":0}
01977      * \endenglish
01978      */
01979     int validatePath1(int type, const std::vector<double> &start, double r1,
01980                       const std::vector<double> &end, double r2, double d,
01981                       bool check_external_collision);
01982
01983 protected:
01984     void *d_;
01985 };
01986 using RobotAlgorithmPtr = std::shared_ptr<RobotAlgorithm>;
01987
01988 } // namespace common_interface
01989 } // namespace arcs
01990
01991 #endif // AUBO_SDK_ROBOT_ALGORITHM_INTERFACE_H
```

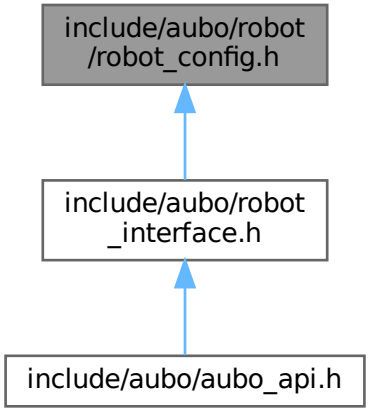
12.28 include/aubo/robot/robot_config.h File Reference

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_ISStation"}
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     *
    {"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     *
      {"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 handguuiding 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     ℃  20℃
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":[0.0,0.000678930024150759,0.002438219962641597,0.0049148499965667725,0.0004869500116910785 4,0.004321440123021603],
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 ℃, default is 20℃
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":[0.0,0.00067893002415075 9,0.002438219962641597,0.0049148499965667725,0.00048695001169107854,0.004321440123021603],
01384     * "beta":[0.0,0.0,0.0,0.0,0.0,0.0,0.0],"d":[0.0,4.769999941345304e-05,-4.769999941345304e-05,4.769999941345304e-
          05,0.0003789000038523227,0.0],
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})
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
02223     *
02224     * \endenglish
02225     */
02226    int confirmSafetyParameters(const RobotSafetyParameterRange &parameters);
02227
02228    /**
02229     * @ingroup RobotConfig
02230     * \chinese
02231     *       CRC32
02232     *
02233     * @return crc32
02234     *
02235     * @throws arcs::common_interface::AuboException
02236     * \endchinese
02237     * \english
02238     * Calculate the CRC32 checksum of safety parameters.
02239     *
02240     * @return crc32
02241     *
02242     * @throws arcs::common_interface::AuboException
02243     * \endenglish
```

```
02244     */
02245     uint32_t calcSafetyParametersCheckSum(
02246         const RobotSafetyParameterRange &parameters);
02247 
02248     /**
02249      * @ingroup RobotConfig
02250      * \chinese
02251      *
02252      *
02253      * @return
02254      *
02255      * @throws arcs::common_interface::AuboException
02256      *
02257      * @par Python
02258      * getJointMaxPositions(self: pyaubo_sdk.RobotConfig) -> List[float]
02259      *
02260      * @par Lua
02261      * getJointMaxPositions() -> table
02262      *
02263      * @par Lua
02264      * JointMaxPositions = getJointMaxPositions()
02265      *
02266      * @par JSON-RPC
02267      * {"jsonrpc":"2.0","method":"rob1.RobotConfig.getJointMaxPositions","params":[],"id":1}
02268      *
02269      * @par JSON-RPC
02270      *
    {"id":1,"jsonrpc":"2.0","result":[6.283185307179586,6.283185307179586,6.283185307179586,6.283185307179586,6.283185307179586,6.2831853071
02271      * \endchinese
02272      * \english
02273      * Get the joint maximum positions (physical limits).
02274      *
02275      * @return Returns the joint maximum positions.
02276      *
02277      * @throws arcs::common_interface::AuboException
02278      *
02279      * @par Python function prototype
02280      * getJointMaxPositions(self: pyaubo_sdk.RobotConfig) -> List[float]
02281      *
02282      * @par Lua function prototype
02283      * getJointMaxPositions() -> table
02284      *
02285      * @par Lua example
02286      * JointMaxPositions = getJointMaxPositions()
02287      *
02288      * @par JSON-RPC request example
02289      * {"jsonrpc":"2.0","method":"rob1.RobotConfig.getJointMaxPositions","params":[],"id":1}
02290      *
02291      * @par JSON-RPC response example
02292      *
    {"id":1,"jsonrpc":"2.0","result":[6.283185307179586,6.283185307179586,6.283185307179586,6.283185307179586,6.283185307179586,6.2831853071
02293      * \endenglish
02294      */
02295     std::vector<double> getJointMaxPositions();
02296 
02297     /**
02298      * @ingroup RobotConfig
02299      * \chinese
02300      *
02301      *
02302      * @return
02303      *
02304      * @throws arcs::common_interface::AuboException
02305      *
02306      * @par Python
02307      * getJointMinPositions(self: pyaubo_sdk.RobotConfig) -> List[float]
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,-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":[3.892084231947355,3.892084231947355,3.892084231947355,3.1066860685499065,3.1066860685499065,3.10668606
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":[3.892084231947355,3.892084231947355,3.892084231947355,3.1066860685499065,3.1066860685499065,3.10668606
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":[31.104877758314785,31.104877758314785,31.104877758314785,20.73625684294463,20.73625684294463,20.736256
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":[31.104877758314785,31.104877758314785,31.104877758314785,20.73625684294463,20.73625684294463,20.736256
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    /**
02661     * @ingroup RobotConfig
02662     * \chinese
02663     *     TCP
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    /**
02732     * @ingroup RobotConfig
02733     * \chinese
02734     *    TCP
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}, {0,0,0,0,0,0,0,0,0})
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}, {0,0,0,0,0,0,0,0,0})
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}, {0,0,0,0,0,0,0,0,0})
02876    *
02877    * @par JSON-RPC
02878    * {"jsonrpc":"2.0","method":"rob1.RobotConfig.setPayload","params":[3,[0,0,0],[0,0,0],[0,0,0,0,0,0,0,0,0]],"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}, {0,0,0,0,0,0,0,0,0})
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     *
03124     * @return          \n
03125     *            failed          \n
03126     *          (0~1)       ("", 1)
03127     *
03128     * @throws arcs::common_interface::AuboException
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     *
{"id":1,"jsonrpc":"2.0","result":[6.2831854820251465,6.2831854820251465,6.2831854820251465,6.2831854820251465,6.2831854820251465,6.283185
03200     * \endchinese
03201     * \english
03202     * Get the joint maximum positions (currently used limit values).
03203     *
03204     * @return Returns the joint maximum positions (currently used limit
03205     * values).
03206     *
03207     * @throws arcs::common_interface::AuboException
03208     *
03209     * @par Python function prototype
03210     * getLimitJointMaxPositions(self: pyaubo_sdk.RobotConfig) -> List[float]
03211     *
03212     * @par Lua function prototype
03213     * getLimitJointMaxPositions() -> table
03214     *
03215     * @par Lua example
03216     * LimitJointMaxPositions = getLimitJointMaxPositions()
03217     *
03218     * @par JSON-RPC request example
03219     * {"jsonrpc":"2.0","method":"rob1.RobotConfig.getLimitJointMaxPositions","params":[],"id":1}
03220     *
03221     * @par JSON-RPC response example
03222     *
{"id":1,"jsonrpc":"2.0","result":[6.2831854820251465,6.2831854820251465,6.2831854820251465,6.2831854820251465,6.2831854820251465,6.283185
03223     * \endenglish
03224     */
03225    std::vector<double> getLimitJointMaxPositions();
03226
03227    /**
03228     * @ingroup RobotConfig
03229     * \chinese
03230     *
03231     *
03232     * @return
03233     *
03234     * @throws arcs::common_interface::AuboException
03235     *
03236     * @par Python
03237     * getLimitJointMinPositions(self: pyaubo_sdk.RobotConfig) -> List[float]
03238     *
03239     * @par Lua
03240     * getLimitJointMinPositions() -> table
03241     *
03242     * @par Lua
03243     * LimitJointMinPositions = getLimitJointMinPositions()
03244     *
03245     * @par JSON-RPC
03246     * {"jsonrpc":"2.0","method":"rob1.RobotConfig.getLimitJointMinPositions","params":[],"id":1}
03247     *
03248     * @par JSON-RPC
03249     * {"id":1,"jsonrpc":"2.0","result":[-6.2831854820251465,-6.2831854820251465,-6.2831854820251465,-
6.2831854820251465,-6.2831854820251465,-6.2831854820251465]}
03250     * \endchinese
03251     * \english
03252     * 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":[3.8920841217041016,3.8920841217041016,3.8920841217041016,3.1066861152648926,3.1066861152648926,3.10668
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":[3.8920841217041016,3.8920841217041016,3.8920841217041016,3.1066861152648926,3.1066861152648926,3.10668
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     *
    {"id":1,"jsonrpc":"2.0","result":[31.104877758314785,31.104877758314785,31.104877758314785,20.73625684294463,20.73625684294463,20.736256
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     *
      {"id":1,"jsonrpc":"2.0","result":[31.104877758314785,31.104877758314785,31.104877758314785,20.73625684294463,20.73625684294463,20.736256
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    SafeguedStopType 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}, {0,0,0,0,0,0,0,0,0})
03599     *
03600     * @par JSON-RPC
03601     * {"jsonrpc":"2.0","method":"rob1.RobotConfig.attachWeldingGun","params":[3,[0,0,0],[0,0,0,0,0,0,0,0,0]],"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     *
```

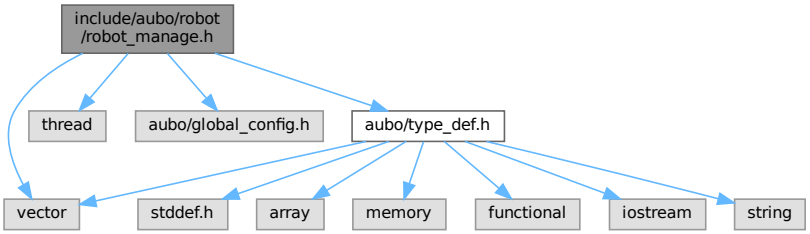
```
03617     * @throws arcs::common_interface::AuboException
03618     *
03619     * @par Python
03620     * setCollisionThreshold(self: pyaubo_sdk.RobotConfig, arg0: List[float])
03621     * -> int
03622     *
03623     * @par Lua
03624     * setCollisionThreshold(threshold: table) -> nil
03625     *
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, 60.0,54.6111111, 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
```

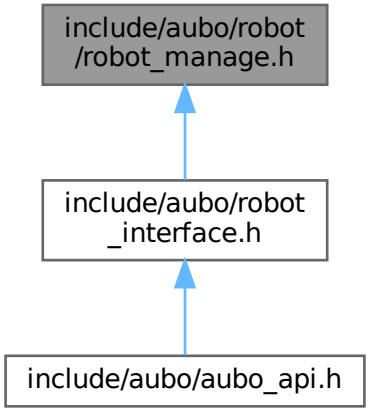
```
03791      * isEndCollisionCheckEnabled() -> bool
03792      *
03793      * @par Lua example
03794      * IsEndCollisionCheckEnabled = isEndCollisionCheckEnabled()
03795      *
03796      * @par JSON-RPC request example
03797      * {"jsonrpc":"2.0","method":"rob1.RobotConfig.isEndCollisionCheckEnabled","params":[],"id":1}
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}, {0,0,0,0,0,0})
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}, {0,0,0,0,0,0})
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     *
    rpc_cli->getRobotInterface(robot_name)->getRobotManage()->setOperationalMode(OperationalModeType::Automatic);
00883     * @endcode
00884     * \endenglish
00885     */
00886    int setOperationalMode(OperationalModeType mode);
00887
00888    /**
00889     * @ingroup RobotManage
00890     * \chinese
00891     *
00892     *
00893     * @return
00894     *
00895     * @throws arcs::common_interface::AuboException
00896     *
00897     * @par Python
00898     * getOperationalMode(self: pyaubo_sdk.RobotManage) ->
00899     * arcs::common_interface::OperationalModeType
00900     *
00901     * @par Lua
00902     * getOperationalMode() -> number
00903     *
00904     * @par Lua
00905     * num = getOperationalMode()
00906     *
00907     * @par JSON-RPC
00908     * {"jsonrpc":"2.0","method":"rob1.RobotManage.getOperationalMode","params":[],"id":1}
00909     *
00910     * @par JSON-RPC
00911     * {"id":1,"jsonrpc":"2.0","result":"Manual"}
00912     *
00913     * @par C++
00914     * @code
00915     * auto rpc_cli = std::make_shared<RpcClient>();
00916     * auto robot_name = rpc_cli->getRobotNames().front();
00917     * OperationalModeType mode =
00918     * rpc_cli->getRobotInterface(robot_name)->getRobotManage()->getOperationalMode();
00919     * @endcode
00920     * \endchinese
00921     * \english
00922     * Get the robot operational mode
00923     *
00924     * @return Robot operational mode
00925     *
00926     * @throws arcs::common_interface::AuboException
00927     *
00928     * @par Python function prototype
00929     * getOperationalMode(self: pyaubo_sdk.RobotManage) ->
00930     * arcs::common_interface::OperationalModeType
00931     *
00932     * @par Lua function prototype
00933     * getOperationalMode() -> number
00934     *
00935     * @par Lua example
00936     * num = getOperationalMode()
00937     *
00938     * @par JSON-RPC request example
00939     * {"jsonrpc":"2.0","method":"rob1.RobotManage.getOperationalMode","params":[],"id":1}
00940     *
00941     * @par JSON-RPC response example
00942     * {"id":1,"jsonrpc":"2.0","result":"Manual"}
00943     *
00944     * @par C++ example
00945     * @code
00946     * auto rpc_cli = std::make_shared<RpcClient>();
00947     * auto robot_name = rpc_cli->getRobotNames().front();
00948     * OperationalModeType mode =
00949     * rpc_cli->getRobotInterface(robot_name)->getRobotManage()->getOperationalMode();
00950     * @endcode
```

```
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
```

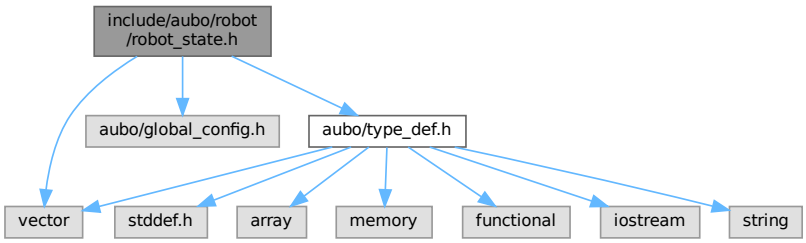
```
01821     *
01822     * @par Lua example
01823     * LinkModeEnabled = isLinkModeEnabled()
01824     *
01825     * @par JSON-RPC request example
01826     * {"jsonrpc":"2.0","method":"rob1.RobotManage.isLinkModeEnabled","params":[],"id":1}
01827     *
01828     * @par JSON-RPC response example
01829     * {"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
```

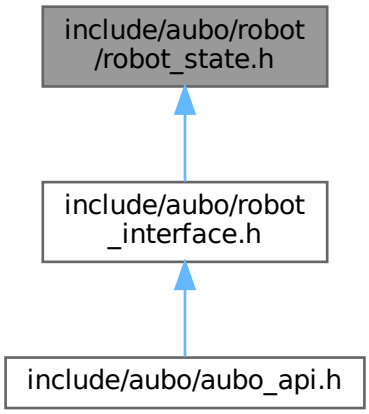
12.32 include/aubo/robot/robot_state.h File Reference

/

#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      *
00031      * @return
00032      *
00033      * @throws arcs::common_interface::AuboException
00034      *
00035      * @par Python
00036      * getRobotModeType(self: pyaubo_sdk.RobotState) ->
00037      * arcs::common_interface::RobotModeType
00038      *
00039      * @par Lua
00040      * getRobotModeType() -> number
00041      *
00042      * @par Lua
00043      * RobotModeType = getRobotModeType()
00044      *
00045      * @par JSON-RPC
```

```
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,-
      0.13250000000000012,0.2092832117232601,3.1415812372223217,0.0,1.5707963267948963]}
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,-
      0.13250000000000012,0.2092832117232601,3.1415812372223217,0.0,1.5707963267948963]}
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"]}
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"]}
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"]}
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
```

```
01509     * JointVoltages = getJointVoltages()
01510     *
01511     * @par JSON-RPC request example
01512     * {"jsonrpc":"2.0","method":"rob1.RobotState.getJointVoltages","params":[],"id":1}
01513     *
01514     * @par JSON-RPC response example
01515     * {"id":1,"jsonrpc":"2.0","result":[2.0,2.5,3.0,2.0,2.5,2.0]}
01516     * \endenglish
01517     */
01518    std::vector<double> getJointVoltages();
01519
01520    /**
01521     * @ingroup RobotState
01522     * \chinese
01523     *
01524     *
01525     * @return
01526     *
01527     * @throws arcs::common_interface::AuboException
01528     *
01529     * @par Python
01530     * getJointTemperatures(self: pyaubo_sdk.RobotState) -> List[float]
01531     *
01532     * @par Lua
01533     * getJointTemperatures() -> table
01534     *
01535     * @par Lua
01536     * JointTemperatures = getJointTemperatures()
01537     *
01538     * @par JSON-RPC
01539     * {"jsonrpc":"2.0","method":"rob1.RobotState.getJointTemperatures","params":[],"id":1}
01540     *
01541     * @par JSON-RPC
01542     * {"id":1,"jsonrpc":"2.0","result":[38.0,38.0,38.0,39.0,38.0,39.0]}
01543     * \endchinese
01544     * \english
01545     * Get the joint temperatures of the manipulator
01546     *
01547     * @return Joint temperatures of the manipulator
01548     *
01549     * @throws arcs::common_interface::AuboException
01550     *
01551     * @par Python function prototype
01552     * getJointTemperatures(self: pyaubo_sdk.RobotState) -> List[float]
01553     *
01554     * @par Lua function prototype
01555     * getJointTemperatures() -> table
01556     *
01557     * @par Lua example
01558     * JointTemperatures = getJointTemperatures()
01559     *
01560     * @par JSON-RPC request example
01561     * {"jsonrpc":"2.0","method":"rob1.RobotState.getJointTemperatures","params":[],"id":1}
01562     *
01563     * @par JSON-RPC response example
01564     * {"id":1,"jsonrpc":"2.0","result":[38.0,38.0,38.0,39.0,38.0,39.0]}
01565     * \endenglish
01566     */
01567    std::vector<double> getJointTemperatures();
01568
01569    /**
01570     * @ingroup RobotState
01571     * \chinese
01572     *      ID
01573     *
01574     * @return     ID
01575     *
01576     * @throws arcs::common_interface::AuboException
01577     *
01578     * @par Python
01579     * getJointUniqueIds(self: pyaubo_sdk.RobotState) -> List[str]
01580     *
01581     * @par Lua
01582     * getJointUniqueIds() -> table
01583     *
01584     * @par Lua
01585     * JointUniqueIds = getJointUniqueIds()
01586     *
01587     * @par JSON-RPC
01588     * {"jsonrpc":"2.0","method":"rob1.RobotState.getJointUniqueIds","params":[],"id":1}
01589     *
01590     * @par JSON-RPC
01591     *
    {"id":1,"jsonrpc":"2.0","result":["00800020ffffff31405153","00800020ffffff3e3f5153","00800020ffffff414b5153","00800020ffffff31065153","0080
01592     * \endchinese
01593     * \english
01594     * Get the globally unique IDs of the joints
```

```
01595     *
01596     * @return Globally unique IDs of the joints
01597     *
01598     * @throws arcs::common_interface::AuboException
01599     *
01600     * @par Python function prototype
01601     * getJointUniqueIds(self: pyaubo_sdk.RobotState) -> List[str]
01602     *
01603     * @par Lua function prototype
01604     * getJointUniqueIds() -> table
01605     *
01606     * @par Lua example
01607     * JointUniqueIds = getJointUniqueIds()
01608     *
01609     * @par JSON-RPC request example
01610     * {"jsonrpc":"2.0","method":"rob1.RobotState.getJointUniqueIds","params":[],"id":1}
01611     *
01612     * @par JSON-RPC response example
01613     *
    {"id":1,"jsonrpc":"2.0","result":["00800020ffffffff31405153","00800020ffffffff3e3f5153","00800020ffffffff414b5153","00800020ffffffff31065153","0080
01614     * \endenglish
01615     */
01616    std::vector<std::string> getJointUniqueIds();
01617
01618    /**
01619     * @ingroup RobotState
01620     * \chinese
01621     *
01622     *
01623     * @return
01624     *
01625     * @throws arcs::common_interface::AuboException
01626     *
01627     * @par Python
01628     * getJointFirmwareVersions(self: pyaubo_sdk.RobotState) -> List[int]
01629     *
01630     * @par Lua
01631     * getJointFirmwareVersions() -> table
01632     *
01633     * @par Lua
01634     * JointFirmwareVersions = getJointFirmwareVersions()
01635     *
01636     * @par JSON-RPC
01637     * {"jsonrpc":"2.0","method":"rob1.RobotState.getJointFirmwareVersions","params":[],"id":1}
01638     *
01639     * @par JSON-RPC
01640     * {"id":1,"jsonrpc":"2.0","result":[1000010,1000010,1000010,1000010,1000010,1000010]}
01641     * \endchinese
01642     * \english
01643     * Get the joint firmware versions
01644     *
01645     * @return Joint firmware versions
01646     *
01647     * @throws arcs::common_interface::AuboException
01648     *
01649     * @par Python function prototype
01650     * getJointFirmwareVersions(self: pyaubo_sdk.RobotState) -> List[int]
01651     *
01652     * @par Lua function prototype
01653     * getJointFirmwareVersions() -> table
01654     *
01655     * @par Lua example
01656     * JointFirmwareVersions = getJointFirmwareVersions()
01657     *
01658     * @par JSON-RPC request example
01659     * {"jsonrpc":"2.0","method":"rob1.RobotState.getJointFirmwareVersions","params":[],"id":1}
01660     *
01661     * @par JSON-RPC response example
01662     * {"id":1,"jsonrpc":"2.0","result":[1000010,1000010,1000010,1000010,1000010,1000010]}
01663     * \endenglish
01664     */
01665    std::vector<int> getJointFirmwareVersions();
01666
01667    /**
01668     * @ingroup RobotState
01669     * \chinese
01670     *
01671     *
01672     * @return
01673     *
01674     * @throws arcs::common_interface::AuboException
01675     *
01676     * @par Python
01677     * getJointHardwareVersions(self: pyaubo_sdk.RobotState) -> List[int]
01678     *
01679     * @par Lua
01680     * getJointHardwareVersions() -> table
```

```
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":"736572630080000000000000"}
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":"736572630080000000000000"}
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     * getSlaveBoardHardwareVersion(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     * getSlaveBoardHardwareVersion(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.getSlaveBoardHardwareVersion","params":[],"id":1}
02003     *
02004     * @par JSON-RPC response example
02005     * {"id":1,"jsonrpc":"2.0","result":6030098}
02006     * \endenglish
02007     */
02008    int getSlaveBoardHardwareVersion();
02009
02010    /**
02011     * @ingroup RobotState
02012     * \chinese
02013     *  Handle  ID
02014     *
02015     * @return Handle  ID
02016     *
02017     * @throws arcs::common_interface::AuboException
02018     *
02019     * @par Python
02020     * getHandleUniqueId(self: pyaubo_sdk.RobotState) -> str
02021     *
02022     * @par Lua
02023     * getHandleUniqueId() -> string
02024     *
02025     * @par Lua
02026     * HandleUniqueId = getHandleUniqueId()
02027     *
02028     * @par JSON-RPC
```

```
02029     * {"jsonrpc":"2.0","method":"rob1.RobotState.getHandleUniqueId","params":[],"id":1}
02030     *
02031     * @par JSON-RPC
02032     * {"id":1,"jsonrpc":"2.0","result":"73657263008000000000000"}
02033     * \endchinese
02034     * \english
02035     * Get the globally unique ID of the Handle
02036     *
02037     * @return Globally unique ID of the Handle
02038     *
02039     * @throws arcs::common_interface::AuboException
02040     *
02041     * @par Python function prototype
02042     * getHandleUniqueId(self: pyaubo_sdk.RobotState) -> str
02043     *
02044     * @par Lua function prototype
02045     * getHandleUniqueId() -> string
02046     *
02047     * @par Lua example
02048     * HandleUniqueId = getHandleUniqueId()
02049     *
02050     * @par JSON-RPC request example
02051     * {"jsonrpc":"2.0","method":"rob1.RobotState.getHandleUniqueId","params":[],"id":1}
02052     *
02053     * @par JSON-RPC response example
02054     * {"id":1,"jsonrpc":"2.0","result":"73657263008000000000000"}
02055     * \endenglish
02056     */
02057    std::string getHandleUniqueId();
02058
02059    /**
02060     * @ingroup RobotState
02061     * \chinese
02062     *  Handle
02063     *
02064     * @return Handle
02065     *
02066     * @throws arcs::common_interface::AuboException
02067     *
02068     * @par Python
02069     * getHandleFirmwareVersion(self: pyaubo_sdk.RobotState) -> int
02070     *
02071     * @par Lua
02072     * getHandleFirmwareVersion() -> number
02073     *
02074     * @par Lua
02075     * HandleFirmwareVersion = getHandleFirmwareVersion()
02076     *
02077     * @par JSON-RPC
02078     * {"jsonrpc":"2.0","method":"rob1.RobotState.getHandleFirmwareVersion","params":[],"id":1}
02079     *
02080     * @par JSON-RPC
02081     * {"id":1,"jsonrpc":"2.0","result":0}
02082     * \endchinese
02083     * \english
02084     * Get the Handle firmware version
02085     *
02086     * @return Handle firmware version
02087     *
02088     * @throws arcs::common_interface::AuboException
02089     *
02090     * @par Python function prototype
02091     * getHandleFirmwareVersion(self: pyaubo_sdk.RobotState) -> int
02092     *
02093     * @par Lua function prototype
02094     * getHandleFirmwareVersion() -> number
02095     *
02096     * @par Lua example
02097     * HandleFirmwareVersion = getHandleFirmwareVersion()
02098     *
02099     * @par JSON-RPC request example
02100     * {"jsonrpc":"2.0","method":"rob1.RobotState.getHandleFirmwareVersion","params":[],"id":1}
02101     *
02102     * @par JSON-RPC response example
02103     * {"id":1,"jsonrpc":"2.0","result":0}
02104     * \endenglish
02105     */
02106    int getHandleFirmwareVersion();
02107
02108    /**
02109     * @ingroup RobotState
02110     * \chinese
02111     *  Handle
02112     *
02113     * @return Handle
02114     *
02115     * @throws arcs::common_interface::AuboException
```

```
02116     *
02117     * @par Python
02118     * getHandleHardwareVersion(self: pyaubo_sdk.RobotState) -> int
02119     *
02120     * @par Lua
02121     * getHandleHardwareVersion() -> number
02122     *
02123     * @par Lua
02124     * HandleHardwareVersion = getHandleHardwareVersion()
02125     *
02126     * @par JSON-RPC
02127     * {"jsonrpc":"2.0","method":"rob1.RobotState.getHandleHardwareVersion","params":[],"id":1}
02128     *
02129     * @par JSON-RPC
02130     * {"id":1,"jsonrpc":"2.0","result":6030098}
02131     * \endchinese
02132     * \english
02133     * Get the Handle hardware version
02134     *
02135     * @return Handle hardware version
02136     *
02137     * @throws arcs::common_interface::AuboException
02138     *
02139     * @par Python function prototype
02140     * getHandleHardwareVersion(self: pyaubo_sdk.RobotState) -> int
02141     *
02142     * @par Lua function prototype
02143     * getHandleHardwareVersion() -> number
02144     *
02145     * @par Lua example
02146     * HandleHardwareVersion = getHandleHardwareVersion()
02147     *
02148     * @par JSON-RPC request example
02149     * {"jsonrpc":"2.0","method":"rob1.RobotState.getHandleHardwareVersion","params":[],"id":1}
02150     *
02151     * @par JSON-RPC response example
02152     * {"id":1,"jsonrpc":"2.0","result":6030098}
02153     * \endenglish
02154     */
02155    int getHandleHardwareVersion();
02156
02157    /**
02158     * @ingroup RobotState
02159     * \chinese
02160     *  Handle
02161     *
02162     * @return Handle
02163     *
02164     * @throws arcs::common_interface::AuboException
02165     *
02166     * @par Python
02167     * getHandleState(self: pyaubo_sdk.RobotState) -> int
02168     *
02169     * @par Lua
02170     * getHandleState() -> number
02171     *
02172     * @par Lua
02173     * HandleState = getHandleState()
02174     *
02175     * @par JSON-RPC
02176     * {"jsonrpc":"2.0","method":"rob1.RobotState.getHandleState","params":[],"id":1}
02177     *
02178     * @par JSON-RPC
02179     * {"id":1,"jsonrpc":"2.0","result":6030098}
02180     * \endchinese
02181     * \english
02182     * Get the Handle state
02183     *
02184     * @return Handle state
02185     *
02186     * @throws arcs::common_interface::AuboException
02187     *
02188     * @par Python function prototype
02189     * getHandleState(self: pyaubo_sdk.RobotState) -> int
02190     *
02191     * @par Lua function prototype
02192     * getHandleState() -> number
02193     *
02194     * @par Lua example
02195     * HandleState = getHandleState()
02196     *
02197     * @par JSON-RPC request example
02198     * {"jsonrpc":"2.0","method":"rob1.RobotState.getHandleState","params":[],"id":1}
02199     *
02200     * @par JSON-RPC response example
02201     * {"id":1,"jsonrpc":"2.0","result":6030098}
02202     * \endenglish
```

```
02203     */
02204     HandleStateType getHandleState();
02205
02206     /**
02207      * @ingroup RobotState
02208      * \chinese
02209      *  Handle
02210      *
02211      * @return Handle
02212      *
02213      * @throws arcs::common_interface::AuboException
02214      *
02215      * @par Python
02216      * getHandleMode(self: pyaubo_sdk.RobotState) -> int
02217      *
02218      * @par Lua
02219      * getHandleMode() -> number
02220      *
02221      * @par Lua
02222      * HandleMode = getHandleMode()
02223      *
02224      * @par JSON-RPC
02225      * {"jsonrpc":"2.0","method":"rob1.RobotState.getHandleMode","params":[],"id":1}
02226      *
02227      * @par JSON-RPC
02228      * {"id":1,"jsonrpc":"2.0","result":6030098}
02229      * \endchinese
02230      * \english
02231      * Get the Handle mode
02232      *
02233      * @return Handle mode
02234      *
02235      * @throws arcs::common_interface::AuboException
02236      *
02237      * @par Python function prototype
02238      * getHandleMode(self: pyaubo_sdk.RobotState) -> int
02239      *
02240      * @par Lua function prototype
02241      * getHandleMode() -> number
02242      *
02243      * @par Lua example
02244      * HandleMode = getHandleMode()
02245      *
02246      * @par JSON-RPC request example
02247      * {"jsonrpc":"2.0","method":"rob1.RobotState.getHandleMode","params":[],"id":1}
02248      *
02249      * @par JSON-RPC response example
02250      * {"id":1,"jsonrpc":"2.0","result":1}
02251      * \endenglish
02252      */
02253     HandleModeType getHandleMode();
02254
02255     /**
02256      * @ingroup RobotState
02257      * \chinese
02258      *  PowerlossBoard  ID
02259      *
02260      * @return PowerlossBoard  ID
02261      *
02262      * @throws arcs::common_interface::AuboException
02263      *
02264      * @par Python
02265      * getPowerlossBoardUniqueId(self: pyaubo_sdk.RobotState) -> str
02266      *
02267      * @par Lua
02268      * getPowerlossBoardUniqueId() -> string
02269      *
02270      * @par Lua
02271      * PowerlossBoardUniqueId = getPowerlossBoardUniqueId()
02272      *
02273      * @par JSON-RPC
02274      * {"jsonrpc":"2.0","method":"rob1.RobotState.getPowerlossBoardUniqueId","params":[],"id":1}
02275      *
02276      * @par JSON-RPC
02277      * {"id":1,"jsonrpc":"2.0","result":"736572630080000000000000"}
02278      * \endchinese
02279      * \english
02280      * Get the globally unique ID of the PowerlossBoard
02281      *
02282      * @return Globally unique ID of the PowerlossBoard
02283      *
02284      * @throws arcs::common_interface::AuboException
02285      *
02286      * @par Python function prototype
02287      * getPowerlossBoardUniqueId(self: pyaubo_sdk.RobotState) -> str
02288      *
02289      * @par Lua function prototype
```

```
02290     * getPowerlossBoardUniqueId() -> string
02291     *
02292     * @par Lua example
02293     * PowerlossBoardUniqueId = getPowerlossBoardUniqueId()
02294     *
02295     * @par JSON-RPC request example
02296     * {"jsonrpc":"2.0","method":"rob1.RobotState.getPowerlossBoardUniqueId","params":[],"id":1}
02297     *
02298     * @par JSON-RPC response example
02299     * {"id":1,"jsonrpc":"2.0","result":"736572630080000000000000"}
02300     * \endenglish
02301     */
02302    std::string getPowerlossBoardUniqueId();
02303
02304    /**
02305     * @ingroup RobotState
02306     * \chinese
02307     *  PowerlossBoard
02308     *
02309     * @return PowerlossBoard
02310     *
02311     * @throws arcs::common_interface::AuboException
02312     *
02313     * @par Python
02314     * getPowerlossBoardFirmwareVersion(self: pyaubo_sdk.RobotState) -> int
02315     *
02316     * @par Lua
02317     * getPowerlossBoardFirmwareVersion() -> number
02318     *
02319     * @par Lua
02320     * PowerlossBoardFirmwareVersion = getPowerlossBoardFirmwareVersion()
02321     *
02322     * @par JSON-RPC
02323     * {"jsonrpc":"2.0","method":"rob1.RobotState.getPowerlossBoardFirmwareVersion","params":[],"id":1}
02324     *
02325     * @par JSON-RPC
02326     * {"id":1,"jsonrpc":"2.0","result":0}
02327     * \endchinese
02328     * \english
02329     * Get the PowerlossBoard firmware version
02330     *
02331     * @return PowerlossBoard firmware version
02332     *
02333     * @throws arcs::common_interface::AuboException
02334     *
02335     * @par Python function prototype
02336     * getPowerlossBoardFirmwareVersion(self: pyaubo_sdk.RobotState) -> int
02337     *
02338     * @par Lua function prototype
02339     * getPowerlossBoardFirmwareVersion() -> number
02340     *
02341     * @par Lua example
02342     * PowerlossBoardFirmwareVersion = getPowerlossBoardFirmwareVersion()
02343     *
02344     * @par JSON-RPC request example
02345     * {"jsonrpc":"2.0","method":"rob1.RobotState.getPowerlossBoardFirmwareVersion","params":[],"id":1}
02346     *
02347     * @par JSON-RPC response example
02348     * {"id":1,"jsonrpc":"2.0","result":0}
02349     * \endenglish
02350     */
02351    int getPowerlossBoardFirmwareVersion();
02352
02353    /**
02354     * @ingroup RobotState
02355     * \chinese
02356     *  PowerlossBoard
02357     *
02358     * @return PowerlossBoard
02359     *
02360     * @throws arcs::common_interface::AuboException
02361     *
02362     * @par Python
02363     * getPowerlossBoardHardwareVersion(self: pyaubo_sdk.RobotState) -> int
02364     *
02365     * @par Lua
02366     * getPowerlossBoardHardwareVersion() -> number
02367     *
02368     * @par Lua
02369     * PowerlossBoardHardwareVersion = getPowerlossBoardHardwareVersion()
02370     *
02371     * @par JSON-RPC
02372     * {"jsonrpc":"2.0","method":"rob1.RobotState.getPowerlossBoardHardwareVersion","params":[],"id":1}
02373     *
02374     * @par JSON-RPC
02375     * {"id":1,"jsonrpc":"2.0","result":6030098}
02376     * \endchinese
```

```
02377     * \english
02378     * Get the PowerlossBoard hardware version
02379     *
02380     * @return PowerlossBoard hardware version
02381     *
02382     * @throws arcs::common_interface::AuboException
02383     *
02384     * @par Python function prototype
02385     * getPowerlossBoardHardwareVersion(self: pyaubo_sdk.RobotState) -> int
02386     *
02387     * @par Lua function prototype
02388     * getPowerlossBoardHardwareVersion() -> number
02389     *
02390     * @par Lua example
02391     * PowerlossBoardHardwareVersion = getPowerlossBoardHardwareVersion()
02392     *
02393     * @par JSON-RPC request example
02394     * {"jsonrpc":"2.0","method":"rob1.RobotState.getPowerlossBoardHardwareVersion","params":[],"id":1}
02395     *
02396     * @par JSON-RPC response example
02397     * {"id":1,"jsonrpc":"2.0","result":6030098}
02398     * \endenglish
02399     */
02400    int getPowerlossBoardHardwareVersion();
02401
02402    /**
02403     * @ingroup RobotState
02404     * \chinese
02405     *       ID
02406     *
02407     * @return      ID
02408     *
02409     * @throws arcs::common_interface::AuboException
02410     *
02411     * @par Python
02412     * getToolUniqueId(self: pyaubo_sdk.RobotState) -> str
02413     *
02414     * @par Lua
02415     * getToolUniqueId() -> string
02416     *
02417     * @par Lua
02418     * ToolUniqueId = getToolUniqueId()
02419     *
02420     * @par JSON-RPC
02421     * {"jsonrpc":"2.0","method":"rob1.RobotState.getToolUniqueId","params":[],"id":1}
02422     *
02423     * @par JSON-RPC
02424     * {"id":1,"jsonrpc":"2.0","result":"397d4e5331541252314d3042"}
02425     * \endchinese
02426     * \english
02427     * Get the globally unique ID of the tool
02428     *
02429     * @return Globally unique ID of the tool
02430     *
02431     * @throws arcs::common_interface::AuboException
02432     *
02433     * @par Python function prototype
02434     * getToolUniqueId(self: pyaubo_sdk.RobotState) -> str
02435     *
02436     * @par Lua function prototype
02437     * getToolUniqueId() -> string
02438     *
02439     * @par Lua example
02440     * ToolUniqueId = getToolUniqueId()
02441     *
02442     * @par JSON-RPC request example
02443     * {"jsonrpc":"2.0","method":"rob1.RobotState.getToolUniqueId","params":[],"id":1}
02444     *
02445     * @par JSON-RPC response example
02446     * {"id":1,"jsonrpc":"2.0","result":"397d4e5331541252314d3042"}
02447     * \endenglish
02448     */
02449    std::string getToolUniqueId();
02450
02451    /**
02452     * @ingroup RobotState
02453     * \chinese
02454     *
02455     *
02456     * @return
02457     *
02458     * @throws arcs::common_interface::AuboException
02459     *
02460     * @par Python
02461     * getToolFirmwareVersion(self: pyaubo_sdk.RobotState) -> int
02462     *
02463     * @par Lua
```

```
02464     * getToolFirmwareVersion() -> number
02465     *
02466     * @par Lua
02467     * ToolFirmwareVersion = getToolFirmwareVersion()
02468     *
02469     * @par JSON-RPC
02470     * {"jsonrpc":"2.0","method":"rob1.RobotState.getToolFirmwareVersion","params":[],"id":1}
02471     *
02472     * @par JSON-RPC
02473     * {"id":1,"jsonrpc":"2.0","result":1001003}
02474     * \endchinese
02475     * \english
02476     * Get the tool firmware version
02477     *
02478     * @return Tool firmware version
02479     *
02480     * @throws arcs::common_interface::AuboException
02481     *
02482     * @par Python function prototype
02483     * getToolFirmwareVersion(self: pyaubo_sdk.RobotState) -> int
02484     *
02485     * @par Lua function prototype
02486     * getToolFirmwareVersion() -> number
02487     *
02488     * @par Lua example
02489     * ToolFirmwareVersion = getToolFirmwareVersion()
02490     *
02491     * @par JSON-RPC request example
02492     * {"jsonrpc":"2.0","method":"rob1.RobotState.getToolFirmwareVersion","params":[],"id":1}
02493     *
02494     * @par JSON-RPC response example
02495     * {"id":1,"jsonrpc":"2.0","result":1001003}
02496     * \endenglish
02497     */
02498    int getToolFirmwareVersion();
02499
02500    /**
02501     * @ingroup RobotState
02502     * \chinese
02503     *
02504     *
02505     * @return
02506     *
02507     * @throws arcs::common_interface::AuboException
02508     *
02509     * @par Python
02510     * getToolHardwareVersion(self: pyaubo_sdk.RobotState) -> int
02511     *
02512     * @par Lua
02513     * getToolHardwareVersion() -> number
02514     *
02515     * @par Lua
02516     * ToolHardwareVersion = getToolHardwareVersion()
02517     *
02518     * @par JSON-RPC
02519     * {"jsonrpc":"2.0","method":"rob1.RobotState.getToolHardwareVersion","params":[],"id":1}
02520     *
02521     * @par JSON-RPC
02522     * {"id":1,"jsonrpc":"2.0","result":1000000}
02523     * \endchinese
02524     * \english
02525     * Get the tool hardware version
02526     *
02527     * @return Tool hardware version
02528     *
02529     * @throws arcs::common_interface::AuboException
02530     *
02531     * @par Python function prototype
02532     * getToolHardwareVersion(self: pyaubo_sdk.RobotState) -> int
02533     *
02534     * @par Lua function prototype
02535     * getToolHardwareVersion() -> number
02536     *
02537     * @par Lua example
02538     * ToolHardwareVersion = getToolHardwareVersion()
02539     *
02540     * @par JSON-RPC request example
02541     * {"jsonrpc":"2.0","method":"rob1.RobotState.getToolHardwareVersion","params":[],"id":1}
02542     *
02543     * @par JSON-RPC response example
02544     * {"id":1,"jsonrpc":"2.0","result":1000000}
02545     * \endenglish
02546     */
02547    int getToolHardwareVersion();
02548
02549    /**
02550     * @ingroup RobotState
```

```
02551     * \chinese
02552     *
02553     *
02554     * @return
02555     * 0:
02556     * 1:
02557     * 2:
02558     *
02559     * @throws arcs::common_interface::AuboException
02560     *
02561     * @par Python
02562     * getToolCommMode(self: pyaubo_sdk.RobotState) -> int
02563     *
02564     * @par Lua
02565     * getToolCommMode() -> number
02566     *
02567     * @par Lua
02568     * ToolCommMode = getToolCommMode()
02569     *
02570     * @par JSON-RPC
02571     * {"jsonrpc":"2.0","method":"rob1.RobotState.getToolCommMode","params":[],"id":1}
02572     *
02573     * @par JSON-RPC
02574     * {"id":1,"jsonrpc":"2.0","result":1}
02575     * \endchinese
02576     * \english
02577     * Get the tool communication mode
02578     *
02579     * @return Tool communication mode
02580     * 0: No serial port
02581     * 1: Serial port only
02582     * 2: With force sensor and serial port
02583     *
02584     * @throws arcs::common_interface::AuboException
02585     *
02586     * @par Python function prototype
02587     * getToolCommMode(self: pyaubo_sdk.RobotState) -> int
02588     *
02589     * @par Lua function prototype
02590     * getToolCommMode() -> number
02591     *
02592     * @par Lua example
02593     * ToolCommMode = getToolCommMode()
02594     *
02595     * @par JSON-RPC request example
02596     * {"jsonrpc":"2.0","method":"rob1.RobotState.getToolCommMode","params":[],"id":1}
02597     *
02598     * @par JSON-RPC response example
02599     * {"id":1,"jsonrpc":"2.0","result":1}
02600     * \endenglish
02601     */
02602    int getToolCommMode();
02603
02604    /**
02605     * @ingroup RobotState
02606     * \chinese
02607     *      ID
02608     *
02609     * @return      ID
02610     *
02611     * @throws arcs::common_interface::AuboException
02612     *
02613     * @par Python
02614     * getPedestalUniqueId(self: pyaubo_sdk.RobotState) -> str
02615     *
02616     * @par Lua
02617     * getPedestalUniqueId() -> string
02618     *
02619     * @par Lua
02620     * PedestalUniqueId = getPedestalUniqueId()
02621     *
02622     * @par JSON-RPC
02623     * {"jsonrpc":"2.0","method":"rob1.RobotState.getPedestalUniqueId","params":[],"id":1}
02624     *
02625     * @par JSON-RPC
02626     * {"id":1,"jsonrpc":"2.0","result":"20525753354306524854433 9"}
02627     * \endchinese
02628     * \english
02629     * Get the globally unique ID of the pedestal
02630     *
02631     * @return Globally unique ID of the pedestal
02632     *
02633     * @throws arcs::common_interface::AuboException
02634     *
02635     * @par Python function prototype
02636     * getPedestalUniqueId(self: pyaubo_sdk.RobotState) -> str
02637     *
```

```
02638     * @par Lua function prototype
02639     * getPedestalUniqueId() -> string
02640     *
02641     * @par Lua example
02642     * PedestalUniqueId = getPedestalUniqueId()
02643     *
02644     * @par JSON-RPC request example
02645     * {"jsonrpc":"2.0","method":"rob1.RobotState.getPedestalUniqueId","params":[],"id":1}
02646     *
02647     * @par JSON-RPC response example
02648     * {"id":1,"jsonrpc":"2.0","result":"20525753354306524854433 9"}
02649     * \endenglish
02650     */
02651    std::string getPedestalUniqueId();
02652
02653    /**
02654     * @ingroup RobotState
02655     * \chinese
02656     *
02657     *
02658     * @return
02659     *
02660     * @throws arcs::common_interface::AuboException
02661     *
02662     * @par Python
02663     * getPedestalFirmwareVersion(self: pyaubo_sdk.RobotState) -> int
02664     *
02665     * @par Lua
02666     * getPedestalFirmwareVersion() -> number
02667     *
02668     * @par Lua
02669     * PedestalFirmwareVersion = getPedestalFirmwareVersion()
02670     *
02671     * @par JSON-RPC
02672     * {"jsonrpc":"2.0","method":"rob1.RobotState.getPedestalFirmwareVersion","params":[],"id":1}
02673     *
02674     * @par JSON-RPC
02675     * {"id":1,"jsonrpc":"2.0","result":1000004}
02676     * \endchinese
02677     * \english
02678     * Get the pedestal firmware version
02679     *
02680     * @return Pedestal firmware version
02681     *
02682     * @throws arcs::common_interface::AuboException
02683     *
02684     * @par Python function prototype
02685     * getPedestalFirmwareVersion(self: pyaubo_sdk.RobotState) -> int
02686     *
02687     * @par Lua function prototype
02688     * getPedestalFirmwareVersion() -> number
02689     *
02690     * @par Lua example
02691     * PedestalFirmwareVersion = getPedestalFirmwareVersion()
02692     *
02693     * @par JSON-RPC request example
02694     * {"jsonrpc":"2.0","method":"rob1.RobotState.getPedestalFirmwareVersion","params":[],"id":1}
02695     *
02696     * @par JSON-RPC response example
02697     * {"id":1,"jsonrpc":"2.0","result":1000004}
02698     * \endenglish
02699     */
02700    int getPedestalFirmwareVersion();
02701
02702    /**
02703     * @ingroup RobotState
02704     * \chinese
02705     *
02706     *
02707     * @return
02708     *
02709     * @throws arcs::common_interface::AuboException
02710     *
02711     * @par Python
02712     * getPedestalHardwareVersion(self: pyaubo_sdk.RobotState) -> int
02713     *
02714     * @par Lua
02715     * getPedestalHardwareVersion() -> number
02716     *
02717     * @par Lua
02718     * PedestalHardwareVersion = getPedestalHardwareVersion()
02719     *
02720     * @par JSON-RPC
02721     * {"jsonrpc":"2.0","method":"rob1.RobotState.getPedestalHardwareVersion","params":[],"id":1}
02722     *
02723     * @par JSON-RPC
02724     * {"id":1,"jsonrpc":"2.0","result":1007000}
```

```
02725     * \endchinese
02726     * \english
02727     * Get the pedestal hardware version
02728     *
02729     * @return Pedestal hardware version
02730     *
02731     * @throws arcs::common_interface::AuboException
02732     *
02733     * @par Python function prototype
02734     * getPedestalHardwareVersion(self: pyaubo_sdk.RobotState) -> int
02735     *
02736     * @par Lua function prototype
02737     * getPedestalHardwareVersion() -> number
02738     *
02739     * @par Lua example
02740     * PedestalHardwareVersion = getPedestalHardwareVersion()
02741     *
02742     * @par JSON-RPC request example
02743     * {"jsonrpc":"2.0","method":"rob1.RobotState.getPedestalHardwareVersion","params":[],"id":1}
02744     *
02745     * @par JSON-RPC response example
02746     * {"id":1,"jsonrpc":"2.0","result":1007000}
02747     * \endenglish
02748     */
02749    int getPedestalHardwareVersion();
02750
02751    /**
02752     * @ingroup RobotState
02753     * \chinese
02754     *
02755     *
02756     * @return
02757     *
02758     * @throws arcs::common_interface::AuboException
02759     *
02760     * @par Python
02761     * getJointTargetPositions(self: pyaubo_sdk.RobotState) -> List[float]
02762     *
02763     * @par Lua
02764     * getJointTargetPositions() -> table
02765     *
02766     * @par Lua
02767     * JointTargetPositions = getJointTargetPositions()
02768     *
02769     * @par JSON-RPC
02770     * {"jsonrpc":"2.0","method":"rob1.RobotState.getJointTargetPositions","params":[],"id":1}
02771     *
02772     * @par JSON-RPC
02773     * {"id":1,"jsonrpc":"2.0","result":[0.0,-
      0.2619944355631239,1.7418124015308052,0.4330219266665035,1.5707963267948966,0.0]}
02774     * \endchinese
02775     * \english
02776     * Get the target joint positions (angles) of the manipulator
02777     *
02778     * @return Target joint positions (angles) of the manipulator
02779     *
02780     * @throws arcs::common_interface::AuboException
02781     *
02782     * @par Python function prototype
02783     * getJointTargetPositions(self: pyaubo_sdk.RobotState) -> List[float]
02784     *
02785     * @par Lua function prototype
02786     * getJointTargetPositions() -> table
02787     *
02788     * @par Lua example
02789     * JointTargetPositions = getJointTargetPositions()
02790     *
02791     * @par JSON-RPC request example
02792     * {"jsonrpc":"2.0","method":"rob1.RobotState.getJointTargetPositions","params":[],"id":1}
02793     *
02794     * @par JSON-RPC response example
02795     * {"id":1,"jsonrpc":"2.0","result":[0.0,-
      0.2619944355631239,1.7418124015308052,0.4330219266665035,1.5707963267948966,0.0]}
02796     * \endenglish
02797     */
02798    std::vector<double> getJointTargetPositions();
02799
02800    /**
02801     * @ingroup RobotState
02802     * \chinese
02803     *
02804     *
02805     * @return
02806     *
02807     * @throws arcs::common_interface::AuboException
02808     *
02809     * @par Python
```

```
02810     * getJointTargetSpeeds(self: pyaubo_sdk.RobotState) -> List[float]
02811     *
02812     * @par Lua
02813     * getJointTargetSpeeds() -> table
02814     *
02815     * @par Lua
02816     * JointTargetSpeeds = getJointTargetSpeeds()
02817     *
02818     * @par JSON-RPC
02819     * {"jsonrpc":"2.0","method":"rob1.RobotState.getJointTargetSpeeds","params":[],"id":1}
02820     *
02821     * @par JSON-RPC
02822     * {"id":1,"jsonrpc":"2.0","result":[0.0,0.0002422710150939977 3,0.0016521760307419697,0.0026521060731088397,0.0,0.0]}
02823     * \endchinese
02824     * \english
02825     * Get the target joint speeds of the manipulator
02826     *
02827     * @return Target joint speeds of the manipulator
02828     *
02829     * @throws arcs::common_interface::AuboException
02830     *
02831     * @par Python function prototype
02832     * getJointTargetSpeeds(self: pyaubo_sdk.RobotState) -> List[float]
02833     *
02834     * @par Lua function prototype
02835     * getJointTargetSpeeds() -> table
02836     *
02837     * @par Lua example
02838     * JointTargetSpeeds = getJointTargetSpeeds()
02839     *
02840     * @par JSON-RPC request example
02841     * {"jsonrpc":"2.0","method":"rob1.RobotState.getJointTargetSpeeds","params":[],"id":1}
02842     *
02843     * @par JSON-RPC response example
02844     * {"id":1,"jsonrpc":"2.0","result":[0.0,0.0002422710150939977 3,0.0016521760307419697,0.0026521060731088397,0.0,0.0]}
02845     * \endenglish
02846     */
02847    std::vector<double> getJointTargetSpeeds();
02848
02849    /**
02850     * @ingroup RobotState
02851     * \chinese
02852     *
02853     *
02854     * @return
02855     *
02856     * @throws arcs::common_interface::AuboException
02857     *
02858     * @par Python
02859     * getJointTargetAccelerations(self: pyaubo_sdk.RobotState) -> List[float]
02860     *
02861     * @par Lua
02862     * getJointTargetAccelerations() -> table
02863     *
02864     * @par Lua
02865     * JointTargetAccelerations = getJointTargetAccelerations()
02866     *
02867     * @par JSON-RPC
02868     * {"jsonrpc":"2.0","method":"rob1.RobotState.getJointTargetAccelerations","params":[],"id":1}
02869     *
02870     * @par JSON-RPC
02871     * {"id":1,"jsonrpc":"2.0","result":[0.0,-0.6737932929246071,-12.610253240108449,0.0,0.0,0.0]}
02872     * \endchinese
02873     * \english
02874     * Get the target joint accelerations of the manipulator
02875     *
02876     * @return Target joint accelerations of the manipulator
02877     *
02878     * @throws arcs::common_interface::AuboException
02879     *
02880     * @par Python function prototype
02881     * getJointTargetAccelerations(self: pyaubo_sdk.RobotState) -> List[float]
02882     *
02883     * @par Lua function prototype
02884     * getJointTargetAccelerations() -> table
02885     *
02886     * @par Lua example
02887     * JointTargetAccelerations = getJointTargetAccelerations()
02888     *
02889     * @par JSON-RPC request example
02890     * {"jsonrpc":"2.0","method":"rob1.RobotState.getJointTargetAccelerations","params":[],"id":1}
02891     *
02892     * @par JSON-RPC response example
02893     * {"id":1,"jsonrpc":"2.0","result":[0.0,-0.6737932929246071,-12.610253240108449,0.0,0.0,0.0]}
02894     * \endenglish
02895     */
02896    std::vector<double> getJointTargetAccelerations();
```

```
02897
02898     /**
02899      * @ingroup RobotState
02900      * \chinese
02901      *
02902      *
02903      * @return
02904      *
02905      * @throws arcs::common_interface::AuboException
02906      *
02907      * @par Python
02908      * getJointTargetTorques(self: pyaubo_sdk.RobotState) -> List[float]
02909      *
02910      * @par Lua
02911      * getJointTargetTorques() -> table
02912      *
02913      * @par Lua
02914      * JointTargetTorques = getJointTargetTorques()
02915      *
02916      * @par JSON-RPC
02917      * {"jsonrpc":"2.0","method":"rob1.RobotState.getJointTargetTorques","params":[],"id":1}
02918      *
02919      * @par JSON-RPC
02920      * {"id":1,"jsonrpc":"2.0","result":[0.0,0.0,0.0,0.0,0.0,0.0]}
02921      * \endchinese
02922      * \english
02923      * Get the target joint torques of the manipulator
02924      *
02925      * @return Target joint torques of the manipulator
02926      *
02927      * @throws arcs::common_interface::AuboException
02928      *
02929      * @par Python function prototype
02930      * getJointTargetTorques(self: pyaubo_sdk.RobotState) -> List[float]
02931      *
02932      * @par Lua function prototype
02933      * getJointTargetTorques() -> table
02934      *
02935      * @par Lua example
02936      * JointTargetTorques = getJointTargetTorques()
02937      *
02938      * @par JSON-RPC request example
02939      * {"jsonrpc":"2.0","method":"rob1.RobotState.getJointTargetTorques","params":[],"id":1}
02940      *
02941      * @par JSON-RPC response example
02942      * {"id":1,"jsonrpc":"2.0","result":[0.0,0.0,0.0,0.0,0.0,0.0]}
02943      * \endenglish
02944      */
02945     std::vector<double> getJointTargetTorques();
02946
02947     /**
02948      * @ingroup RobotState
02949      * \chinese
02950      *
02951      *
02952      * @return
02953      *
02954      * @throws arcs::common_interface::AuboException
02955      *
02956      * @par Python
02957      * getJointTargetCurrents(self: pyaubo_sdk.RobotState) -> List[float]
02958      *
02959      * @par Lua
02960      * getJointTargetCurrents() -> table
02961      *
02962      * @par Lua
02963      * JointTargetCurrents = getJointTargetCurrents()
02964      *
02965      * @par JSON-RPC
02966      * {"jsonrpc":"2.0","method":"rob1.RobotState.getJointTargetCurrents","params":[],"id":1}
02967      *
02968      * @par JSON-RPC
02969      * {"id":1,"jsonrpc":"2.0","result":[0.0,0.0,0.0,0.0,0.0,0.0]}
02970      * \endchinese
02971      * \english
02972      * Get the target joint currents of the manipulator
02973      *
02974      * @return Target joint currents of the manipulator
02975      *
02976      * @throws arcs::common_interface::AuboException
02977      *
02978      * @par Python function prototype
02979      * getJointTargetCurrents(self: pyaubo_sdk.RobotState) -> List[float]
02980      *
02981      * @par Lua function prototype
02982      * getJointTargetCurrents() -> table
02983      *
```

```
02984     * @par Lua example
02985     * JointTargetCurrents = getJointTargetCurrents()
02986     *
02987     * @par JSON-RPC request example
02988     * {"jsonrpc":"2.0","method":"rob1.RobotState.getJointTargetCurrents","params":[],"id":1}
02989     *
02990     * @par JSON-RPC response example
02991     * {"id":1,"jsonrpc":"2.0","result":[0.0,0.0,0.0,0.0,0.0,0.0]}
02992     * \endenglish
02993     */
02994    std::vector<double> getJointTargetCurrents();
02995
02996    /**
02997     * @ingroup RobotState
02998     * \chinese
02999     *
03000     *
03001     *
03002     * @return          false;     true
03003     *
03004     * @throws arcs::common_interface::AuboException
03005     *
03006     * @par Python
03007     * isTeachPendantEnabled(self: pyaubo_sdk.RobotState) -> bool
03008     *
03009     * @par Lua
03010     * isTeachPendantEnabled() -> boolean
03011     *
03012     * @par Lua
03013     * TeachPendantEnabled = isTeachPendantEnabled()
03014     *
03015     * @par JSON-RPC
03016     * {"jsonrpc":"2.0","method":"rob1.RobotState.isTeachPendantEnabled","params":[],"id":1}
03017     *
03018     * @par JSON-RPC
03019     * {"id":1,"jsonrpc":"2.0","result":true}
03020     * \endchinese
03021     * \english
03022     * Get whether the teach pendant is enabled.
03023     * Indicates whether the enable button of teach pendant is in the pressed
03024     * state.
03025     *
03026     * @return Returns false if the enable button of teach pendant is pressed;
03027     * otherwise returns true
03028     *
03029     * @throws arcs::common_interface::AuboException
03030     *
03031     * @par Python function prototype
03032     * isTeachPendantEnabled(self: pyaubo_sdk.RobotState) -> bool
03033     *
03034     * @par Lua function prototype
03035     * isTeachPendantEnabled() -> boolean
03036     *
03037     * @par Lua example
03038     * TeachPendantEnabled = isTeachPendantEnabled()
03039     *
03040     * @par JSON-RPC request example
03041     * {"jsonrpc":"2.0","method":"rob1.RobotState.isTeachPendantEnabled","params":[],"id":1}
03042     *
03043     * @par JSON-RPC response example
03044     * {"id":1,"jsonrpc":"2.0","result":true}
03045     * \endenglish
03046     */
03047    bool isTeachPendantEnabled();
03048
03049    /**
03050     * @ingroup RobotState
03051     * \chinese
03052     *
03053     *
03054     * @return      false;     true
03055     *
03056     * @throws arcs::common_interface::AuboException
03057     *
03058     * @par Python
03059     * isToolFlangeEnabled(self: pyaubo_sdk.RobotState) -> bool
03060     *
03061     * @par Lua
03062     * isToolFlangeEnabled() -> boolean
03063     *
03064     * @par Lua
03065     * toolEnabled = isToolFlangeEnabled()
03066     *
03067     * @par JSON-RPC
03068     * {"jsonrpc":"2.0","method":"rob1.RobotState.isToolFlangeEnabled","params":[],"id":1}
03069     *
03070     * @par JSON-RPC
```

```
03071     * {"id":1,"jsonrpc":"2.0","result":true}
03072     * \endchinese
03073     * \english
03074     * Get whether the tool flange is enabled.
03075     *
03076     * @return Returns false if the tool flange is disabled; otherwise returns
03077     * true
03078     *
03079     * @throws arcs::common_interface::AuboException
03080     *
03081     * @par Python function prototype
03082     * isToolFlangeEnabled(self: pyaubo_sdk.RobotState) -> bool
03083     *
03084     * @par Lua function prototype
03085     * isToolFlangeEnabled() -> boolean
03086     *
03087     * @par Lua example
03088     * toolEnabled = isToolFlangeEnabled()
03089     *
03090     * @par JSON-RPC request example
03091     * {"jsonrpc":"2.0","method":"rob1.RobotState.isToolFlangeEnabled","params":[],"id":1}
03092     *
03093     * @par JSON-RPC response example
03094     * {"id":1,"jsonrpc":"2.0","result":true}
03095     * \endenglish
03096     */
03097    bool isToolFlangeEnabled();
03098
03099    /**
03100     * @ingroup RobotState
03101     * \chinese
03102     *
03103     *
03104     * @return
03105     *
03106     * @throws arcs::common_interface::AuboException
03107     *
03108     * @par Python
03109     * getControlBoxTemperature(self: pyaubo_sdk.RobotState) -> float
03110     *
03111     * @par Lua
03112     * getControlBoxTemperature() -> number
03113     *
03114     * @par Lua
03115     * ControlBoxTemperature = getControlBoxTemperature()
03116     *
03117     * @par JSON-RPC
03118     * {"jsonrpc":"2.0","method":"rob1.RobotState.getControlBoxTemperature","params":[],"id":1}
03119     *
03120     * @par JSON-RPC
03121     * {"id":1,"jsonrpc":"2.0","result":25.0}
03122     * \endchinese
03123     * \english
03124     * Get the control box temperature
03125     *
03126     * @return Control box temperature
03127     *
03128     * @throws arcs::common_interface::AuboException
03129     *
03130     * @par Python function prototype
03131     * getControlBoxTemperature(self: pyaubo_sdk.RobotState) -> float
03132     *
03133     * @par Lua function prototype
03134     * getControlBoxTemperature() -> number
03135     *
03136     * @par Lua example
03137     * ControlBoxTemperature = getControlBoxTemperature()
03138     *
03139     * @par JSON-RPC request example
03140     * {"jsonrpc":"2.0","method":"rob1.RobotState.getControlBoxTemperature","params":[],"id":1}
03141     *
03142     * @par JSON-RPC response example
03143     * {"id":1,"jsonrpc":"2.0","result":25.0}
03144     * \endenglish
03145     */
03146    double getControlBoxTemperature();
03147
03148    /**
03149     * @ingroup RobotState
03150     * \chinese
03151     *
03152     *
03153     * @return
03154     *
03155     * @throws arcs::common_interface::AuboException
03156     *
03157     * @par Python
```

```
03158     * getControlBoxHumidity(self: pyaubo_sdk.RobotState) -> float
03159     *
03160     * @par Lua
03161     * getControlBoxHumidity() -> number
03162     *
03163     * @par Lua
03164     * ControlBoxHumidity = getControlBoxHumidity()
03165     *
03166     * @par JSON-RPC
03167     * {"jsonrpc":"2.0","method":"rob1.RobotState.getControlBoxHumidity","params":[],"id":1}
03168     *
03169     * @par JSON-RPC
03170     * {"id":1,"jsonrpc":"2.0","result":20.0}
03171     * \endchinese
03172     * \english
03173     * Get the control box humidity
03174     *
03175     * @return Control box humidity
03176     *
03177     * @throws arcs::common_interface::AuboException
03178     *
03179     * @par Python function prototype
03180     * getControlBoxHumidity(self: pyaubo_sdk.RobotState) -> float
03181     *
03182     * @par Lua function prototype
03183     * getControlBoxHumidity() -> number
03184     *
03185     * @par Lua example
03186     * ControlBoxHumidity = getControlBoxHumidity()
03187     *
03188     * @par JSON-RPC request example
03189     * {"jsonrpc":"2.0","method":"rob1.RobotState.getControlBoxHumidity","params":[],"id":1}
03190     *
03191     * @par JSON-RPC response example
03192     * {"id":1,"jsonrpc":"2.0","result":20.0}
03193     * \endenglish
03194     */
03195    double getControlBoxHumidity();
03196
03197    /**
03198     * @ingroup RobotState
03199     * \chinese
03200     *
03201     *
03202     * @return
03203     *
03204     * @throws arcs::common_interface::AuboException
03205     *
03206     * @par Python
03207     * getMainVoltage(self: pyaubo_sdk.RobotState) -> float
03208     *
03209     * @par Lua
03210     * getMainVoltage() -> number
03211     *
03212     * @par Lua
03213     * MainVoltage = getMainVoltage()
03214     *
03215     * @par JSON-RPC
03216     * {"jsonrpc":"2.0","method":"rob1.RobotState.getMainVoltage","params":[],"id":1}
03217     *
03218     * @par JSON-RPC
03219     * {"id":1,"jsonrpc":"2.0","result":52.75}
03220     * \endchinese
03221     * \english
03222     * Get the main bus voltage
03223     *
03224     * @return Main bus voltage
03225     *
03226     * @throws arcs::common_interface::AuboException
03227     *
03228     * @par Python function prototype
03229     * getMainVoltage(self: pyaubo_sdk.RobotState) -> float
03230     *
03231     * @par Lua function prototype
03232     * getMainVoltage() -> number
03233     *
03234     * @par Lua example
03235     * MainVoltage = getMainVoltage()
03236     *
03237     * @par JSON-RPC request example
03238     * {"jsonrpc":"2.0","method":"rob1.RobotState.getMainVoltage","params":[],"id":1}
03239     *
03240     * @par JSON-RPC response example
03241     * {"id":1,"jsonrpc":"2.0","result":52.75}
03242     * \endenglish
03243     */
03244    double getMainVoltage();
```

```
03245
03246     /**
03247      * @ingroup RobotState
03248      * \chinese
03249      *
03250      *
03251      * @return
03252      *
03253      * @throws arcs::common_interface::AuboException
03254      *
03255      * @par Python
03256      * getMainCurrent(self: pyaubo_sdk.RobotState) -> float
03257      *
03258      * @par Lua
03259      * getMainCurrent() -> number
03260      *
03261      * @par Lua
03262      * MainCurrent = getMainCurrent()
03263      *
03264      * @par JSON-RPC
03265      * {"jsonrpc":"2.0","method":"rob1.RobotState.getMainCurrent","params":[],"id":1}
03266      *
03267      * @par JSON-RPC
03268      * {"id":1,"jsonrpc":"2.0","result":0.3204345703125}
03269      * \endchinese
03270      * \english
03271      * Get the main bus current
03272      *
03273      * @return Main bus current
03274      *
03275      * @throws arcs::common_interface::AuboException
03276      *
03277      * @par Python function prototype
03278      * getMainCurrent(self: pyaubo_sdk.RobotState) -> float
03279      *
03280      * @par Lua function prototype
03281      * getMainCurrent() -> number
03282      *
03283      * @par Lua example
03284      * MainCurrent = getMainCurrent()
03285      *
03286      * @par JSON-RPC request example
03287      * {"jsonrpc":"2.0","method":"rob1.RobotState.getMainCurrent","params":[],"id":1}
03288      *
03289      * @par JSON-RPC response example
03290      * {"id":1,"jsonrpc":"2.0","result":0.3204345703125}
03291      * \endenglish
03292      */
03293     double getMainCurrent();
03294
03295     /**
03296      * @ingroup RobotState
03297      * \chinese
03298      *
03299      *
03300      * @return
03301      *
03302      * @throws arcs::common_interface::AuboException
03303      *
03304      * @par Python
03305      * getRobotVoltage(self: pyaubo_sdk.RobotState) -> float
03306      *
03307      * @par Lua
03308      * getRobotVoltage() -> number
03309      *
03310      * @par Lua
03311      * RobotVoltage = getRobotVoltage()
03312      *
03313      * @par JSON-RPC
03314      * {"jsonrpc":"2.0","method":"rob1.RobotState.getRobotVoltage","params":[],"id":1}
03315      *
03316      * @par JSON-RPC
03317      * {"id":1,"jsonrpc":"2.0","result":52.75}
03318      * \endchinese
03319      * \english
03320      * Get the robot voltage
03321      *
03322      * @return Robot voltage
03323      *
03324      * @throws arcs::common_interface::AuboException
03325      *
03326      * @par Python function prototype
03327      * getRobotVoltage(self: pyaubo_sdk.RobotState) -> float
03328      *
03329      * @par Lua function prototype
03330      * getRobotVoltage() -> number
03331      *
```

```
03332     * @par Lua example
03333     * RobotVoltage = getRobotVoltage()
03334     *
03335     * @par JSON-RPC request example
03336     * {"jsonrpc":"2.0","method":"rob1.RobotState.getRobotVoltage","params":[],"id":1}
03337     *
03338     * @par JSON-RPC response example
03339     * {"id":1,"jsonrpc":"2.0","result":52.75}
03340     * \endenglish
03341     */
03342    double getRobotVoltage();
03343
03344    /**
03345     * @ingroup RobotState
03346     * \chinese
03347     *
03348     *
03349     * @return
03350     *
03351     * @throws arcs::common_interface::AuboException
03352     *
03353     * @par Python
03354     * getRobotCurrent(self: pyaubo_sdk.RobotState) -> float
03355     *
03356     * @par Lua
03357     * getRobotCurrent() -> number
03358     *
03359     * @par Lua
03360     * RobotCurrent = getRobotCurrent()
03361     *
03362     * @par JSON-RPC
03363     * {"jsonrpc":"2.0","method":"rob1.RobotState.getRobotCurrent","params":[],"id":1}
03364     *
03365     * @par JSON-RPC
03366     * {"id":1,"jsonrpc":"2.0","result":0.3204345703125}
03367     * \endchinese
03368     * \english
03369     * Get the robot current
03370     *
03371     * @return Robot current
03372     *
03373     * @throws arcs::common_interface::AuboException
03374     *
03375     * @par Python function prototype
03376     * getRobotCurrent(self: pyaubo_sdk.RobotState) -> float
03377     *
03378     * @par Lua function prototype
03379     * getRobotCurrent() -> number
03380     *
03381     * @par Lua example
03382     * RobotCurrent = getRobotCurrent()
03383     *
03384     * @par JSON-RPC request example
03385     * {"jsonrpc":"2.0","method":"rob1.RobotState.getRobotCurrent","params":[],"id":1}
03386     *
03387     * @par JSON-RPC response example
03388     * {"id":1,"jsonrpc":"2.0","result":0.3204345703125}
03389     * \endenglish
03390     */
03391    double getRobotCurrent();
03392
03393    /**
03394     * @ingroup RobotState
03395     * \chinese
03396     *
03397     *
03398     * @return
03399     *
03400     * @throws arcs::common_interface::AuboException
03401     *
03402     * @par Python
03403     * getPedestalBrakeResTemp(self: pyaubo_sdk.RobotState) -> float
03404     *
03405     * @par Lua
03406     * getPedestalBrakeResTemp() -> number
03407     *
03408     * @par Lua
03409     * PedestalBrakeResTemp = getPedestalBrakeResTemp()
03410     *
03411     * @par JSON-RPC
03412     * {"jsonrpc":"2.0","method":"rob1.RobotState.getPedestalBrakeResTemp","params":[],"id":1}
03413     *
03414     * @par JSON-RPC
03415     * {"id":1,"jsonrpc":"2.0","result":23.1111}
03416     * \endchinese
03417     * \english
03418     * Get the Pedestal Brake Res Temp
```

```
03419     *
03420     * @return Pedestal Brake Res Temp
03421     *
03422     * @throws arcs::common_interface::AuboException
03423     *
03424     * @par Python function prototype
03425     * getPedestalBrakeResTemp(self: pyaubo_sdk.RobotState) -> float
03426     *
03427     * @par Lua function prototype
03428     * getPedestalBrakeResTemp() -> number
03429     *
03430     * @par Lua example
03431     * PedestalBrakeResTemp = getPedestalBrakeResTemp()
03432     *
03433     * @par JSON-RPC request example
03434     * {”jsonrpc”:”2.0”,”method”:”rob1.RobotState.getPedestalBrakeResTemp”,”params”:[],”id”:1}
03435     *
03436     * @par JSON-RPC response example
03437     * {”id”:1,”jsonrpc”:”2.0”,”result”:24.235}
03438     * \endenglish
03439     */
03440    double getPedestalBrakeResTemp();
03441
03442    /**
03443     * @ingroup RobotState
03444     * \chinese
03445     *
03446     *
03447     * @return
03448     *
03449     * @throws arcs::common_interface::AuboException
03450     *
03451     * @par JSON-RPC
03452     * {”jsonrpc”:”2.0”,”method”:”rob1.RobotState.getSlowDownLevel”,”params”:[],”id”:1}
03453     *
03454     * @par JSON-RPC
03455     * {”id”:1,”jsonrpc”:”2.0”,”result”:0}
03456     * \endchinese
03457     * \english
03458     * Get the robot slow down level
03459     *
03460     * @return Robot slow down level
03461     *
03462     * @throws arcs::common_interface::AuboException
03463     *
03464     * @par JSON-RPC request example
03465     * {”jsonrpc”:”2.0”,”method”:”rob1.RobotState.getSlowDownLevel”,”params”:[],”id”:1}
03466     *
03467     * @par JSON-RPC response example
03468     * {”id”:1,”jsonrpc”:”2.0”,”result”:0}
03469     * \endenglish
03470     */
03471    int getSlowDownLevel();
03472
03473    /**
03474     * @ingroup RobotState
03475     * \chinese
03476     *
03477     *
03478     * @param name        selectTcpForceSensor
03479     * @return       true;       false
03480     *
03481     * @throws arcs::common_interface::AuboException
03482     *
03483     * @par JSON-RPC
03484     * {”jsonrpc”:”2.0”,”method”:”rob1.RobotState.getTcpForceSensorStatus”,”params”:[”tool.KWR75A”],”id”:1}
03485     *
03486     * @par JSON-RPC
03487     * {”id”:1,”jsonrpc”:”2.0”,”result”:0}
03488     * \endchinese
03489     * \english
03490     * Get the communication status of the tool force sensor
03491     *
03492     * @param name  force sensor name, it is consistent with the parameters of
03493     * ’selectTcpForceSensor’
03494     * @return Returns true if communication is normal; otherwise returns false
03495     *
03496     * @throws arcs::common_interface::AuboException
03497     *
03498     * @par JSON-RPC request example
03499     * {”jsonrpc”:”2.0”,”method”:”rob1.RobotState.getSlowDownLevel”,”params”:[”tool.KWR75A”],”id”:1}
03500     *
03501     * @par JSON-RPC response example
03502     * {”id”:1,”jsonrpc”:”2.0”,”result”:0}
03503     * \endenglish
03504     */
03505    bool getTcpForceSensorStatus(const std::string &name);
```

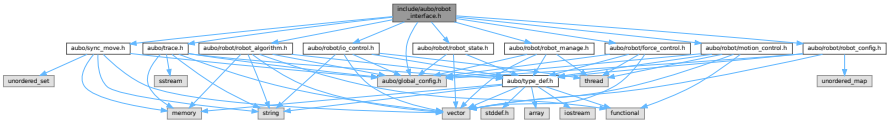
```
03506
03507 protected:
03508     void *d_;
03509 };
03510 using RobotStatePtr = std::shared_ptr<RobotState>;
03511 } // namespace common_interface
03512 } // namespace arcs
03513 #endif // AUBO_SDK_ROBOT_STATE_INTERFACE_H
```

12.34 include/aubo/robot_interface.h File Reference

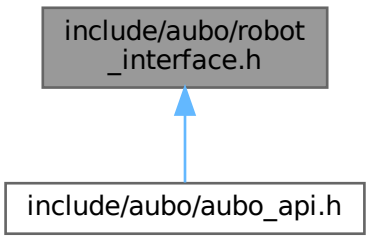
API

```
#include <aubo/sync_move.h>
#include <aubo/trace.h>
#include <aubo/robot/motion_control.h>
#include <aubo/robot/force_control.h>
#include <aubo/robot/io_control.h>
#include <aubo/robot/robot_algorithm.h>
#include <aubo/robot/robot_state.h>
#include <aubo/robot/robot_manage.h>
#include <aubo/robot/robot_config.h>
#include <aubo/global_config.h>
```

Include dependency graph for robot_interface.h:



This graph shows which files directly or indirectly include this file:



Classes

- class arcs::common_interface::RobotInterface

Namespaces

- namespace arcs
- namespace arcs::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
```

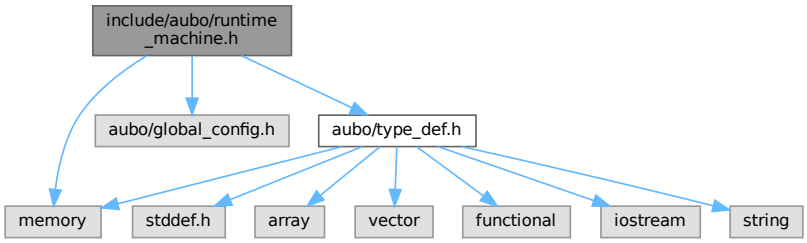
```
00227     */
00228     SyncMovePtr getSyncMove();
00229
00230     /**
00231      * @ingroup RobotInterface
00232      * @ref RobotAlgorithm
00233      * \chinese
00234      *
00235      *
00236      * @return RobotAlgorithmPtr
00237      *
00238      * @par Python
00239      * getRobotAlgorithm(self: 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
```

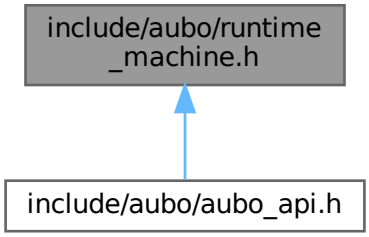
12.36 include/aubo/runtime_machine.h File Reference

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 breakpoints.

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 state 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);
```

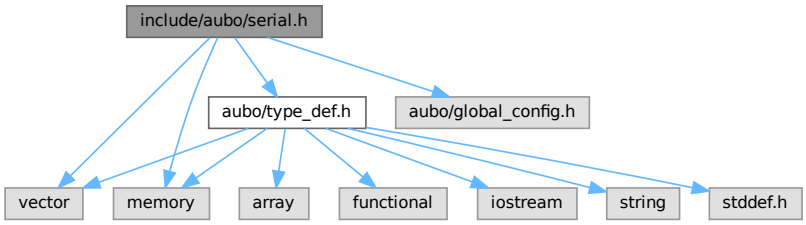
```
01374
01375     /**
01376      * @ingroup RuntimeMachine
01377      * \chinese
01378      *
01379      *
01380      * @param name
01381      * @return
01382      *
01383      * @par Python
01384      * timerReset(self: pyaubo_sdk.RuntimeMachine, arg0: str) -> int
01385      *
01386      * @par Lua
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
```

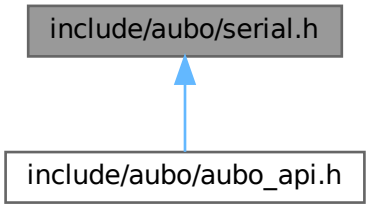
```
01548     * {"id":1,"jsonrpc":"2.0","result":0}
01549     *
01550     * \endchinese
01551     * \english
01552     * End configuring trigger
01553     *
01554     * @return
01555     *
01556     * @par JSON-RPC Request Example
01557     * {"jsonrpc":"2.0","method":"RuntimeMachine.triggEnd","params":[],"id":1}
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>
```

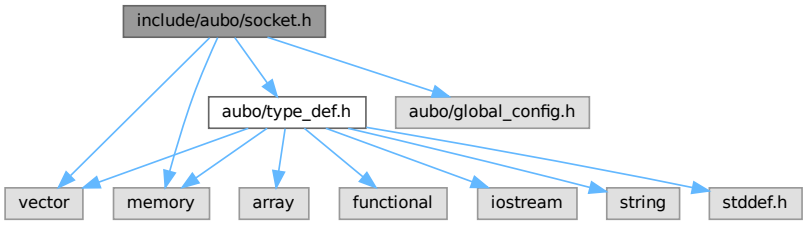
```
00342     *
00343     * @param str
00344     * @param serial_name
00345     * @return
00346     *
00347     * @par Python
00348     * serialSendString(self: pyaubo_sdk.Serial, arg0: str, arg1: str) -> int
00349     *
00350     * @par Lua
00351     * serialSendString(str: string, serial_name: string) -> 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

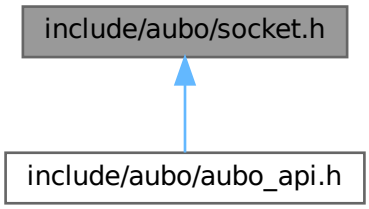
socket

```
#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     *
```

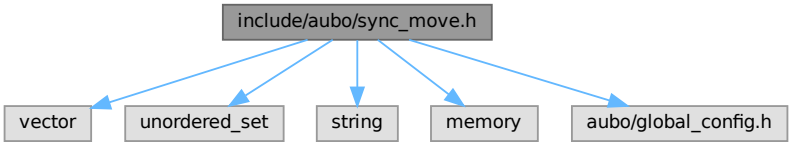
```
00431     *
00432     *
00433     * @param value
00434     * @param socket_name
00435     * @return
00436     *
00437     * @par Python
00438     * socketSendByte(self: pyaubo_sdk.Socket, arg0: str, arg1: str) -> int
00439     *
00440     * @par Lua
00441     * socketSendByte(value: string, socket_name: 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
```

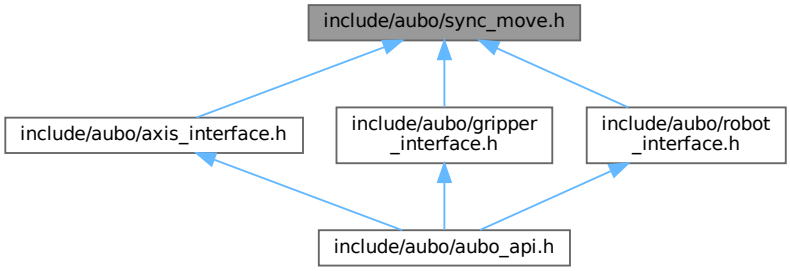
```
00605      * \chinese
00606      *    char
00607      *
00608      * @param is_check
00609      * @param str       char
00610      * @param socket_name
00611      * @return
00612      *
00613      * @par Python
00614      * socketSendAllString(self: pyaubo_sdk.Socket, arg0: bool, arg1: List[str], 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 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
    zones.
00114     * This means that move instructions with the same ID must either all have corner zones, or all have stop points.
00115     * 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.
00116     * Synchronously executed move instructions can have corner zones of different sizes (e.g. one uses z10 and one uses z50).
00117     *
00118     * @param id
00119     * @return
00120     *
00121     * @par Python prototype
00122     * syncMoveSegment(self: pyaubo_sdk.SyncMove, arg0: int) -> bool
00123     *
00124     * @par Lua prototype
00125     * syncMoveSegment(id: number) -> boolean
00126     * @endcoe
00127     * \endenglish
00128     */
00129    bool syncMoveSegment(int id);
00130
00131    /**
00132     * @ingroup SyncMove
00133     * \chinese
00134     * syncMoveOff
00135     *
00136     * syncMoveOff                syncMoveOff
00137     *            syncMoveOff
00138     *
00139     * @param syncident
00140     * @return
00141     *
00142     * @par Python
00143     * syncMoveOff(self: pyaubo_sdk.SyncMove, arg0: str) -> int
00144     *
00145     * @par Lua
00146     * syncMoveOff(syncident: string) -> nil
00147     * @endcoe
00148     * \endchinese
00149     * \english
00150     * syncMoveOff is used to end synchronized movement mode.
00151     *
00152     * A syncMoveOff instruction will wait for the other task programs. When
00153     * all task programs have reached the syncMoveOff, they will continue
00154     * their execution in unsynchronized mode.
00155     * A stop point must be programmed before the syncMoveOff instruction.
00156     *
00157     * @param syncident
00158     * @return
00159     *
00160     * @par Python prototype
00161     * syncMoveOff(self: pyaubo_sdk.SyncMove, arg0: str) -> int
00162     *
00163     * @par Lua prototype
00164     * syncMoveOff(syncident: string) -> nil
00165     * @endcoe
00166     * \endenglish
00167     */
```

```
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     *
00625     *    MotionPlus
00626     *
00627     * @param name:
00628     *
00629     * @return
00630     * \endchinese
00631     * \english
00632     * Returns a list of immediate child object frame names. Parent-child
00633     * relationships are defined by world model attachments. If being used with
00634     * MotionPlus, the child objects may also be an axis group or an axis.
00635     *
00636     * @param name: the name of the parent object.
00637     *
00638     * @return a list of immediate child object frame names
00639     * \endenglish
00640     */
00641    std::vector<std::string> frameGetChildren(const std::string &name);
00642
00643    /**
00644     * @ingroup SyncMove
00645     * \chinese
00646     *            name      ref_frame     pose
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,
       q_limits, and axis_index, respectively.
00741    * The pose parameter is typically obtained from a calibration process when the external axis is commissioned.
00742    * 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.
00743    *
00744    * @param 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.
00745    * @param 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.
00746    * @param 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.
00747    * @param 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.
00748    * @param type: Axis type, 0 for rotary, 1 for linear.
00749    * @param v_limit: Maximum velocity.
00750    * @param a_limit: Maximum acceleration.
00751    * @param q_limits: Position limits.
00752    * @param axis_index: Axis index.
00753    * \endenglish
00754    */
00755   int axisGroupAddAxis(const std::string &group_name, const std::string &name,
00756                        const std::string &parent,
00757                        const std::vector<double> &pose);
00758
00759   /**
00760    * @ingroup SyncMove
00761    * \chinese
00762    *          pose
00763    *
00764    *
00765    *
00766    * @param name:
00767    * @param pose ( ):                0
00768    * \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
```

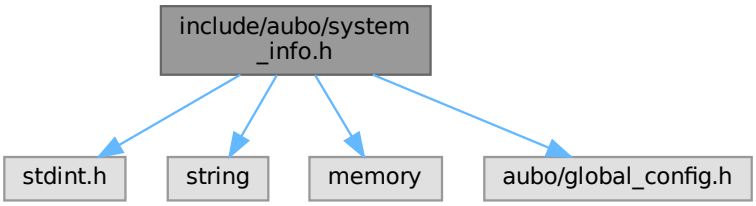
```
00856     * their external axis indices.
00857     * \endenglish
00858     */
00859    std::vector<double> axisGroupGetTargetPositions(
00860        const std::string &group_name);
00861
00862    /**
00863     * @ingroup SyncMove
00864     * \chinese
00865     *
00866     *     group_name
00867     *
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     *                              RTDE
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
```

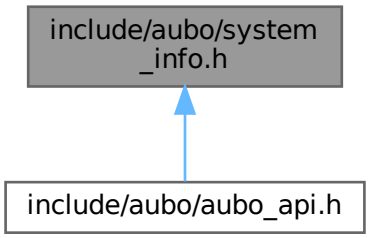
12.44 include/aubo/system_info.h File Reference

```
#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
```

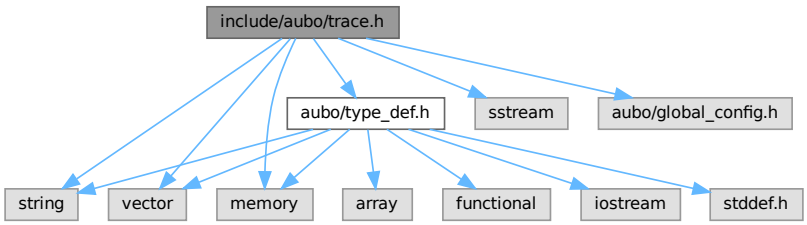
```
00257     *
00258     * @par JSON-RPC
00259     * {"jsonrpc":"2.0","method":"SystemInfo.getControlSoftwareVersionHash","params":[],"id":1}
00260     *
00261     * @par JSON-RPC
00262     * {"id":1,"jsonrpc":"2.0","result":"fa4f64a"}
00263     *
00264     * \endchinese
00265     * \english
00266     * 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
```

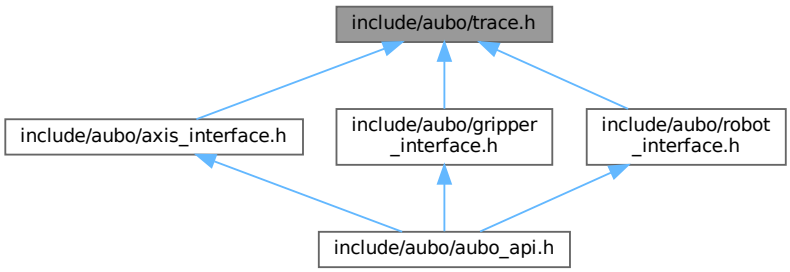
12.46 include/aubo/trace.h File Reference

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    \~engish @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 responose 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}
```

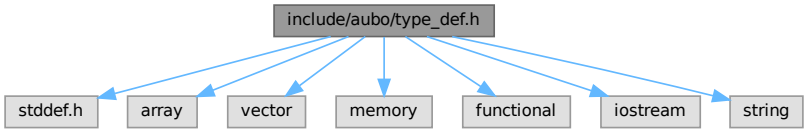
```
00140     *
00141     * @par JSON-RPC
00142     * {"id":1,"jsonrpc":"2.0","result":0}
00143     * \endchinese
00144     * \english
00145     * Send a popup request to the connected RTDE client
00146     *
00147     * @param level
00148     * @param title
00149     * @param msg
00150     * @param 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!",1],"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
```

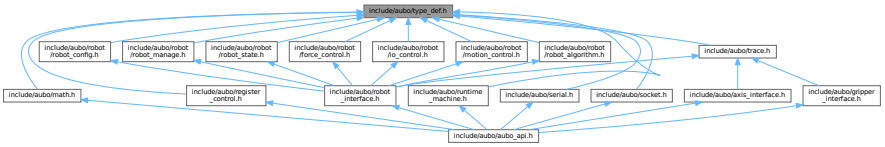
12.48 include/aubo/type_def.h File Reference

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

  Cartesion 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_HandleStateType_DECLARES
- #define ENUM_HandleModeType_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_D }
- enum class arcs::common_interface::AxisModeType : int { arcs::common_interface::ENUM_AxisModeType_DECI }
- enum class arcs::common_interface::HandleStateType : int { arcs::common_interface::ENUM_HandleStateType_D }
- enum class arcs::common_interface::HandleModeType : int { arcs::common_interface::ENUM_HandleModeType_ }
- enum class arcs::common_interface::SafetyModeType : int { arcs::common_interface::ENUM_SafetyModeType_D }

  Safety Mode.
- enum class arcs::common_interface::OperationalModeType : int { arcs::common_interface::ENUM_OperationalMo }

  Operational Mode.
- enum class arcs::common_interface::RobotControlModeType : int { arcs::common_interface::ENUM_RobotContro }

  Robot Control Mode.
- enum class arcs::common_interface::JointServoModeType : int { arcs::common_interface::ENUM_JointServoMode }

  Joint Servo Mode.
- enum class arcs::common_interface::JointStateType : int { arcs::common_interface::ENUM_JointStateType_DEC }

  Joint State.
- enum class arcs::common_interface::StandardOutputRunState : int { arcs::common_interface::ENUM_StandardOu }

  Standard Output Run State.

- enum class arcs::common_interface::StandardInputAction : int { arcs::common_interface::ENUM_StandardInputA }

  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_DECLA }
- 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_DEC }
- 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_INVAL_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 690 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 527 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 500 of file type_def.h.

12.48.2.6 ENUM_HandleModeType_DECLARES

```
#define ENUM_HandleModeType_DECLARES
```

Value:

```
ENUM_ITEM(None, 0, "    ") \
ENUM_ITEM(Default, 1, "S     ") \
ENUM_ITEM(MultiFuncG1, 2, "G       ") \
```

Definition at line 543 of file type_def.h.

12.48.2.7 ENUM_HandleStateType_DECLARES

```
#define ENUM_HandleStateType_DECLARES
```

Value:

```
ENUM_ITEM(Disconnected, 0, "   ") \
ENUM_ITEM(Init, 1, "   ") \
ENUM_ITEM(Connected, 2, "   ") \
ENUM_ITEM(Lock, 3, "  ") \
ENUM_ITEM(Fault, 4, "   ") \
```

Definition at line 536 of file type_def.h.

12.48.2.8 ENUM_ITEM [1/5]

```
#define ENUM_ITEM(
                  c,
                  n,
                  ...)
```

Value:

```
if (v == T::c) {          \
    return name + #c;     \
}
```

Definition at line 561 of file type_def.h.

12.48.2.9 ENUM_ITEM [2/5]

```
#define ENUM_ITEM(
                  c,
                  n,
                  ...)
```

Value:

```
c = n,
```

Definition at line 561 of file type_def.h.

12.48.2.10 ENUM_ITEM [3/5]

```
#define ENUM_ITEM(
                  n,
                  v,
                  s)
```

Value:

```
if (abs(retval) == v)  \
    index = n##_INDEX;
```

Definition at line 561 of file type_def.h.

12.48.2.11 ENUM_ITEM [4/5]

```
#define ENUM_ITEM(
                  n,
                  v,
                  s)
```

Value:

```
n##_INDEX,
```

Definition at line 561 of file type_def.h.

12.48.2.12 ENUM_ITEM [5/5]

```
#define ENUM_ITEM(
                  n,
                  v,
                  s)
```

Value:

```
s,
```

Definition at line 561 of file type_def.h.

12.48.2.13 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 411 of file type_def.h.

12.48.2.14 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,    " bootloader ,      (node bootloader state, pause all communication)") \
```

Definition at line 419 of file type_def.h.

12.48.2.15 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 393 of file type_def.h.

12.48.2.16 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 494 of file type_def.h.

12.48.2.17 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 401 of file type_def.h.

12.48.2.18 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 554 of file type_def.h.

12.48.2.19 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 349 of file type_def.h.

12.48.2.20 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 326 of file type_def.h.

12.48.2.21 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 548 of file type_def.h.

12.48.2.22 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 471 of file type_def.h.

12.48.2.23 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 364 of file type_def.h.

12.48.2.24 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 483 of file type_def.h.

12.48.2.25 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 426 of file type_def.h.

12.48.2.26 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, "       ") \
ENUM_ITEM(WaypointArrived, 17, "        ")
```

Definition at line 449 of file type_def.h.

12.48.2.27 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 505 of file type_def.h.

12.48.2.28 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 520 of file type_def.h.

12.48.2.29 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().

12.48.2.30 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().

12.48.2.31 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().

12.48.2.32 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].qdmax.begin(), params[i].qdmax.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_safetyguard_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     };                          ///< \~chinese     \~english inclination angle
00151     float tool_azimuth{ 0. }; ///< \~chinese      \~english azimuth angle
00152     std::vector<float> safety_home;
00153 
00154     /// \~chinese    IO          \~english Configurable IO input
00155     /// and output safety functions
00156     uint32_t safety_input_emergency_stop;
00157     uint32_t safety_input_safeguard_stop;
00158     uint32_t safety_input_safeguard_reset;
00159     uint32_t safety_input_auto_safeguard_stop;
00160     uint32_t safety_input_auto_safeguard_reset;
00161     uint32_t safety_input_three_position_switch;
00162     uint32_t safety_input_operational_mode;
00163     uint32_t safety_input_reduced_mode;
00164     uint32_t safety_input_handguide;
00165 
00166     uint32_t safety_output_emergency_stop;
00167     uint32_t safety_output_not_emergency_stop;
00168     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_safetyguard_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_IMPLETEMENT, 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 unmatch")                   \
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_UNKOWN_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_INVALID_FILE_PATH, 32,                                       \
00279             "Please verify the file path. It appears to be invalid.") \
00280     ENUM_ITEM(AUBO_INST_QUEUED, 100, "Instruction pused into queue succeed")    \
00281     ENUM_ITEM(AUBO_INTERNAL_ERR, 101, "Internal error caused by alg .etc.")     \
00282     ENUM_ITEM(AUBO_BADSTATE_THREAD_DETACHED, 200,                               \
00283             "Bad state: Operation not allowed on detached thread")              \
00284     ENUM_ITEM(AUBO_BADSTATE_THREAD_KILLED, 201,                                 \
00285             "Bad state: Operation not allowed on killed thread")                \
00286     ENUM_ITEM(AUBO_BADSTATE_TASK_NOT_FOUND, 202,                                \
00287             "Bad state: Specified task id does not exist in the task queue")\
00288     ENUM_ITEM(AUBO_BADSTATE_RTM_NOT_STARTED, 203,                               \
00289             "Bad state: RuntimeMachine has not been started yet")               \
00290     ENUM_ITEM(AUBO_BADSTATE_RTM_NOT_STOPPED, 204,                               \
00291             "Bad state: RuntimeMachine must be in Stopped state for this operation") \
00292     ENUM_ITEM(AUBO_BADSTATE_RTM_NOT_PAUSED, 205,                                \
00293             "Bad state: RuntimeMachine must be in Paused state for this operation") \
00294     ENUM_ITEM(AUBO_BADSTATE_RTM_ABORTING, 206, "Bad state: RuntimeMachine is in aborting state") \
00295     ENUM_ITEM(AUBO_BADSTATE_PSTOP, 207, "Bad state: Operation blocked by protective stop") \
00296     ENUM_ITEM(AUBO_BADSTATE_ROBOT_ESTOP, 208, "Bad state: Robot emergency stop triggered") \
00297     ENUM_ITEM(AUBO_BADSTATE_SYSTEM_ESTOP, 209, "Bad state: System emergency stop triggered") \
00298     ENUM_ITEM(AUBO_BADSTATE_INVALID_ROBOT_MODE, 210,                            \
00299             "Bad state: Robot is not in required operation mode")               \
00300     ENUM_ITEM(AUBO_BADSTATE_INVALID_SAFETY_MODE, 211,                           \
00301             "Bad state: Robot is not in required safety mode")                  \
00302     ENUM_ITEM(AUBO_BADSTATE_ROBOT_NOT_RUNNING, 212,                             \
00303             "Bad state: Robot must be in Running mode for this operation")  \
00304     ENUM_ITEM(AUBO_BADSTATE_ROBOT_NOT_POWERED_OFF, 213,                         \
00305             "Bad state: Robot must be in PowerOff mode for this operation") \
00306     ENUM_ITEM(AUBO_BADSTATE_SERIAL_OPEN_FAILED, 214, "Bad state: Failed to open serial device") \
00307     ENUM_ITEM(AUBO_BADSTATE_SERIAL_NOT_A_TERMINAL, 215,                         \
00308             "Bad state: Specified device is not a terminal")                    \
00309     ENUM_ITEM(AUBO_BADSTATE_SERIAL_CONFIG_FAILED, 216,                          \
00310             "Bad state: Failed to configure serial port parameters")            \
00311     ENUM_ITEM(AUBO_BADSTATE_KINEMATICS_COMPENSATE_FAILED, 217,                  \
00312             "Bad state: Failed to set kinematics compensation parameters")  \
00313     ENUM_ITEM(AUBO_BADSTATE_ROBOT_NOT_STEADY, 218, "Bad state: Robot is not in steady state") \
00314     ENUM_ITEM(AUBO_BADSTATE_FREE_DRIVE_ACTIVE, 219, "Bad state: Free-drive mode is active") \
00315     ENUM_ITEM(AUBO_BADSTATE_FORCE_CTRL_ACTIVE, 220, "Bad state: Force control mode is active") \
00316     ENUM_ITEM(AUBO_BADSTATE_SIMULATION_MODE_ACTIVE, 221,                        \
00317             "Bad state: Operation not allowed in simulation mode")              \
00318     ENUM_ITEM(AUBO_BADSTATE_MB_ERROR, 222, "Bad state: Modbus signal has error") \
00319     ENUM_ITEM(AUBO_BADSTATE_MB_TIMEOUT, 223, "Bad state: Modbus signal timeout") \
00320     ENUM_ITEM(AUBO_ERR_UNKOWN, 99999, "Unkown error occurred.")
00321
00322 /**
00323  * The RuntimeState enum
00324  *
00325  */
00326 #define ENUM_RuntimeState_DECLARES                      \
00327     ENUM_ITEM(Running, 0, "    ")                   \
00328     ENUM_ITEM(Retracting, 1, "  ")                  \
00329     ENUM_ITEM(Pausing, 2, "  ")                    \
00330     ENUM_ITEM(Paused, 3, "  ")                     \
00331     ENUM_ITEM(Stepping, 4, "    ")                 \
00332     ENUM_ITEM(Stopping, 5, "   (    )")           \
00333     ENUM_ITEM(Stopped, 6, "  ")                    \
00334     ENUM_ITEM(Aborting, 7, " (      )")
00335
00336 /**
00337  * \chinese
00338  * @brief The RobotModeType enum
00339  *
00340  *
00341  * \endchinese
```

```
00342  *
00343  * \english
00344  * @brief The RobotModeType enum
00345  *
00346  * Hardware related
00347  * \endenglish
00348  */
00349 #define ENUM_RobotModeType_DECLARES                                                    \
00350     ENUM_ITEM(NoController, -1,          ”       ,  aubo_control      NoController”) \
00351     ENUM_ITEM(Disconnected, 0,           ”       (          EtherCAT     )”)  \
00352     ENUM_ITEM(ConfirmSafety, 1,          ”       ,      ”)                          \
00353     ENUM_ITEM(Booting, 2,               ”       ”)                                  \
00354     ENUM_ITEM(PowerOff, 3,              ”       ”)                                    \
00355     ENUM_ITEM(PowerOn, 4,               ”       ,     ( ),       ”)          \
00356     ENUM_ITEM(Idle, 5,                ”     ,     ( ),     ,       ”)    \
00357     ENUM_ITEM(BrakeReleasing, 6,        ”     ,     ”)                            \
00358     ENUM_ITEM(BackDrive, 7,             ”       ,     ”)                        \
00359     ENUM_ITEM(Running, 8,               ”     ,     ,       ”)                  \
00360     ENUM_ITEM(Maintaince, 9,            ”   :       ”)                          \
00361     ENUM_ITEM(Error, 10,                ””)                                       \
00362     ENUM_ITEM(PowerOffing, 11,          ”       ”)
00363
00364 #define ENUM_SafetyModeType_DECLARES                          \
00365     ENUM_ITEM(Undefined, 0,          ”     ”)          \
00366     ENUM_ITEM(Normal, 1,             ”     ”)           \
00367     ENUM_ITEM(ReducedMode, 2,        ”     ”)          \
00368     ENUM_ITEM(Recovery, 3,           ”       ,     recovery  ”) \
00369     ENUM_ITEM(Violation, 4,          ”       ,      ”) \
00370     ENUM_ITEM(ProtectiveStop, 5,     ”       ,     ,     ”) \
00371     ENUM_ITEM(SafeguardStop, 6,      ”IO       ,     ,     ”) \
00372     ENUM_ITEM(SystemEmergencyStop,7, ”       (     ),       ”) \
00373     ENUM_ITEM(RobotEmergencyStop, 8, ”             ,      ”) \
00374     ENUM_ITEM(Fault, 9,              ”       ”)
00375     //ValidateJointId
00376
00377 /**
00378  * \chinese
00379  *  ISO 10218-1:2011(E) 5.7
00380  * Automatic: In automatic mode, the robot shall execute the task programme and
00381  * the safeguarding measures shall be functioning. Automatic operation shall be
00382  * prevented if any stop condition is detected. Switching from this mode shall
00383  * result in a stop.
00384  * \endchinese
00385  * \english
00386  * Based on ISO 10218-1:2011(E) Section 5.7
00387  * Automatic: In automatic mode, the robot shall execute the task programme and
00388  * the safeguarding measures shall be functioning. Automatic operation shall be
00389  * prevented if any stop condition is detected. Switching from this mode shall
00390  * result in a stop.
00391  * \endenglish
00392  */
00393 #define ENUM_OperationalModeType_DECLARES                                                \
00394     ENUM_ITEM(Disabled, 0, ”   :    Operational Mode”)                                \
00395     ENUM_ITEM(Automatic, 1, ”   :       ,       (auto mode: robot normal operation, speed will not be limited)”)          \
00396     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)”) \
00397
00398 /**
00399  * \~chinese       ,       \~english Robot control mode, the final control object
00400  */
00401 #define ENUM_RobotControlModeType_DECLARES                \
00402     ENUM_ITEM(Unknown, 0,   ”      (unknown control mode)”)             \
00403     ENUM_ITEM(Position, 1,  ”     movej (position control)”)             \
00404     ENUM_ITEM(Speed, 2,     ”     speedj/speedl (speed control)”)      \
00405     ENUM_ITEM(Servo, 3,     ”     servoj (position control)”)            \
00406     ENUM_ITEM(Freedrive, 4, ”     freedrive_mode”)     \
00407     ENUM_ITEM(Force, 5,     ”     force_mode”)         \
00408     ENUM_ITEM(Torque, 6,    ”      (joint torque control)”)                \
00409     ENUM_ITEM(Collision, 7,   ”    (collision mode)”) \
00410
00411 #define ENUM_JointServoModeType_DECLARES                  \
00412     ENUM_ITEM(Unknown, -1, ”  ”)                          \
00413     ENUM_ITEM(Open, 0, ”    (open loop mode)”)                      \
00414     ENUM_ITEM(Current, 1, ”      (current servo mode)”)                \
00415     ENUM_ITEM(Velocity, 2, ”     (speed servo mode)”)                  \
00416     ENUM_ITEM(Position, 3, ”     (position servo mode)”)                 \
00417     ENUM_ITEM(Torque, 4, ”     (torque servo mode)”) \
00418
00419 #define ENUM_JointStateType_DECLARES                          \
00420     ENUM_ITEM(Poweroff, 0, ”          (node not conected to interface board or already powered off)”)          \
00421     ENUM_ITEM(Idle, 2,        ”    (node idle)”)                          \
00422     ENUM_ITEM(Fault, 3,       ”   ,       ,     (node error, node stopped servo move, brake engaged)”) \
00423     ENUM_ITEM(Running, 4,     ”    (node servo)”)                         \
00424     ENUM_ITEM(Bootload, 5,    ” bootloader  ,      (node bootloader state, pause all communication)”) \
00425
00426 #define ENUM_StandardInputAction_DECLARES                     \
00427     ENUM_ITEM(Default, 0,  ”  ”)                          \
```

```
00428     ENUM_ITEM(Handguide, 1, ”       ”)              \
00429     ENUM_ITEM(GoHome, 2, ”          ”)          \
00430     ENUM_ITEM(StartProgram, 3, ”       ”)           \
00431     ENUM_ITEM(StopProgram, 4, ”        ”)           \
00432     ENUM_ITEM(PauseProgram, 5, ”       ”)           \
00433     ENUM_ITEM(PopupDismiss, 6, ”       ”)           \
00434     ENUM_ITEM(PowerOn, 7, ”    /      ”)         \
00435     ENUM_ITEM(PowerOff, 8, ”     /      ”)        \
00436     ENUM_ITEM(ResumeProgram, 9, ”       ”)          \
00437     ENUM_ITEM(SlowDown1, 10, ”     1     ”)         \
00438     ENUM_ITEM(SlowDown2, 11, ”     2     ”)         \
00439     ENUM_ITEM(SafeStop, 12, ”       ”)              \
00440     ENUM_ITEM(RunningGuard, 13, ”      ”)             \
00441     ENUM_ITEM(MoveToFirstPoint, 14, ”          ”) \
00442     ENUM_ITEM(xSlowDown1, 15, ”     1     ”)        \
00443     ENUM_ITEM(xSlowDown2, 16, ”     2     ”)        \
00444     ENUM_ITEM(ConveyorTrack, 17, ”          ”)  \
00445     ENUM_ITEM(xConveyorTrack, 18, ”          ”) \
00446     ENUM_ITEM(UnlockProtectiveStop, 19, ”        ”) \
00447     ENUM_ITEM(ArbitraryResumeProgram , 20, ”                ”)
00448
00449 #define ENUM_StandardOutputRunState_DECLARES            \
00450     ENUM_ITEM(None, 0,      ”       ”)            \
00451     ENUM_ITEM(StopLow, 1, ”       ”)            \
00452     ENUM_ITEM(StopHigh, 2, ”       ”)          \
00453     ENUM_ITEM(RunningHigh, 3, ”      ”)         \
00454     ENUM_ITEM(PausedHigh, 4, ”      ”)          \
00455     ENUM_ITEM(AtHome, 5, ”          ”)        \
00456     ENUM_ITEM(Handguiding, 6, ”        ”)     \
00457     ENUM_ITEM(PowerOn, 7, ”         ”)          \
00458     ENUM_ITEM(RobotEmergencyStop, 8, ”        ”) \
00459     ENUM_ITEM(SystemEmergencyStop, 9, ”          ”) \
00460     ENUM_ITEM(InternalEmergencyStop, 8, ”        ”) \
00461     ENUM_ITEM(ExternalEmergencyStop, 9, ”          ”) \
00462     ENUM_ITEM(SystemError, 10, ”                ”) \
00463     ENUM_ITEM(NotSystemError, 11, ”              ”) \
00464     ENUM_ITEM(RobotOperable, 12, ”            ”) \
00465     ENUM_ITEM(OperationalMode, 13, ”            ”) \
00466     ENUM_ITEM(SafeguardStop, 14, ”         ”) \
00467     ENUM_ITEM(ProtectiveStop, 15, ”         ”) \
00468     ENUM_ITEM(ProgramAborted, 16, ”        ”) \
00469     ENUM_ITEM(WaypointArrived, 17, ”        ”)
00470
00471 #define ENUM_SafetyInputAction_DECLARES                  \
00472     ENUM_ITEM(Unassigned, 0, ”       ”)              \
00473     ENUM_ITEM(EmergencyStop, 1, ”      ”)            \
00474     ENUM_ITEM(SafeguardStop, 2, ”       ,    ”)   \
00475     ENUM_ITEM(SafeguardReset, 3, ”      ,    ”)   \
00476     ENUM_ITEM(ThreePositionSwitch, 4, ”3    ”)        \
00477     ENUM_ITEM(OperationalMode, 5, ”        ”)       \
00478     ENUM_ITEM(HandGuide, 6, ”   ”)              \
00479     ENUM_ITEM(ReducedMode, 7, ”    1(   )              0   ”)        \
00480     ENUM_ITEM(AutomaticModeSafeguardStop, 8, ”       (       )”) \
00481     ENUM_ITEM(AutomaticModeSafeguardReset, 9, ”         (       )”)
00482
00483 #define ENUM_SafetyOutputRunState_DECLARES                    \
00484     ENUM_ITEM(Unassigned, 0, ”     ”)                \
00485     ENUM_ITEM(SystemEmergencyStop, 1, ”            ”)       \
00486     ENUM_ITEM(NotSystemEmergencyStop, 2, ”              ”)     \
00487     ENUM_ITEM(RobotMoving, 3, ”          0.1rad/s”)                \
00488     ENUM_ITEM(RobotNotMoving, 4, ”           0.1rad/s”)           \
00489     ENUM_ITEM(ReducedMode, 5, ”         ”)                    \
00490     ENUM_ITEM(NotReducedMode, 6, ”          ”)                \
00491     ENUM_ITEM(SafeHome, 7, ”         Home ”)                \
00492     ENUM_ITEM(RobotNotStopping, 8, ”            ”)
00493
00494 #define ENUM_PayloadIdentifyMoveAxis_DECLARES      \
00495     ENUM_ITEM(Joint_2_6, 0,” 2 6   ”)         \
00496     ENUM_ITEM(Joint_3_6, 1,” 3 6   ”)         \
00497     ENUM_ITEM(Joint_4_6, 2,” 4 6   ”)         \
00498     ENUM_ITEM(Joint_4_5_6, 3,” 4 5 6   ”)    \
00499
00500 #define ENUM_EnvelopingShape_DECLARES \
00501     ENUM_ITEM(Cube, 1,”   ”) \
00502     ENUM_ITEM(Column, 2,”   ”) \
00503     ENUM_ITEM(Stl, 3,” STL          ”)
00504
00505 #define ENUM_TaskFrameType_DECLARES                      \
00506     ENUM_ITEM(NONE, 0,””)        \
00507     ENUM_ITEM(POINT_FORCE, 1, ”      ,       y    TCP       , x z        ” \
00508                             ”      ,       y    TCP       , x z        ” \
00509                             ”  TCP          10mm” \
00510                             ”  X ,     X     Y      ,     X     Y   , ” \
00511                             ”         Z , Y-X  Y-Z    ,          ”) \
00512     ENUM_ITEM(FRAME_FORCE, 2,”       SIMPLE_FORC”) \
00513     ENUM_ITEM(MOTION_FORCE, 3,”      ,       x   TCP    x-y    y      ,     x-y   ”)\
00514     ENUM_ITEM(TOOL_FORCE, 4,”          ”)
```

```
00515
00516 #ifdef ERROR
00517 #undef ERROR
00518 #endif
00519
00520 #define ENUM_TraceLevel_DECLARES  \
00521     ENUM_ITEM(FATAL, 0, "") \
00522     ENUM_ITEM(ERROR, 1, "") \
00523     ENUM_ITEM(WARNING, 2, "") \
00524     ENUM_ITEM(INFO, 3, "") \
00525     ENUM_ITEM(DEBUG, 4, "")
00526
00527 #define ENUM_AxisModeType_DECLARES  \
00528     ENUM_ITEM(NoController, -1, "       ,   aubo_control       NoController") \
00529     ENUM_ITEM(Disconnected, 0, "   ") \
00530     ENUM_ITEM(PowerOff, 1, "  ") \
00531     ENUM_ITEM(BrakeReleasing, 2, "    ") \
00532     ENUM_ITEM(Idle, 3, "  ") \
00533     ENUM_ITEM(Running, 4, "   ") \
00534     ENUM_ITEM(Fault, 5, "   ")
00535
00536 #define ENUM_HandleStateType_DECLARES  \
00537     ENUM_ITEM(Disconnected, 0, "   ") \
00538     ENUM_ITEM(Init, 1, "   ") \
00539     ENUM_ITEM(Connected, 2, "   ") \
00540     ENUM_ITEM(Lock, 3, "  ") \
00541     ENUM_ITEM(Fault, 4, "   ") \
00542
00543 #define ENUM_HandleModeType_DECLARES  \
00544     ENUM_ITEM(None, 0, "    ") \
00545     ENUM_ITEM(Default, 1, "S     ") \
00546     ENUM_ITEM(MultiFuncG1, 2, "G       ") \
00547
00548 #define ENUM_SafeguedStopType_DECLARES  \
00549     ENUM_ITEM(None, 0, "    ") \
00550     ENUM_ITEM(SafeguedStopIOInput, 1, "   (IO  )") \
00551     ENUM_ITEM(SafeguedStop3PE, 2, "   (   )") \
00552     ENUM_ITEM(SafeguedStopOperational, 3, "   (   )")
00553
00554 #define ENUM_RobotEmergencyStopType_DECLARES  \
00555     ENUM_ITEM(RobotEmergencyStopNone, 0, "    ") \
00556     ENUM_ITEM(RobotEmergencyStopControlBox, 1, "   (    )") \
00557     ENUM_ITEM(RobotEmergencyStopTeachPendant, 2, "   (    )") \
00558     ENUM_ITEM(RobotEmergencyStopHandle, 3, "   (    )") \
00559     ENUM_ITEM(RobotEmergencyStopEI, 4, "   ( IO )")
00560
00561 #define ENUM_ITEM(c, n, ...) c = n,
00562 enum AuboErrorCodes : int
00563 {
00564     ENUM_AuboErrorCodes_DECLARES
00565 };
00566
00567 enum class RuntimeState : int
00568 {
00569     ENUM_RuntimeState_DECLARES
00570 };
00571
00572 enum class RobotModeType : int
00573 {
00574     ENUM_RobotModeType_DECLARES
00575 };
00576
00577 enum class AxisModeType : int
00578 {
00579     ENUM_AxisModeType_DECLARES
00580 };
00581
00582 enum class HandleStateType : int
00583 {
00584     ENUM_HandleStateType_DECLARES
00585 };
00586
00587 enum class HandleModeType : int
00588 {
00589     ENUM_HandleModeType_DECLARES
00590 };
00591
00592 /**
00593  * \~chinese     : \~english Safety Mode
00594  *
00595  */
00596 enum class SafetyModeType : int
00597 {
00598     ENUM_SafetyModeType_DECLARES
00599 };
00600
00601 /**
```

```
00602  * \~chinese      \~english Operational Mode
00603  */
00604 enum class OperationalModeType : int
00605 {
00606     ENUM_OperationalModeType_DECLARES
00607 };
00608
00609 /**
00610  * \~chinese       \~english Robot Control Mode
00611  */
00612 enum class RobotControlModeType : int
00613 {
00614     ENUM_RobotControlModeType_DECLARES
00615 };
00616
00617 /**
00618  * \~chinese       \~english Joint Servo Mode
00619  */
00620 enum class JointServoModeType : int
00621 {
00622     ENUM_JointServoModeType_DECLARES
00623 };
00624
00625 /**
00626  * \~chinese      \~english Joint State
00627  */
00628 enum class JointStateType : int
00629 {
00630     ENUM_JointStateType_DECLARES
00631 };
00632
00633 /**
00634  * \~chinese       \~english Standard Output Run State
00635  */
00636 enum class StandardOutputRunState : int
00637 {
00638     ENUM_StandardOutputRunState_DECLARES
00639 };
00640
00641 /**
00642  * @brief The StandardInputAction enum
00643  */
00644 enum class StandardInputAction : int
00645 {
00646     ENUM_StandardInputAction_DECLARES
00647 };
00648
00649 enum class SafetyInputAction : int
00650 {
00651     ENUM_SafetyInputAction_DECLARES
00652 };
00653
00654 enum class SafetyOutputRunState : int
00655 {
00656     ENUM_SafetyOutputRunState_DECLARES
00657 };
00658
00659 enum TaskFrameType
00660 {
00661     ENUM_TaskFrameType_DECLARES
00662 };
00663
00664 enum EnvelopingShape : int
00665 {
00666     ENUM_EnvelopingShape_DECLARES
00667 };
00668
00669 enum PayloadIdentifyMoveAxis : int
00670 {
00671     ENUM_PayloadIdentifyMoveAxis_DECLARES
00672 };
00673
00674 enum TraceLevel
00675 {
00676     ENUM_TraceLevel_DECLARES
00677 };
00678
00679 enum SafeguedStopType : int
00680 {
00681     ENUM_SafeguedStopType_DECLARES
00682 };
00683
00684 enum RobotEmergencyStopType : int
00685 {
00686     ENUM_RobotEmergencyStopType_DECLARES
00687 };
00688 #undef ENUM_ITEM
```

```
00689
00690 #define DECL_TO_STRING_FUNC(ENUM)                          \
00691     inline std::string toString(ENUM v)                    \
00692     {                                                      \
00693         using T = ENUM;                                    \
00694         std::string name = #ENUM ".";                      \
00695         ENUM_##ENUM##_DECLARES                             \
00696                                                            \
00697             return #ENUM ".Unkown";                        \
00698     }                                                      \
00699     inline std::ostream &operator<<(std::ostream &os, ENUM v) \
00700     {                                                      \
00701         os << toString(v);                                 \
00702         return os;                                         \
00703     }
00704
00705 #define ENUM_ITEM(c, n, ...) \
00706     if (v == T::c) {         \
00707         return name + #c;    \
00708     }
00709
00710 DECL_TO_STRING_FUNC(RuntimeState)
00711 DECL_TO_STRING_FUNC(RobotModeType)
00712 DECL_TO_STRING_FUNC(AxisModeType)
00713 DECL_TO_STRING_FUNC(SafetyModeType)
00714 DECL_TO_STRING_FUNC(OperationalModeType)
00715 DECL_TO_STRING_FUNC(RobotControlModeType)
00716 DECL_TO_STRING_FUNC(JointServoModeType)
00717 DECL_TO_STRING_FUNC(JointStateType)
00718 DECL_TO_STRING_FUNC(StandardInputAction)
00719 DECL_TO_STRING_FUNC(StandardOutputRunState)
00720 DECL_TO_STRING_FUNC(SafetyInputAction)
00721 DECL_TO_STRING_FUNC(SafetyOutputRunState)
00722 DECL_TO_STRING_FUNC(TaskFrameType)
00723 DECL_TO_STRING_FUNC(TraceLevel)
00724 DECL_TO_STRING_FUNC(HandleModeType)
00725 DECL_TO_STRING_FUNC(HandleStateType)
00726
00727 #undef ENUM_ITEM
00728
00729 enum class ForceControlState
00730 {
00731     Stopped,
00732     Starting,
00733     Stropping,
00734     Running
00735 };
00736
00737 enum class RefFrameType
00738 {
00739     None, ///
00740     Tool, ///< \~chinese      \~english Tool coordinate system
00741     Path, ///< \~chinese      \~english Trajectory coordinate system
00742     Base  ///< \~chinese      \~english Base coordinate system
00743 };
00744
00745 /// \~chinese        \~english Circular motion parameters definition
00746 struct CircleParameters
00747 {
00748     std::vector<double> pose_via; ///< \~chinese          \~english Pose of the intermediate point in circular motion
00749     std::vector<double> pose_to; ///< \~chinese          \~english Pose of the end point in circular motion
00750     double a; ///< \~chinese    ,   : m/s^2 \~english Acceleration, unit: m/s^2
00751     double v; ///< \~chinese     : m/s \~english Speed, unit: m/s
00752     double blend_radius; ///< \~chinese     , : m \~english Blending radius, unit: m
00753     double duration; ///< \~chinese      : s \~english Running time, unit: s
00754     double helix;
00755     double spiral;
00756     double direction;
00757     int loop_times; ///< \~chinese      \~english Currently not supported
00758 };
00759
00760 inline std::ostream &operator<<(std::ostream &os, CircleParameters p)
00761 {
00762     return os;
00763 }
00764
00765 struct SpiralParameters
00766 {
00767     std::vector<double> frame; ///< \~chinese              \~english Reference point, the center point of the spiral and the
    reference coordinate system
00768     int plane; ///< \~chinese       0-XY 1-YZ 2-ZX \~english Reference plane selection 0-XY 1-YZ 2-ZX
00769     double angle; ///< \~chinese              \~english The angle of rotation, if positive, the robot rotates counterclockwise
00770     double spiral; ///< \~chinese      \~english Positive outward
00771     double helix;  ///< \~chinese      \~english Positive upward
00772 };
00773
00774 inline std::ostream &operator<<(std::ostream &os, SpiralParameters p)
```

```
00775 {
00776     return os;
00777 }
00778
00779 struct Enveloping
00780 {
00781     EnvelopingShape shape; // \~chinese      \~english Enveloping shape
00782     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.
00783                 // \~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;
00784                 // \~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;
00785     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
00786 };
00787
00788 inline std::ostream &operator«(std::ostream &os, Enveloping p)
00789 {
00790     return os;
00791 }
00792
00793 /// \~chinese          \~english Trajectory configuration for
00794 /// payload identification
00795 struct TrajConfig
00796 {
00797     std::vector<Enveloping> envelopings; // \~chinese      \~english Enveloping combination
00798     PayloadIdentifyMoveAxis move_axis; // \~chinese     (ID),   0   \~english Axis of movement (ID), index starts from 0
00799     std::vector<double> init_joint; // \~chinese       \~english Initial joint positions
00800     std::vector<double> upper_joint_bound; // \~chinese      \~english Upper joint limits
00801     std::vector<double> lower_joint_bound; // \~chinese      \~english Lower joint limits
00802     std::vector<double> max_velocity; // \~chinese             3.0 \~english Maximum joint velocities, default value is 3.0
00803     std::vector<double> max_acceleration; // \~chinese               5.0 \~english Maximum joint accelerations, default value is 5.0
00804 };
00805
00806 inline std::ostream &operator«(std::ostream &os, TrajConfig p)
00807 {
00808     return os;
00809 }
00810
00811 // result with error code
00812 using ResultWithErrno = std::tuple<std::vector<double>, int>;
00813 using ResultWithErrno1 = std::tuple<std::vector<std::vector<double», int>;
00814 using ResultWithErrno2 = std::tuple<std::vector<std::string>, int>;
00815 using ResultWithErrno3 = std::tuple<int, int>;
00816
00817 // mass, cog, aom, inertia
00818 using Payload = std::tuple<double, std::vector<double>, std::vector<double>,
00819                            std::vector<double»;
00820
00821 // force_offset, com, mass, angle
00822 using ForceSensorCalibResult =
00823     std::tuple<std::vector<double>, std::vector<double>, double,
00824                std::vector<double»;
00825
00826 // force_offset, com, mass, angle error
00827 using ForceSensorCalibResultWithError =
00828     std::tuple<std::vector<double>, std::vector<double>, double,
00829                std::vector<double>, double>;
00830
00831 // \~chinese     m,d,k \~english Dynamics model m, d, k
00832 using DynamicsModel =
00833     std::tuple<std::vector<double>, std::vector<double>, std::vector<double»;
00834
00835 // double xmin;
00836 // double xmax;
00837 // double ymin;
00838 // double ymax;
00839 // double zmin;
00840 // double zmax;
00841 using Box = std::vector<double>;
00842
00843 // double xcbottom;
00844 // double ycbottom;
00845 // double zcbottom;
00846 // double height;
00847 // double radius;
00848 using Cylinder = std::vector<double>;
00849
00850 // double xc;
00851 // double yc;
00852 // double radius;
00853 using Sphere = std::vector<double>;
00854
00855 struct RobotMsg
00856 {
```

```
00857     uint64_t timestamp; ///< \~chinese         \~english Timestamp,
00858                         ///< i.e., system time
00859     TraceLevel level; ///< \~chinese      \~english Log level
00860     int code;         ///< \~chinese     \~english Error code
00861     std::string
00862         source; ///< \~chinese          alias \~english Alias of the
00863                 ///< robot sending the message
00864                 ///< \~chinese    /root/arcs_ws/config/aubo_control.conf
00865                 ///< \~english Can be found in
00866                 ///< /root/arcs_ws/config/aubo_control.conf
00867                 ///< \~chinese          alias \~english The robot's
00868                 ///< alias can be found in the configuration file
00869                 ///< /root/arcs_ws/config/aubo_control.conf
00870     std::vector<std::string> args; ///< \~chinese      \~english Robot parameters
00871 };
00872 using RobotMsgVector = std::vector<RobotMsg>;
00873
00874 struct GripperStatus
00875 {
00876     std::string name;
00877     bool is_connected;
00878     bool is_enabled;
00879     double position;
00880     double velocity;
00881     double force;
00882     double angle;
00883     double r_velocity;
00884     double torque;
00885     bool object_detection;
00886     bool motion_state;
00887     double voltage;
00888     double temperature;
00889     int status_code;
00890 };
00891 using GripperStatusVector = std::vector<GripperStatus>;
00892
00893 /// \~chinese RTDE  \~english RTDE menu
00894 struct RtdeRecipe
00895 {
00896     bool to_server;   ///< \~chinese  /  \~english Input/Output
00897     int chanel;       ///< \~chinese   \~english Channel
00898     double frequency; ///< \~chinese     \~english Update frequency
00899     int trigger; ///< \~chinese    (     ): 0 -  ; 1 -
00900                  ///< \~english Trigger method (this feature is not yet
00901                  ///< implemented): 0 - Periodic; 1 - Change
00902     std::vector<std::string> segments; ///< \~chinese      \~english segment list
00903 };
00904
00905 /// \~chinese     \~english Error type
00906 enum error_type
00907 {
00908     parse_error = -32700,      ///< \~chinese      \~english Parse error
00909     invalid_request = -32600, ///< \~chinese      \~english Invalid request
00910     method_not_found = -32601, ///< \~chinese      \~english Method not found
00911     invalid_params = -32602, ///< \~chinese      \~english Invalid parameters
00912     internal_error = -32603, ///< \~chinese      \~english Internal error
00913     server_error = -32500, ///< \~chinese      \~english Server error
00914     invalid = -32400       ///< \~chinese    \~english Invalid
00915 };
00916
00917 /// \~chinese    \~english Exception code
00918 enum ExceptionCode
00919 {
00920     EC_DISCONNECTED = -1, ///< \~chinese      \~english Disconnected
00921     EC_NOT_LOGINED = -2,  ///< \~chinese     \~english Not logged in
00922     EC_INVAL_SOCKET = -3, ///< \~chinese      \~english Invalid socket
00923     EC_REQUEST_BUSY = -4, ///< \~chinese      \~english Request busy
00924     EC_SEND_FAILED = -5,  ///< \~chinese     \~english Send failed
00925     EC_RECV_TIMEOUT = -6, ///< \~chinese      \~english Receive timeout
00926     EC_RECV_ERROR = -7,   ///< \~chinese      \~english Receive error
00927     EC_PARSE_ERROR = -8,  ///< \~chinese      \~english Parse error
00928     EC_INVALID_REQUEST = -9, ///< \~chinese      \~english Invalid request
00929     EC_METHOD_NOT_FOUND = -10, ///< \~chinese      \~english Method not found
00930     EC_INVALID_PARAMS = -11, ///< \~chinese      \~english Invalid parameters
00931     EC_INTERNAL_ERROR = -12, ///< \~chinese      \~english Internal error
00932     EC_SERVER_ERROR = -13, ///< \~chinese      \~english Server error
00933     EC_INVALID = -14       ///< \~chinese    \~english Invalid
00934 };
00935
00936 /// \~chinese       AuboException \~english Custom exception class
00937 /// AuboException
00938 class AuboException : public std::exception
00939 {
00940 public:
00941     AuboException(int code, const std::string &prefix,
00942                   const std::string &message) noexcept
00943         : code_(code), message_(prefix + "-" + message)
```

```
00944     {
00945     }
00946
00947     AuboException(int code, const std::string &message) noexcept
00948         : code_(code), message_(message)
00949     {
00950     }
00951
00952     error_type type() const
00953     {
00954         if (code_ >= -32603 && code_ <= -32600) {
00955             return static_cast<error_type>(code_);
00956         } else if (code_ >= -32099 && code_ <= -32000) {
00957             return server_error;
00958         } else if (code_ == -32700) {
00959             return parse_error;
00960         }
00961         return invalid;
00962     }
00963
00964     int code() const { return code_; }
00965     const char *what() const noexcept override { return message_.c_str(); }
00966
00967 private:
00968     int code_;            ///< \~chinese     \~english Exception code
00969     std::string message_; ///< \~chinese      \~chinese Exception message
00970 };
00971
00972 inline const char *returnValue2Str(int retval)
00973 {
00974     static const char *retval_str[] = {
00975 #define ENUM_ITEM(n, v, s) s,
00976         ENUM_AuboErrorCodes_DECLARES
00977 #undef ENUM_ITEM
00978     };
00979
00980     enum arcs_index
00981     {
00982 #define ENUM_ITEM(n, v, s) n##_INDEX,
00983         ENUM_AuboErrorCodes_DECLARES
00984 #undef ENUM_ITEM
00985     };
00986
00987     int index = -1;
00988
00989 #define ENUM_ITEM(n, v, s) \
00990     if (abs(retval) == v)  \
00991         index = n##_INDEX;
00992     ENUM_AuboErrorCodes_DECLARES
00993 #undef ENUM_ITEM
00994
00995         if (index == -1)
00996     {
00997         index = AUBO_ERR_UNKOWN_INDEX;
00998     }
00999
01000     return retval_str[(unsigned)index];
01001 }
01002
01003 } // namespace common_interface
01004 } // namespace arcs
01005 #endif
01006
01007 // clang-format on
01008
01009 #if defined ENABLE_JSON_TYPES
01010 #include "bindings/jsonrpc/json_types.h"
01011 #endif
```

Index

_D
error_stack.h, 716, 717
_PH1_
error_stack.h, 718
_PH2_
error_stack.h, 718
_PH3_
error_stack.h, 718
_PH4_
error_stack.h, 718
~AuboApi
arcs::common_interface::AuboApi, 612
~AxisInterface
arcs::common_interface::AxisInterface, 618
~ForceControl
arcs::common_interface::ForceControl, 624
~GripperInterface
arcs::common_interface::GripperInterface, 626
~IoControl
arcs::common_interface::IoControl, 636
~Math
arcs::common_interface::Math, 638
~MotionControl
arcs::common_interface::MotionControl, 644
~RegisterControl
arcs::common_interface::RegisterControl, 650
~RobotAlgorithm
arcs::common_interface::RobotAlgorithm, 653
~RobotConfig
arcs::common_interface::RobotConfig, 658
~RobotInterface
arcs::common_interface::RobotInterface, 660
~RobotManage
arcs::common_interface::RobotManage, 662
~RobotState
arcs::common_interface::RobotState, 679
~RuntimeMachine
arcs::common_interface::RuntimeMachine, 684
~Serial
arcs::common_interface::Serial, 686
~Socket
arcs::common_interface::Socket, 687
~SyncMove
arcs::common_interface::SyncMove, 691
~SystemInfo
arcs::common_interface::SystemInfo, 692
~Trace
arcs::common_interface::Trace, 694
a
arcs::common_interface::CircleParameters, 619
abort
RuntimeMachine ( ), 523
addCollisionBox
RobotAlgorithm ( ), 352
addCrossConnection
IoControl (IO), 163
addFixedModbusGroup
RegisterControl ( ), 66
addGroupInput
IoControl (IO), 165
addGroupOutput
IoControl (IO), 166
addInt32RegEncoder
RegisterControl ( ), 67
addModbusEncoder
RegisterControl ( ), 67
alarm
Trace ( ), 583
allow_manual_high_speed
arcs::common_interface::RobotSafetyParameterRange, 667
angle
arcs::common_interface::GripperStatus, 630
arcs::common_interface::SpiralParameters, 688
arbitraryResume
RuntimeMachine ( ), 523
arcs, 591
arcs::common_interface, 591
AuboApiPtr, 594
AuboErrorCodes, 599
AxisInterfacePtr, 594
AxisModeType, 599
Base, 604
Box, 594
Cylinder, 594
DynamicsModel, 594
EC_DISCONNECTED, 600
EC_INTERNAL_ERROR, 601
EC_INVAL_SOCKET, 600
EC_INVALID, 601
EC_INVALID_PARAMS, 601
EC_INVALID_REQUEST, 601
EC_METHOD_NOT_FOUND, 601
EC_NOT_LOGINED, 600
EC_PARSE_ERROR, 601

EC_RECV_ERROR, 600
EC_RECV_TIMEOUT, 600
EC_REQUEST_BUSY, 600
EC_SEND_FAILED, 600
EC_SERVER_ERROR, 601
ENUM_AuboErrorCodes_DECLARES, 599
ENUM_AxisModeType_DECLARES, 599
ENUM_EnvelopingShape_DECLARES, 600
ENUM_HandleModeType_DECLARES, 601
ENUM_HandleStateType_DECLARES, 602
ENUM_JointServoModeType_DECLARES, 602
ENUM_JointStateType_DECLARES, 602
ENUM_OperationalModeType_DECLARES, 603
ENUM_PayloadIdentifyMoveAxis_DECLARES, 603
ENUM_RobotControlModeType_DECLARES, 604
ENUM_RobotEmergencyStopType_DECLARES, 604
ENUM_RobotModeType_DECLARES, 605
ENUM_RuntimeState_DECLARES, 605
ENUM_SafeguedStopType_DECLARES, 605
ENUM_SafetyInputAction_DECLARES, 605
ENUM_SafetyModeType_DECLARES, 606
ENUM_SafetyOutputRunState_DECLARES, 606
ENUM_StandardInputAction_DECLARES, 606
ENUM_StandardOutputRunState_DECLARES, 606
ENUM_TaskFrameType_DECLARES, 607
ENUM_TraceLevel_DECLARES, 607
EnvelopingShape, 599
error_type, 600
ExceptionCode, 600
ForceControlPtr, 594
ForceControlState, 601
ForceSensorCalibResult, 595
ForceSensorCalibResultWithError, 595
GripperInterfacePtr, 595
GripperStatusVector, 595
HandleModeType, 601
HandleStateType, 601
internal_error, 600
invalid, 600
invalid_params, 600
invalid_request, 600
IoControlPtr, 595
JointServoModeType, 602
JointStateType, 602
MathPtr, 595
method_not_found, 600
MotionControlPtr, 595
None, 604
OperationalModeType, 602
operator<<, 607, 608
parse_error, 600
Path, 604
PathBuffer_CubicSpline, 603
PathBuffer_JointBSpline, 603
PathBuffer_JointBSplineC, 603
PathBuffer_JointSpline, 603
PathBuffer_JointSplineC, 603
PathBuffer_TOPPRA, 603
PathBufferType, 603
Payload, 596
PayloadIdentifyMoveAxis, 603
RefFrameType, 603
RegisterControlPtr, 596
ResultWithErrno, 596
ResultWithErrno1, 596
ResultWithErrno2, 596
ResultWithErrno3, 596
returnValue2Str, 608
RobotAlgorithmPtr, 596
RobotConfigPtr, 597
RobotControlModeType, 604
RobotEmergencyStopType, 604
RobotInterfacePtr, 597
RobotManagePtr, 597
RobotModeType, 604
RobotMsgVector, 597
RobotStatePtr, 597
Running, 601
RuntimeMachinePtr, 597
RuntimeState, 605
SafeguedStopType, 605
SafetyInputAction, 605
SafetyModeType, 605
SafetyOutputRunState, 606
SerialPtr, 597
server_error, 600
SocketPtr, 597
Sphere, 598
StandardInputAction, 606
StandardOutputRunState, 606
Starting, 601
Stopped, 601
Stropping, 601
SyncMovePtr, 598
SystemInfoPtr, 598
TaskFrameType, 606
TaskSet, 598
Tool, 604
TraceLevel, 607
TracePtr, 598
Vector3d, 598
Vector3f, 598
Vector4d, 598
Vector4f, 599
Vector6f, 599
arcs::common_interface::AuboApi, 611
~AuboApi, 612
AuboApi, 612

d_, 612
arcs::common_interface::AuboException, 613
AuboException, 614
code, 615
code_, 615
message_, 615
type, 615
what, 615
arcs::common_interface::AxisInterface, 616
~AxisInterface, 618
AxisInterface, 618
d_, 618
arcs::common_interface::CircleParameters, 618
a, 619
blend_radius, 619
direction, 619
duration, 619
helix, 620
loop_times, 620
pose_to, 620
pose_via, 620
spiral, 620
v, 620
arcs::common_interface::Enveloping, 621
ep_args, 622
shape, 622
stl_path, 622
arcs::common_interface::ForceControl, 622
~ForceControl, 624
d_, 625
ForceControl, 624
arcs::common_interface::GripperInterface, 625
~GripperInterface, 626
d_, 628
gripperGetAngle, 626
gripperGetForce, 626
gripperGetMotionState, 626
gripperGetMountPose, 626
gripperGetObjectDetection, 627
gripperGetPosition, 627
gripperGetRVelocity, 627
gripperGetSoftwareVersion, 627
gripperGetTemperature, 627
gripperGetTorque, 627
gripperGetVelocity, 627
gripperGetVoltage, 627
GripperInterface, 626
gripperSetAngle, 628
gripperSetForce, 628
gripperSetRVelocity, 628
gripperSetTorque, 628
gripperSetVelocity, 628
arcs::common_interface::GripperStatus, 629
angle, 630
force, 630
is_connected, 630
is_enabled, 630
motion_state, 630
name, 630
object_detection, 630
position, 630
r_velocity, 631
status_code, 631
temperature, 631
torque, 631
velocity, 631
voltage, 631
arcs::common_interface::IoControl, 632
~IoControl, 636
d_, 636
IoControl, 636
arcs::common_interface::Math, 636
~Math, 638
d_, 638
Math, 638
arcs::common_interface::MotionControl, 638
~MotionControl, 644
d_, 646
getConveyorTrackNextItem, 644
isConveyorTrackExceed, 645
MotionControl, 644
servoCartesianWithAxisGroup, 646
servoJointWithAxisGroup, 646
arcs::common_interface::RegisterControl, 647
~RegisterControl, 650
d_, 651
modbusSetOutputSignalWithTimeout, 650
RegisterControl, 650
arcs::common_interface::RobotAlgorithm, 651
~RobotAlgorithm, 653
d_, 655
pathMovej, 654
RobotAlgorithm, 653
arcs::common_interface::RobotConfig, 655
~RobotConfig, 658
d_, 659
disableAxisGroup, 659
enableAxisGroup, 659
getRobotBaseParent, 659
RobotConfig, 658
arcs::common_interface::RobotInterface, 659
~RobotInterface, 660
d_, 660
RobotInterface, 660
arcs::common_interface::RobotManage, 661
~RobotManage, 662
d_, 663
RobotManage, 662
arcs::common_interface::RobotMsg, 663
args, 664
code, 664
level, 664
source, 664
timestamp, 664
arcs::common_interface::RobotSafetyParameterRange, 665

allow_manual_high_speed, 667
crc32, 667
cubic, 668
elbow_force, 668
elbow_speed, 668
joint_torque, 668
momentum, 668
orig, 668
params, 668
plane, 669
planes, 669
power, 669
qdmax, 669
qmax, 669
qmin, 669
reduced_entry_distance, 669
reduced_entry_time, 670
restrict_elbow, 670
RobotSafetyParameterRange, 667
safety_home, 670
safety_input_auto_safeguard_reset, 670
safety_input_auto_safeguard_stop, 670
safety_input_emergency_stop, 670
safety_input_handguide, 671
safety_input_operational_mode, 671
safety_input_reduced_mode, 671
safety_input_safeguard_reset, 671
safety_input_safeguard_stop, 671
safety_input_three_position_switch, 671
safety_output_emergency_stop, 672
safety_output_not_emergency_stop, 672
safety_output_not_reduced_mode, 672
safety_output_reduced_mode, 672
safety_output_robot_moving, 672
safety_output_robot_not_stopping, 672
safety_output_robot_steady, 673
safety_output_safe_home, 673
safety_output_safetyguard_stop, 673
size, 673
stop_distance, 673
stop_time, 673
tcp_force, 674
tcp_speed, 674
tool_azimuth, 674
tool_deviation, 674
tool_inclination, 674
tool_orientation, 674
tools, 675
tp_3pe_for_handguide, 675
trigger_planes, 675
arcs::common_interface::RobotState, 675
~RobotState, 679
d_, 680
getBaseForceSensor, 679
RobotState, 679
arcs::common_interface::RtdeRecipe, 680
chanel, 681
frequency, 681
segments, 681
to_server, 681
trigger, 682
arcs::common_interface::RuntimeMachine, 682
~RuntimeMachine, 684
d_, 685
getExecutionStatus1, 684
getRuntimeState, 684
RuntimeMachine, 684
arcs::common_interface::Serial, 685
~Serial, 686
d_, 686
Serial, 686
arcs::common_interface::Socket, 686
~Socket, 687
d_, 687
Socket, 687
arcs::common_interface::SpiralParameters, 688
angle, 688
frame, 688
helix, 689
plane, 689
spiral, 689
arcs::common_interface::SyncMove, 689
~SyncMove, 691
d_, 692
SyncMove, 691
arcs::common_interface::SystemInfo, 692
~SystemInfo, 692
d_, 693
SystemInfo, 692
arcs::common_interface::Trace, 693
~Trace, 694
d_, 694
notify, 694
Trace, 694
arcs::common_interface::TrajConfig, 695
envelopings, 696
init_joint, 696
lower_joint_bound, 696
max_acceleration, 696
max_velocity, 696
move_axis, 696
upper_joint_bound, 696
arcs::common_interface::VibrationRecalibrationParameter, 697
cog, 698
inertia, 698
mass, 698
points, 698
stiff_param, 698
stiff_section, 698
arcs::common_interface::WObjectData, 699
attach_frame, 699
obj_coord, 699
remote_tool, 700
user_coord, 700
arcs::error_stack, 608

ARCS_ERROR_CODES, 609
codeCompose, 609
dump, 609
errorCode2Str, 609
ErrorCodes, 608
mod, 609
str2ErrorCode, 609
ARCS_ERROR_CODES
arcs::error_stack, 609
error_stack.h, 718
args
arcs::common_interface::RobotMsg, 664
attach_frame
arcs::common_interface::WObjectData, 699
attachRobotBaseTo
RobotConfig ( ), 384
attachWeldingGun
RobotConfig ( ), 384
AUBO_SDK_HAL_ERROR_H
error_stack.h, 718
AUBO_SDK_RTM_ERROR_H
error_stack.h, 719
AUBO_SDK_SYSTEM_ERROR_H
error_stack.h, 719
AuboApi
arcs::common_interface::AuboApi, 612
AuboApi ( ), 25
getAxisInterface, 26
getAxisNames, 26
getGripperInterface, 26
getMath, 26
getRegisterControl, 27
getRobotInterface, 27
getRobotNames, 28
getRuntimeMachine, 28
getSerial, 29
getSocket, 29
getSyncMove, 29
getSystemInfo, 30
getTrace, 30
AuboApiPtr
arcs::common_interface, 594
AuboErrorCodes
arcs::common_interface, 599
AuboException
arcs::common_interface::AuboException, 614
AuboStudio SDK, 1
axisGroupAdd
SyncMove ( ), 565
axisGroupAddAxis
SyncMove ( ), 565
axisGroupDelete
SyncMove ( ), 566
axisGroupGetActualPositions
SyncMove ( ), 566
axisGroupGetAxisIndex
SyncMove ( ), 567
axisGroupGetAxisName
SyncMove ( ), 567
axisGroupGetTargetPositions
SyncMove ( ), 567
axisGroupMoveJoint
SyncMove ( ), 568
axisGroupOffsetPositions
SyncMove ( ), 568
axisGroupSpeedJoint
SyncMove ( ), 569
axisGroupUpdateAxis
SyncMove ( ), 569
AxisInterface
arcs::common_interface::AxisInterface, 618
AxisInterface ( ), 30
clearAxisError, 32
enableExtAxis, 32
followAnotherAxis, 33
getAxisModeType, 33
getErrorCode, 33
getExtAxisAcceleration, 33
getExtAxisBusCurrent, 34
getExtAxisBusVoltage, 34
getExtAxisCurrent, 34
getExtAxisMaxAcceleration, 34
getExtAxisMaxAccelerationLimit, 34
getExtAxisMaxPosition, 35
getExtAxisMaxPositionLimit, 35
getExtAxisMaxVelocity, 35
getExtAxisMaxVelocityLimit, 35
getExtAxisMountingPose, 35
getExtAxisPose, 36
getExtAxisPosition, 36
getExtAxisTemperature, 36
getExtAxisType, 36
getExtAxisVelocity, 36
getExtMinPosition, 37
getExtMinPositionLimit, 37
getReductionRatio, 37
moveExtJoint, 37
poweroffExtAxis, 38
poweronExtAxis, 38
setExtAxisAcceleration, 38
setExtAxisMaxAccelerationLimit, 38
setExtAxisMaxPositionLimit, 39
setExtAxisMaxVelocityLimit, 39
setExtAxisMountingPose, 39
setExtAxisType, 40
setExtAxisVelocity, 40
setExtAxisZero, 40
setExtMinPositionLimit, 40
setReductionRatio, 41
speedExtJoint, 41
stopExtJoint, 41
stopFollowAnotherAxis, 42
AxisInterfacePtr
arcs::common_interface, 594
AxisModeType
arcs::common_interface, 599

backdrive
RobotManage ( ), 441
Base
arcs::common_interface, 604
blend_radius
arcs::common_interface::CircleParameters, 619
Box
arcs::common_interface, 594

calcJacobian
RobotAlgorithm ( ), 353
calcSafetyParametersCheckSum
RobotConfig ( ), 385
calculateCircleFourthPoint
Math ( ), 46
calibrateCoordinate
Math ( ), 47
calibrateTcpForceSensor
RobotAlgorithm ( ), 354
calibrateTcpForceSensor2
RobotAlgorithm ( ), 355
calibrateTcpForceSensor3
RobotAlgorithm ( ), 356
calibVibrationParams
RobotAlgorithm ( ), 356
calibVibrationParams1
RobotAlgorithm ( ), 356
calibWorkpieceCoordinatePara
RobotAlgorithm ( ), 357
CARTESIAN_DOF
type_def.h, 1186
chanel
arcs::common_interface::RtdeRecipe, 681
changePoseWithXYRef
Math ( ), 48
clearAxisError
AxisInterface ( ), 32
clearBreakPoints
RuntimeMachine ( ), 524
clearCrossConnections
IoControl (IO), 167
clearNamedVariable
RegisterControl ( ), 68
clearPath
MotionControl ( ), 252
clearPreloadPrograms
RuntimeMachine ( ), 524
code
arcs::common_interface::AuboException, 615
arcs::common_interface::RobotMsg, 664
code_
arcs::common_interface::AuboException, 615
codeCompose
arcs::error_stack, 609
cog
arcs::common_interface::VibrationRecalibrationParameter, 698
confirmSafetyParameters
RobotConfig ( ), 385
conveyorTrackCircle
MotionControl ( ), 252
conveyorTrackClearItems
MotionControl ( ), 253
conveyorTrackCreatItem
MotionControl ( ), 253
conveyorTrackLine
MotionControl ( ), 254
conveyorTrackStop
MotionControl ( ), 255
conveyorTrackSwitch
MotionControl ( ), 256
crc32
arcs::common_interface::RobotSafetyParameterRange, 667
cubic
arcs::common_interface::RobotSafetyParameterRange, 668
Cylinder
arcs::common_interface, 594

d_
arcs::common_interface::AuboApi, 612
arcs::common_interface::AxisInterface, 618
arcs::common_interface::ForceControl, 625
arcs::common_interface::GripperInterface, 628
arcs::common_interface::IoControl, 636
arcs::common_interface::Math, 638
arcs::common_interface::MotionControl, 646
arcs::common_interface::RegisterControl, 651
arcs::common_interface::RobotAlgorithm, 655
arcs::common_interface::RobotConfig, 659
arcs::common_interface::RobotInterface, 660
arcs::common_interface::RobotManage, 663
arcs::common_interface::RobotState, 680
arcs::common_interface::RuntimeMachine, 685
arcs::common_interface::Serial, 686
arcs::common_interface::Socket, 687
arcs::common_interface::SyncMove, 692
arcs::common_interface::SystemInfo, 693
arcs::common_interface::Trace, 694
DECL_TO_STRING_FUNC
type_def.h, 1186
deleteCrossConnection
IoControl (IO), 168
deleteGroupInput
IoControl (IO), 169
deleteGroupOutput
IoControl (IO), 169
deleteTask
RuntimeMachine ( ), 525
deleteVirtualEncoder
RegisterControl ( ), 68
deltaPoseAdd
Math ( ), 49
deltaPoseTrans
Math ( ), 49

detachTask
RuntimeMachine ( ), 525
direction
arcs::common_interface::CircleParameters, 619
disableAxisGroup
arcs::common_interface::RobotConfig, 659
disableJointSoftServo
MotionControl ( ), 257
disbaleVibrationSuppress
MotionControl ( ), 258
doc Directory Reference, 588
doc/sdk_overview_en.md, 701
dump
arcs::error_stack, 609
duration
arcs::common_interface::CircleParameters, 619
DynamicsModel
arcs::common_interface, 594

EC_DISCONNECTED
arcs::common_interface, 600
EC_INTERNAL_ERROR
arcs::common_interface, 601
EC_INVAL_SOCKET
arcs::common_interface, 600
EC_INVALID
arcs::common_interface, 601
EC_INVALID_PARAMS
arcs::common_interface, 601
EC_INVALID_REQUEST
arcs::common_interface, 601
EC_METHOD_NOT_FOUND
arcs::common_interface, 601
EC_NOT_LOGINED
arcs::common_interface, 600
EC_PARSE_ERROR
arcs::common_interface, 601
EC_RECV_ERROR
arcs::common_interface, 600
EC_RECV_TIMEOUT
arcs::common_interface, 600
EC_REQUEST_BUSY
arcs::common_interface, 600
EC_SEND_FAILED
arcs::common_interface, 600
EC_SERVER_ERROR
arcs::common_interface, 601
elbow_force
arcs::common_interface::RobotSafetyParameterRange, 668
elbow_speed
arcs::common_interface::RobotSafetyParameterRange, 668
enableAxisGroup
arcs::common_interface::RobotConfig, 659
enableEndCollisionCheck
RobotConfig ( ), 386
enableExtAxis
AxisInterface ( ), 32
enableJointSoftServo
MotionControl ( ), 258
enableVibrationSuppress
MotionControl ( ), 259
enterCritical
RuntimeMachine ( ), 525
ENUM_AuboErrorCodes_DECLARES
arcs::common_interface, 599
type_def.h, 1186
ENUM_AxisModeType_DECLARES
arcs::common_interface, 599
type_def.h, 1186
ENUM_EnvelopingShape_DECLARES
arcs::common_interface, 600
type_def.h, 1186
ENUM_HandleModeType_DECLARES
arcs::common_interface, 601
type_def.h, 1187
ENUM_HandleStateType_DECLARES
arcs::common_interface, 602
type_def.h, 1187
ENUM_ITEM
type_def.h, 1187, 1188
ENUM_JointServoModeType_DECLARES
arcs::common_interface, 602
type_def.h, 1188
ENUM_JointStateType_DECLARES
arcs::common_interface, 602
type_def.h, 1188
ENUM_OperationalModeType_DECLARES
arcs::common_interface, 603
type_def.h, 1189
ENUM_PayloadIdentifyMoveAxis_DECLARES
arcs::common_interface, 603
type_def.h, 1189
ENUM_RobotControlModeType_DECLARES
arcs::common_interface, 604
type_def.h, 1189
ENUM_RobotEmergencyStopType_DECLARES
arcs::common_interface, 604
type_def.h, 1189
ENUM_RobotModeType_DECLARES
arcs::common_interface, 605
type_def.h, 1190
ENUM_RuntimeState_DECLARES
arcs::common_interface, 605
type_def.h, 1190
ENUM_SafeguedStopType_DECLARES
arcs::common_interface, 605
type_def.h, 1190
ENUM_SafetyInputAction_DECLARES
arcs::common_interface, 605
type_def.h, 1190
ENUM_SafetyModeType_DECLARES
arcs::common_interface, 606
type_def.h, 1191

ENUM_SafetyOutputRunState_DECLARES
arcs::common_interface, 606
type_def.h, 1191
ENUM_StandardInputAction_DECLARES
arcs::common_interface, 606
type_def.h, 1191
ENUM_StandardOutputRunState_DECLARES
arcs::common_interface, 606
type_def.h, 1192
ENUM_TaskFrameType_DECLARES
arcs::common_interface, 607
type_def.h, 1192
ENUM_TraceLevel_DECLARES
arcs::common_interface, 607
type_def.h, 1192
envelopings
arcs::common_interface::TrajConfig, 696
EnvelopingShape
arcs::common_interface, 599
ep_args
arcs::common_interface::Enveloping, 622
error_stack.h
_D, 716, 717
_PH1_, 718
_PH2_, 718
_PH3_, 718
_PH4_, 718
ARCS_ERROR_CODES, 718
AUBO_SDK_HAL_ERROR_H, 718
AUBO_SDK_RTM_ERROR_H, 719
AUBO_SDK_SYSTEM_ERROR_H, 719
EX_PEDSTRAL_ERRORS, 719
EX_TOOL_ERRORS, 719
EXT_AXIS_ERRORS, 719
HAL_ERRORS, 719
HARDWARE_INTERFACE_ERRORS, 720
JOINT_ERRORS, 720
JOINT_EX_ERRORS, 720
PEDSTRAL_ERRORS, 720
RTM_ERRORS, 720
SAFETY_INTERFACE_BOARD_ERRORS, 720
SYSTEM_ERRORS, 721
TOOL_ERRORS, 721
error_type
arcs::common_interface, 600
errorCode2Str
arcs::error_stack, 609
ErrorCodes
arcs::error_stack, 608
EX_PEDSTRAL_ERRORS
error_stack.h, 719
hal_error.h, 724
EX_TOOL_ERRORS
error_stack.h, 719
hal_error.h, 724
ExceptionCode
arcs::common_interface, 600
exitCritical
RuntimeMachine ( ), 526
exitHandguideMode
RobotManage ( ), 442
EXT_AXIS_ERRORS
error_stack.h, 719
hal_error.h, 724

fcCalDynamicModel
ForceControl( ), 124
fcDisable
ForceControl( ), 125
fcEnable
ForceControl( ), 125
fcSetSensorLimits
ForceControl( ), 126
fcSetSensorThresholds
ForceControl( ), 127
firmwareUpdate
RobotConfig ( ), 386
followAnotherAxis
AxisInterface ( ), 33
followJoint
MotionControl ( ), 259
followLine
MotionControl ( ), 259
force
arcs::common_interface::GripperStatus, 630
ForceControl
arcs::common_interface::ForceControl, 624
ForceControl( ), 121
fcCalDynamicModel, 124
fcDisable, 125
fcEnable, 125
fcSetSensorLimits, 126
fcSetSensorThresholds, 127
getActualJointPositionsHistory, 128
getDynamicModel, 128
getFcSensorLimits, 129
getFcSensorThresholds, 129
isCondFullfiled, 130
isFcEnabled, 131
isSoftFloatEnabled, 131
resetDamping, 132
resetLpFilter, 132
setCondActive, 133
setCondAdvanced, 134
setCondCylinder, 135
setCondDistance, 136
setCondForce, 136
setCondOrient, 137
setCondPlane, 138
setCondSphere, 139
setCondTcpSpeed, 140
setDamping, 141
setDynamicModel, 142
setDynamicModel1, 143
setDynamicModelContact, 144
setDynamicModelInsert, 145

setDynamicModelSearch, 146
setLpFilter, 146
setSoftFloatParams, 147
setSupvForce, 148
setSupvOrient, 149
setSupvPosBox, 150
setSupvPosCylinder, 150
setSupvPosSphere, 151
setSupvReoriSpeed, 152
setSupvTcpSpeed, 153
setTargetForce, 154
softFloatDisable, 155
softFloatEnable, 155
speedChangeDisable, 156
speedChangeEnable, 156
speedChangeTune, 157
toolContact, 158
ForceControlPtr
arcs::common_interface, 594
ForceControlState
arcs::common_interface, 601
ForceSensorCalibResult
arcs::common_interface, 595
ForceSensorCalibResultWithError
arcs::common_interface, 595
forceTrans
Math ( ), 50
forwardDynamics
RobotAlgorithm ( ), 357
forwardDynamics1
RobotAlgorithm ( ), 358
forwardKinematics
RobotAlgorithm ( ), 359
forwardKinematics1
RobotAlgorithm ( ), 360
forwardKinematicsAll
RobotAlgorithm ( ), 361
forwardToolKinematics
RobotAlgorithm ( ), 362
frame
arcs::common_interface::SpiralParameters, 688
frameAdd
SyncMove ( ), 570
frameAttach
SyncMove ( ), 570
frameConvertPose
SyncMove ( ), 571
frameDelete
SyncMove ( ), 571
frameDeleteAll
SyncMove ( ), 572
frameExist
SyncMove ( ), 572
frameGetChildren
SyncMove ( ), 572
frameGetParent
SyncMove ( ), 573
frameGetPose
SyncMove ( ), 573
frameMove
SyncMove ( ), 574
freedrive
RobotManage ( ), 443
frequency
arcs::common_interface::RtdeRecipe, 681
frictionModelIdentify
RobotAlgorithm ( ), 363

generateDiagnoseFile
RobotManage ( ), 443
generatePayloadIdentifyTraj
RobotAlgorithm ( ), 364
getActualJointPositionsHistory
ForceControl( ), 128
getActualTcpOffset
RobotState ( ), 470
getAdvancePlanContext
RuntimeMachine ( ), 526
getAdvancePtr
RuntimeMachine ( ), 527
getAxisInterface
AuboApi ( ), 26
getAxisModeType
AxisInterface ( ), 33
getAxisNames
AuboApi ( ), 26
getBaseForce
RobotState ( ), 470
getBaseForceOffset
RobotConfig ( ), 387
getBaseForceSensor
arcs::common_interface::RobotState, 679
getBaseForceSensorNames
RobotConfig ( ), 388
getBool
RegisterControl ( ), 69
getBoolInput
RegisterControl ( ), 69
getBoolOutput
RegisterControl ( ), 70
getCollisionLevel
RobotConfig ( ), 389
getCollisionStopType
RobotConfig ( ), 389
getCollisionThreshold
RobotConfig ( ), 390
getConfigurableDigitalInput
IoControl (IO), 170
getConfigurableDigitalInputAction
IoControl (IO), 171
getConfigurableDigitalInputNum
IoControl (IO), 172
getConfigurableDigitalInputs
IoControl (IO), 173
getConfigurableDigitalOutput
IoControl (IO), 174

getConfigurableDigitalOutputNum
IoControl (IO), 175
getConfigurableDigitalOutputRunstate
IoControl (IO), 175
getConfigurableDigitalOutputs
IoControl (IO), 176
getControlBoxHumidity
RobotState ( ), 471
getControlBoxTemperature
RobotState ( ), 471
getControlBoxType
RobotConfig ( ), 391
getControlSoftwareBuildDate
SystemInfo ( ), 579
getControlSoftwareFullVersion
SystemInfo ( ), 579
getControlSoftwareVersionCode
SystemInfo ( ), 580
getControlSoftwareVersionHash
SystemInfo ( ), 580
getControlSystemTime
SystemInfo ( ), 581
getConveyorTrackNextItem
arcs::common_interface::MotionControl, 644
getConveyorTrackQueue
MotionControl ( ), 260
getCycletime
RobotConfig ( ), 392
getDefaultJointAcc
RobotConfig ( ), 392
getDefaultJointSpeed
RobotConfig ( ), 393
getDefaultToolAcc
RobotConfig ( ), 394
getDefaultToolSpeed
RobotConfig ( ), 395
getDeltaPoseBySensorDistance
Math ( ), 50
getDigitalOutputRunstate
IoControl (IO), 177
getDof
RobotConfig ( ), 395
getDouble
RegisterControl ( ), 71
getDoubleInput
RegisterControl ( ), 72
getDoubleOutput
RegisterControl ( ), 73
getDuration
MotionControl ( ), 261
getDynamicModel
ForceControl( ), 128
getElbowPosistion
RobotState ( ), 472
getElbowVelocity
RobotState ( ), 473
getEncDecoderType
IoControl (IO), 178
getEncTickCount
IoControl (IO), 178
getEqradius
MotionControl ( ), 262
getErrorCode
AxisInterface ( ), 33
getExecId
MotionControl ( ), 262
getExecutionStatus
RuntimeMachine ( ), 527
getExecutionStatus1
arcs::common_interface::RuntimeMachine, 684
getExtAxisAcceleration
AxisInterface ( ), 33
getExtAxisBusCurrent
AxisInterface ( ), 34
getExtAxisBusVoltage
AxisInterface ( ), 34
getExtAxisCurrent
AxisInterface ( ), 34
getExtAxisMaxAcceleration
AxisInterface ( ), 34
getExtAxisMaxAccelerationLimit
AxisInterface ( ), 34
getExtAxisMaxPosition
AxisInterface ( ), 35
getExtAxisMaxPositionLimit
AxisInterface ( ), 35
getExtAxisMaxVelocity
AxisInterface ( ), 35
getExtAxisMaxVelocityLimit
AxisInterface ( ), 35
getExtAxisMountingPose
AxisInterface ( ), 35
getExtAxisPose
AxisInterface ( ), 36
getExtAxisPosition
AxisInterface ( ), 36
getExtAxisTemperature
AxisInterface ( ), 36
getExtAxisType
AxisInterface ( ), 36
getExtAxisVelocity
AxisInterface ( ), 36
getExtMinPosition
AxisInterface ( ), 37
getExtMinPositionLimit
AxisInterface ( ), 37
getFcSensorLimits
ForceControl( ), 129
getFcSensorThresholds
ForceControl( ), 129
getFirmwareUpdateProcess
RobotConfig ( ), 396
getFloat
RegisterControl ( ), 73
getFloatInput

RegisterControl ( ), 74
getFloatOutput
RegisterControl ( ), 75
getForceControl
RobotInterface( ), 118
getFreedriveDamp
RobotConfig ( ), 397
getFuturePathPointsJoint
MotionControl ( ), 263
getGravity
RobotConfig ( ), 398
getGripperInterface
AuboApi ( ), 26
getGroupInput
IoControl (IO), 179
getGroupOutput
IoControl (IO), 180
getHandguidDamp
RobotConfig ( ), 398
getHandguideFeature
RobotManage ( ), 444
getHandguideFreeAxes
RobotManage ( ), 445
getHandguideStatus
RobotManage ( ), 445
getHandguideTrigger
RobotManage ( ), 445
getHandleFirmwareVersion
RobotState ( ), 474
getHandleHardwareVersion
RobotState ( ), 474
getHandleIoStatus
IoControl (IO), 181
getHandleMode
RobotState ( ), 475
getHandleState
RobotState ( ), 476
getHandleType
IoControl (IO), 181
getHandleUniqueId
RobotState ( ), 476
getHardwareCustomParameters
RobotConfig ( ), 399
getHomePosition
RobotConfig ( ), 400
getInt16Register
RegisterControl ( ), 76
getInt16RegisterBit
RegisterControl ( ), 77
getInt32
RegisterControl ( ), 78
getInt32Input
RegisterControl ( ), 78
getInt32Output
RegisterControl ( ), 79
getInterfaceVersionCode
SystemInfo ( ), 581
getInterpPtr
RuntimeMachine ( ), 528
getIoControl
RobotInterface( ), 118
getJointAccelerations
RobotState ( ), 477
getJointContactTorques
RobotState ( ), 478
getJointCurrents
RobotState ( ), 479
getJointFirmwareVersions
RobotState ( ), 479
getJointGravityTorques
RobotState ( ), 480
getJointHardwareVersions
RobotState ( ), 481
getJointMaxAccelerations
RobotConfig ( ), 400
getJointMaxPositions
RobotConfig ( ), 401
getJointMaxSpeeds
RobotConfig ( ), 402
getJointMinPositions
RobotConfig ( ), 402
getJointPositions
RobotState ( ), 482
getJointPositionsHistory
RobotState ( ), 482
getJointServoMode
RobotState ( ), 483
getJointSpeeds
RobotState ( ), 483
getJointState
RobotState ( ), 484
getJointTargetAccelerations
RobotState ( ), 485
getJointTargetCurrents
RobotState ( ), 485
getJointTargetPositions
RobotState ( ), 486
getJointTargetSpeeds
RobotState ( ), 487
getJointTargetTorques
RobotState ( ), 488
getJointTemperatures
RobotState ( ), 488
getJointTorqueSensors
RobotState ( ), 489
getJointUniqueIds
RobotState ( ), 490
getJointVoltages
RobotState ( ), 491
getKinematicsCompensate
RobotConfig ( ), 403
getKinematicsParam
RobotConfig ( ), 404
getLimitJointMaxAccelerations
RobotConfig ( ), 405
getLimitJointMaxPositions

RobotConfig ( ), 406
getLimitJointMaxSpeeds
RobotConfig ( ), 407
getLimitJointMinPositions
RobotConfig ( ), 407
getLimitTcpMaxSpeed
RobotConfig ( ), 408
getLookAheadSize
MotionControl ( ), 264
getMainCurrent
RobotState ( ), 491
getMainPtr
RuntimeMachine ( ), 528
getMainVoltage
RobotState ( ), 492
getMasterBoardFirmwareVersion
RobotState ( ), 493
getMasterBoardHardwareVersion
RobotState ( ), 494
getMasterBoardUniqueId
RobotState ( ), 494
getMath
AuboApi ( ), 26
getModbusDeviceStatus
RegisterControl ( ), 80
getMotionControl
RobotInterface( ), 119
getMotionLeftTime
MotionControl ( ), 264
getMountingPose
RobotConfig ( ), 409
getName
RobotConfig ( ), 410
getNamedVariableType
RegisterControl ( ), 81
getOperationalMode
RobotManage ( ), 446
getPauseJointPositions
MotionControl ( ), 265
getPayload
RobotConfig ( ), 410
getPayloadIdentifyResult
RobotAlgorithm ( ), 365
getPedestalBrakeResTemp
RobotState ( ), 495
getPedestalFirmwareVersion
RobotState ( ), 496
getPedestalHardwareVersion
RobotState ( ), 497
getPedestalUniqueId
RobotState ( ), 497
getPlanContext
RuntimeMachine ( ), 529
getPowerlossBoardFirmwareVersion
RobotState ( ), 498
getPowerlossBoardHardwareVersion
RobotState ( ), 499
getPowerlossBoardUniqueId
RobotState ( ), 500
getPreloadProgram
RuntimeMachine ( ), 529
getProgress
MotionControl ( ), 266
getQueueSize
MotionControl ( ), 267
getRecordCache
RobotManage ( ), 446
getReductionRatio
AxisInterface ( ), 37
getRegisterControl
AuboApi ( ), 27
getResumeMode
MotionControl ( ), 267
getRobotAlgorithm
RobotInterface( ), 119
getRobotBaseParent
arcs::common_interface::RobotConfig, 659
getRobotConfig
RobotInterface( ), 119
getRobotConfiguration
RobotAlgorithm ( ), 365
getRobotControlMode
RobotManage ( ), 447
getRobotCurrent
RobotState ( ), 500
getRobotEmergencyStopSource
RobotConfig ( ), 411
getRobotInterface
AuboApi ( ), 27
getRobotManage
RobotInterface( ), 120
getRobotModeType
RobotState ( ), 501
getRobotNames
AuboApi ( ), 28
getRobotState
RobotInterface( ), 120
getRobotSubType
RobotConfig ( ), 411
getRobotType
RobotConfig ( ), 412
getRobotVoltage
RobotState ( ), 502
getRuntimeMachine
AuboApi ( ), 28
getRuntimeState
arcs::common_interface::RuntimeMachine, 684
getSafeguardStopSource
RobotConfig ( ), 413
getSafeguardStopType
RobotConfig ( ), 413
getSafetyModeType
RobotState ( ), 503
getSafetyParametersCheckSum
RobotConfig ( ), 414

getSelectedTcpForceSensorName
RobotConfig ( ), 414
getSerial
AuboApi ( ), 29
getServoModeSelect
MotionControl ( ), 268
getSlaveBoardFirmwareVersion
RobotState ( ), 503
getSlaveBoardHardwareVersion
RobotState ( ), 504
getSlaveBoardUniqueId
RobotState ( ), 505
getSlowDownFraction
RobotConfig ( ), 415
getSlowDownLevel
RobotState ( ), 506
getSocket
AuboApi ( ), 29
getSpeedFraction
MotionControl ( ), 268
getStandardAnalogInput
IoControl (IO), 182
getStandardAnalogInputDomain
IoControl (IO), 183
getStandardAnalogInputNum
IoControl (IO), 184
getStandardAnalogOutput
IoControl (IO), 185
getStandardAnalogOutputDomain
IoControl (IO), 186
getStandardAnalogOutputNum
IoControl (IO), 187
getStandardAnalogOutputRunstate
IoControl (IO), 188
getStandardDigitalInput
IoControl (IO), 189
getStandardDigitalInputAction
IoControl (IO), 190
getStandardDigitalInputNum
IoControl (IO), 190
getStandardDigitalInputs
IoControl (IO), 191
getStandardDigitalOutput
IoControl (IO), 192
getStandardDigitalOutputNum
IoControl (IO), 193
getStandardDigitalOutputRunstate
IoControl (IO), 194
getStandardDigitalOutputs
IoControl (IO), 195
getStaticLinkInputNum
IoControl (IO), 195
getStaticLinkInputs
IoControl (IO), 196
getStaticLinkOutputNum
IoControl (IO), 197
getStaticLinkOutputs
IoControl (IO), 198
getStatus
RuntimeMachine ( ), 530
getString
RegisterControl ( ), 81
getSyncMove
AuboApi ( ), 29
RobotInterface( ), 120
getSystemInfo
AuboApi ( ), 30
getTargetTcpPose
RobotState ( ), 506
getTaskQueueSize
RuntimeMachine ( ), 530
getTcpForce
RobotState ( ), 507
getTcpForceOffset
RobotConfig ( ), 416
getTcpForceSensorNames
RobotConfig ( ), 416
getTcpForceSensorPose
RobotConfig ( ), 417
getTcpForceSensors
RobotState ( ), 507
getTcpForceSensorStatus
RobotState ( ), 508
getTcpMaxAccelerations
RobotConfig ( ), 417
getTcpMaxLinearVelocity
MotionControl ( ), 269
getTcpMaxSpeeds
RobotConfig ( ), 418
getTcpOffset
RobotConfig ( ), 419
getTcpPose
RobotState ( ), 509
getTcpSpeed
RobotState ( ), 510
getTcpTargetForce
RobotState ( ), 510
getTcpTargetPose
RobotState ( ), 511
getTcpTargetSpeed
RobotState ( ), 512
getTimer
RuntimeMachine ( ), 531
getToolAnalogInput
IoControl (IO), 198
getToolAnalogInputDomain
IoControl (IO), 199
getToolAnalogInputNum
IoControl (IO), 200
getToolAnalogOutput
IoControl (IO), 201
getToolAnalogOutputDomain
IoControl (IO), 202
getToolAnalogOutputNum
IoControl (IO), 203
getToolAnalogOutputRunstate

IoControl (IO), 204
getToolButtonStatus
IoControl (IO), 204
getToolCommMode
RobotState ( ), 513
getToolDigitalInput
IoControl (IO), 205
getToolDigitalInputAction
IoControl (IO), 206
getToolDigitalInputNum
IoControl (IO), 207
getToolDigitalInputs
IoControl (IO), 208
getToolDigitalOutput
IoControl (IO), 208
getToolDigitalOutputNum
IoControl (IO), 209
getToolDigitalOutputRunstate
IoControl (IO), 210
getToolDigitalOutputs
IoControl (IO), 211
getToolFirmwareVersion
RobotState ( ), 513
getToolHardwareVersion
RobotState ( ), 514
getToolPose
RobotState ( ), 515
getToolUniqueId
RobotState ( ), 516
getToolVoltageOutputDomain
IoControl (IO), 212
getTrace
AuboApi ( ), 30
RobotInterface( ), 121
getTrajectoryQueueSize
MotionControl ( ), 270
getTriggInterrupts
RuntimeMachine ( ), 531
getVecChar
RegisterControl ( ), 82
getVecDouble
RegisterControl ( ), 83
getVecFloat
RegisterControl ( ), 84
getVecInt32
RegisterControl ( ), 85
getWatchDogAction
RegisterControl ( ), 85
getWatchDogTimeout
RegisterControl ( ), 86
getWeaveDirection
MotionControl ( ), 271
getWorkObjectHold
MotionControl ( ), 272
gotoLine
RuntimeMachine ( ), 532
gripperAdd
GripperInterface ( ), 43
gripperConnect
GripperInterface ( ), 43
gripperDelete
GripperInterface ( ), 43
gripperDisconnect
GripperInterface ( ), 43
gripperEnable
GripperInterface ( ), 43
gripperGetAngle
arcs::common_interface::GripperInterface, 626
gripperGetForce
arcs::common_interface::GripperInterface, 626
gripperGetHardwareVersion
GripperInterface ( ), 43
gripperGetMotionState
arcs::common_interface::GripperInterface, 626
gripperGetMountPose
arcs::common_interface::GripperInterface, 626
gripperGetNames
GripperInterface ( ), 43
gripperGetObjectDetection
arcs::common_interface::GripperInterface, 627
gripperGetPosition
arcs::common_interface::GripperInterface, 627
gripperGetRVelocity
arcs::common_interface::GripperInterface, 627
gripperGetSoftwareVersion
arcs::common_interface::GripperInterface, 627
gripperGetStatusCode
GripperInterface ( ), 43
gripperGetSupportedModels
GripperInterface ( ), 43
gripperGetTemperature
arcs::common_interface::GripperInterface, 627
gripperGetTorque
arcs::common_interface::GripperInterface, 627
gripperGetVelocity
arcs::common_interface::GripperInterface, 627
gripperGetVoltage
arcs::common_interface::GripperInterface, 627
gripperGetWorkMode
GripperInterface ( ), 44
GripperInterface
arcs::common_interface::GripperInterface, 626
GripperInterface ( ), 42
gripperAdd, 43
gripperConnect, 43
gripperDelete, 43
gripperDisconnect, 43
gripperEnable, 43
gripperGetHardwareVersion, 43
gripperGetNames, 43
gripperGetStatusCode, 43
gripperGetSupportedModels, 43
gripperGetWorkMode, 44
gripperIsConnected, 44
gripperIsEnabled, 44
gripperMove, 44

gripperRename, 44
gripperResetSlaveId, 44
gripperScanDevices, 44
gripperSetMountPose, 44
gripperSetPosition, 45
gripperSetWorkMode, 45
gripperStop, 45
GripperInterfacePtr
arcs::common_interface, 595
gripperIsConnected
GripperInterface ( ), 44
gripperIsEnabled
GripperInterface ( ), 44
gripperMove
GripperInterface ( ), 44
gripperRename
GripperInterface ( ), 44
gripperResetSlaveId
GripperInterface ( ), 44
gripperScanDevices
GripperInterface ( ), 44
gripperSetAngle
arcs::common_interface::GripperInterface, 628
gripperSetForce
arcs::common_interface::GripperInterface, 628
gripperSetMountPose
GripperInterface ( ), 44
gripperSetPosition
GripperInterface ( ), 45
gripperSetRVelocity
arcs::common_interface::GripperInterface, 628
gripperSetTorque
arcs::common_interface::GripperInterface, 628
gripperSetVelocity
arcs::common_interface::GripperInterface, 628
gripperSetWorkMode
GripperInterface ( ), 45
GripperStatusVector
arcs::common_interface, 595
gripperStop
GripperInterface ( ), 45

hal_error.h
EX_PEDSTRAL_ERRORS, 724
EX_TOOL_ERRORS, 724
EXT_AXIS_ERRORS, 724
HAL_ERRORS, 724
HARDWARE_INTERFACE_ERRORS, 725
JOINT_ERRORS, 725
JOINT_EX_ERRORS, 725
PEDSTRAL_ERRORS, 725
SAFETY_INTERFACE_BOARD_ERRORS, 725
TOOL_ERRORS, 726
HAL_ERRORS
error_stack.h, 719
hal_error.h, 724
handguideMode
RobotManage ( ), 448
HandleModeType
arcs::common_interface, 601
HandleStateType
arcs::common_interface, 601
HARDWARE_INTERFACE_ERRORS
error_stack.h, 720
hal_error.h, 725
hasBaseForceSensor
RobotConfig ( ), 419
hasEncoderSensor
IoControl (IO), 212
hasItemOnConveyorToTrack
MotionControl ( ), 273
hasNamedVariable
RegisterControl ( ), 86
hasTcpForceSensor
RobotConfig ( ), 420
helix
arcs::common_interface::CircleParameters, 620
arcs::common_interface::SpiralParameters, 689
homMatrixToPose
Math ( ), 51

include Directory Reference, 589
include/aubo Directory Reference, 587
include/aubo/aubo_api.h, 701, 702
include/aubo/axis_interface.h, 706, 707
include/aubo/error_stack Directory Reference, 588
include/aubo/error_stack/error_stack.h, 715, 722
include/aubo/error_stack/hal_error.h, 723, 727
include/aubo/error_stack/rtm_error.h, 733, 734
include/aubo/error_stack/system_error.h, 737, 738
include/aubo/gripper_interface.h, 739, 740
include/aubo/math.h, 744, 745
include/aubo/register_control.h, 758, 760
include/aubo/robot Directory Reference, 589
include/aubo/robot/force_control.h, 801, 803
include/aubo/robot/io_control.h, 833, 835
include/aubo/robot/motion_control.h, 901, 903
include/aubo/robot/robot_algorithm.h, 981, 983
include/aubo/robot/robot_config.h, 1007, 1009
include/aubo/robot/robot_manage.h, 1053, 1055
include/aubo/robot/robot_state.h, 1077, 1078
include/aubo/robot_interface.h, 1119, 1120
include/aubo/runtime_machine.h, 1125, 1126
include/aubo/serial.h, 1144, 1146
include/aubo/socket.h, 1150, 1152
include/aubo/sync_move.h, 1159, 1161
include/aubo/system_info.h, 1172, 1174
include/aubo/trace.h, 1178, 1179
include/aubo/type_def.h, 1182, 1194
inertia
arcs::common_interface::VibrationRecalibrationParameter, 698
init_joint
arcs::common_interface::TrajConfig, 696

internal_error
arcs::common_interface, 600
interpolatePose
Math ( ), 51
invalid
arcs::common_interface, 600
invalid_params
arcs::common_interface, 600
invalid_request
arcs::common_interface, 600
inverseKinematics
RobotAlgorithm ( ), 367
inverseKinematics1
RobotAlgorithm ( ), 368
inverseKinematics2
RobotAlgorithm ( ), 369
inverseKinematicsAll
RobotAlgorithm ( ), 370
inverseKinematicsAll1
RobotAlgorithm ( ), 371
inverseToolKinematics
RobotAlgorithm ( ), 372
inverseToolKinematicsAll
RobotAlgorithm ( ), 373
IoControl
arcs::common_interface::IoControl, 636
IoControl (IO), 158
addCrossConnection, 163
addGroupInput, 165
addGroupOutput, 166
clearCrossConnections, 167
deleteCrossConnection, 168
deleteGroupInput, 169
deleteGroupOutput, 169
getConfigurableDigitalInput, 170
getConfigurableDigitalInputAction, 171
getConfigurableDigitalInputNum, 172
getConfigurableDigitalInputs, 173
getConfigurableDigitalOutput, 174
getConfigurableDigitalOutputNum, 175
getConfigurableDigitalOutputRunstate, 175
getConfigurableDigitalOutputs, 176
getDigitalOutputRunstate, 177
getEncDecoderType, 178
getEncTickCount, 178
getGroupInput, 179
getGroupOutput, 180
getHandleIoStatus, 181
getHandleType, 181
getStandardAnalogInput, 182
getStandardAnalogInputDomain, 183
getStandardAnalogInputNum, 184
getStandardAnalogOutput, 185
getStandardAnalogOutputDomain, 186
getStandardAnalogOutputNum, 187
getStandardAnalogOutputRunstate, 188
getStandardDigitalInput, 189
getStandardDigitalInputAction, 190
getStandardDigitalInputNum, 190
getStandardDigitalInputs, 191
getStandardDigitalOutput, 192
getStandardDigitalOutputNum, 193
getStandardDigitalOutputRunstate, 194
getStandardDigitalOutputs, 195
getStaticLinkInputNum, 195
getStaticLinkInputs, 196
getStaticLinkOutputNum, 197
getStaticLinkOutputs, 198
getToolAnalogInput, 198
getToolAnalogInputDomain, 199
getToolAnalogInputNum, 200
getToolAnalogOutput, 201
getToolAnalogOutputDomain, 202
getToolAnalogOutputNum, 203
getToolAnalogOutputRunstate, 204
getToolButtonStatus, 204
getToolDigitalInput, 205
getToolDigitalInputAction, 206
getToolDigitalInputNum, 207
getToolDigitalInputs, 208
getToolDigitalOutput, 208
getToolDigitalOutputNum, 209
getToolDigitalOutputRunstate, 210
getToolDigitalOutputs, 211
getToolVoltageOutputDomain, 212
hasEncoderSensor, 212
isToolIoInput, 213
setAnalogOutputAfterEStopDefault, 214
setConfigurableDigitalInputAction, 215
setConfigurableDigitalOutput, 216
setConfigurableDigitalOutputAfterEStop, 217
setConfigurableDigitalOutputPulse, 218
setConfigurableDigitalOutputRunstate, 219
setDigitalInputActionDefault, 220
setDigitalOutputAfterEStopDefault, 221
setDigitalOutputRunstate, 222
setDigitalOutputRunstateDefault, 222
setEncDecoderType, 223
setEncTickCount, 224
setGroupOutput, 224
setStandardAnalogInputDomain, 225
setStandardAnalogOutput, 226
setStandardAnalogOutputAfterEStop, 227
setStandardAnalogOutputDomain, 228
setStandardAnalogOutputRunstate, 229
setStandardDigitalInputAction, 230
setStandardDigitalOutput, 231
setStandardDigitalOutputAfterEStop, 232
setStandardDigitalOutputPulse, 233
setStandardDigitalOutputRunstate, 234
setToolAnalogInputDomain, 235
setToolAnalogOutput, 236
setToolAnalogOutputDomain, 237
setToolAnalogOutputRunstate, 238
setToolDigitalInputAction, 239
setToolDigitalOutput, 240

setToolDigitalOutputPulse, 241
setToolDigitalOutputRunstate, 242
setToolIoInput, 243
setToolVoltageOutputDomain, 244
unwindEncDeltaTickCount, 245
IoControlPtr
arcs::common_interface, 595
is_connected
arcs::common_interface::GripperStatus, 630
is_enabled
arcs::common_interface::GripperStatus, 630
isBackdriveEnabled
RobotManage ( ), 449
isBlending
MotionControl ( ), 274
isCollisionOccurred
RobotState ( ), 516
isCondFullfiled
ForceControl( ), 130
isConveyorTrackExceed
arcs::common_interface::MotionControl, 645
isConveyorTrackSync
MotionControl ( ), 274
isEndCollisionCheckEnabled
RobotConfig ( ), 421
isFcEnabled
ForceControl( ), 131
isFreedriveEnabled
RobotManage ( ), 449
isHandguideEnabled
RobotManage ( ), 450
isJointSoftServoEnabled
MotionControl ( ), 275
isLinkModeEnabled
RobotManage ( ), 451
isPowerOn
RobotState ( ), 517
isServoModeEnabled
MotionControl ( ), 276
isSimulationEnabled
RobotManage ( ), 452
isSoftFloatEnabled
ForceControl( ), 131
isSpeedFractionCritical
MotionControl ( ), 276
isSteady
RobotState ( ), 518
isSupportedTimeOptimal
MotionControl ( ), 277
isSyncMoveOn
SyncMove ( ), 574
isTaskAlive
RuntimeMachine ( ), 532
isTeachPendantEnabled
RobotState ( ), 518
isTimeOptimalEnabled
MotionControl ( ), 278
isToolFlangeEnabled
RobotState ( ), 519
isToolIoInput
IoControl (IO), 213
isWithinSafetyLimits
RobotState ( ), 520

JOINT_ERRORS
error_stack.h, 720
hal_error.h, 725
JOINT_EX_ERRORS
error_stack.h, 720
hal_error.h, 725
joint_torque
arcs::common_interface::RobotSafetyParameterRange, 668
jointOffsetDisable
MotionControl ( ), 279
jointOffsetEnable
MotionControl ( ), 279
jointOffsetSet
MotionControl ( ), 280
JointServoModeType
arcs::common_interface, 602
JointStateType
arcs::common_interface, 602

level
arcs::common_interface::RobotMsg, 664
loadProgram
RuntimeMachine ( ), 533
lockRobotBrake
RobotManage ( ), 453
loop_times
arcs::common_interface::CircleParameters, 620
lower_joint_bound
arcs::common_interface::TrajConfig, 696

mass
arcs::common_interface::VibrationRecalibrationParameter, 698
Math
arcs::common_interface::Math, 638
Math ( ), 45
calculateCircleFourthPoint, 46
calibrateCoordinate, 47
changePoseWithXYRef, 48
deltaPoseAdd, 49
deltaPoseTrans, 49
forceTrans, 50
getDeltaPoseBySensorDistance, 50
homMatrixToPose, 51
interpolatePose, 51
poseAdd, 52
poseAngleDistance, 53
poseDistance, 53
poseEqual, 54
poseInverse, 55
poseRotation, 55

poseSub, 56
poseToHomMatrix, 57
poseTrans, 57
poseTransInv, 58
quaternionToRpy, 59
rpyToQuaternion, 60
tcpOffsetIdentify, 61
transferRefFrame, 61
MathPtr
arcs::common_interface, 595
max_acceleration
arcs::common_interface::TrajConfig, 696
max_velocity
arcs::common_interface::TrajConfig, 696
MB_ERR_ACKNOWLEDGE
register_control.h, 759
MB_ERR_DISCONNECTED
register_control.h, 759
MB_ERR_ILLEGAL_DATA_ADDRESS
register_control.h, 759
MB_ERR_ILLEGAL_DATA_VALUE
register_control.h, 759
MB_ERR_ILLEGAL_FUNCTION
register_control.h, 759
MB_ERR_NOT_INIT
register_control.h, 759
MB_ERR_SLAVE_DEVICE_BUSY
register_control.h, 759
MB_ERR_SLAVE_DEVICE_FAILURE
register_control.h, 759
message_
arcs::common_interface::AuboException, 615
method_not_found
arcs::common_interface, 600
mod
arcs::error_stack, 609
modbusAddSignal
RegisterControl ( ), 87
modbusDeleteAllSignals
RegisterControl ( ), 88
modbusDeleteSignal
RegisterControl ( ), 89
ModbusErrorNum
register_control.h, 759
modbusGetSignalError
RegisterControl ( ), 89
modbusGetSignalErrors
RegisterControl ( ), 90
modbusGetSignalIndex
RegisterControl ( ), 90
modbusGetSignalNames
RegisterControl ( ), 91
modbusGetSignalStatus
RegisterControl ( ), 91
modbusGetSignalTypes
RegisterControl ( ), 92
modbusGetSignalValues
RegisterControl ( ), 92
modbusSendCustomCommand
RegisterControl ( ), 93
modbusSetDigitalInputAction
RegisterControl ( ), 94
modbusSetOutputRunstate
RegisterControl ( ), 95
modbusSetOutputRunstate1
RegisterControl ( ), 96
modbusSetOutputSignal
RegisterControl ( ), 97
modbusSetOutputSignal1
RegisterControl ( ), 97
modbusSetOutputSignalPulse
RegisterControl ( ), 98
modbusSetOutputSignalWithTimeout
arcs::common_interface::RegisterControl, 650
modbusSetSignalUpdateFrequency
RegisterControl ( ), 99
momentum
arcs::common_interface::RobotSafetyParameterRange, 668
motion_state
arcs::common_interface::GripperStatus, 630
MotionControl
arcs::common_interface::MotionControl, 644
MotionControl ( ), 246
clearPath, 252
conveyorTrackCircle, 252
conveyorTrackClearItems, 253
conveyorTrackCreatItem, 253
conveyorTrackLine, 254
conveyorTrackStop, 255
conveyorTrackSwitch, 256
disableJointSoftServo, 257
disbaleVibrationSuppress, 258
enableJointSoftServo, 258
enableVibrationSuppress, 259
followJoint, 259
followLine, 259
getConveyorTrackQueue, 260
getDuration, 261
getEqradius, 262
getExecId, 262
getFuturePathPointsJoint, 263
getLookAheadSize, 264
getMotionLeftTime, 264
getPauseJointPositions, 265
getProgress, 266
getQueueSize, 267
getResumeMode, 267
getServoModeSelect, 268
getSpeedFraction, 268
getTcpMaxLinearVelocity, 269
getTrajectoryQueueSize, 270
getWeaveDirection, 271
getWorkObjectHold, 272
hasItemOnConveyorToTrack, 273
isBlending, 274

isConveyorTrackSync, 274
isJointSoftServoEnabled, 275
isServoModeEnabled, 276
isSpeedFractionCritical, 276
isSupportedTimeOptimal, 277
isTimeOptimalEnabled, 278
jointOffsetDisable, 279
jointOffsetEnable, 279
jointOffsetSet, 280
moveCircle, 281
moveCircle2, 282
moveCircleWithAxisGroup, 283
moveIntersection, 284
moveJoint, 285
moveJointWithAxisGroup, 287
moveLine, 287
moveLineWithAxisGroup, 289
movePathBuffer, 289
moveProcess, 290
moveSpiral, 291
moveSpline, 292
pathBufferAlloc, 293
pathBufferAppend, 294
pathBufferEval, 295
pathBufferFilter, 296
pathBufferFree, 297
pathBufferList, 298
pathBufferValid, 299
pathOffsetCoordinate, 300
pathOffsetDisable, 301
pathOffsetEnable, 302
pathOffsetLimits, 303
pathOffsetSet, 303
pathOffsetSupv, 304
resetTcpMaxLinearVelocity, 305
restoPath, 306
resumeMoveJoint, 307
resumeMoveLine, 308
resumeSpeedJoint, 309
resumeSpeedLine, 310
resumeStopJoint, 311
resumeStopLine, 312
servoCartesian, 313
servoCartesianWithAxes, 315
servoJoint, 316
servoJointWithAxes, 317
setCirclePathMode, 318
setConveyorTrackCompensate, 319
setConveyorTrackEncoder, 320
setConveyorTrackLimit, 321
setConveyorTrackSensorOffset, 322
setConveyorTrackStartWindow, 323
setConveyorTrackSyncSeparation, 324
setConveyorTrackVelThreshold, 325
setEndPath, 326
setEqradius, 327
setFuturePointSamplePeriod, 328
setLookAheadSize, 329
setResumeStartPoint, 329
setServoMode, 330
setServoModeSelect, 331
setSpeedFraction, 332
setTcpMaxLinearVelocity, 333
setTimeOptimalEnable, 334
setWorkObjectHold, 334
speedFractionCritical, 335
speedFractionCritical1, 336
speedJoint, 337
speedLine, 338
startMove, 340
stopJoint, 340
stopLine, 341
stopMove, 342
storePath, 343
trackCartesian, 344
trackJoint, 344
weaveEnd, 345
weaveStart, 346
weaveUpdateParameters, 348
MotionControlPtr
arcs::common_interface, 595
move_axis
arcs::common_interface::TrajConfig, 696
moveCircle
MotionControl ( ), 281
moveCircle2
MotionControl ( ), 282
moveCircleWithAxisGroup
MotionControl ( ), 283
moveExtJoint
AxisInterface ( ), 37
moveIntersection
MotionControl ( ), 284
moveJoint
MotionControl ( ), 285
moveJointWithAxisGroup
MotionControl ( ), 287
moveLine
MotionControl ( ), 287
moveLineWithAxisGroup
MotionControl ( ), 289
movePathBuffer
MotionControl ( ), 289
moveProcess
MotionControl ( ), 290
moveSpiral
MotionControl ( ), 291
moveSpline
MotionControl ( ), 292

name
arcs::common_interface::GripperStatus, 630
needVibrationRecalib
RobotAlgorithm ( ), 373
newTask
RuntimeMachine ( ), 533
None

arcs::common_interface, 604
nop
RuntimeMachine ( ), 534
notify
arcs::common_interface::Trace, 694

obj_coord
arcs::common_interface::WObjectData, 699
object_detection
arcs::common_interface::GripperStatus, 630
OperationalModeType
arcs::common_interface, 602
operator<<
arcs::common_interface, 607, 608
orig
arcs::common_interface::RobotSafetyParameterRange, 668

params
arcs::common_interface::RobotSafetyParameterRange, 668
parse_error
arcs::common_interface, 600
Path
arcs::common_interface, 604
pathBlend3Points
RobotAlgorithm ( ), 374
PathBuffer_CubicSpline
arcs::common_interface, 603
PathBuffer_JointBSpline
arcs::common_interface, 603
PathBuffer_JointBSplineC
arcs::common_interface, 603
PathBuffer_JointSpline
arcs::common_interface, 603
PathBuffer_JointSplineC
arcs::common_interface, 603
PathBuffer_TOPPRA
arcs::common_interface, 603
pathBufferAlloc
MotionControl ( ), 293
pathBufferAppend
MotionControl ( ), 294
pathBufferEval
MotionControl ( ), 295
pathBufferFilter
MotionControl ( ), 296
pathBufferFree
MotionControl ( ), 297
pathBufferList
MotionControl ( ), 298
PathBufferType
arcs::common_interface, 603
pathBufferValid
MotionControl ( ), 299
pathMovej
arcs::common_interface::RobotAlgorithm, 654
pathMoveS
RobotAlgorithm ( ), 374
pathOffsetCoordinate
MotionControl ( ), 300
pathOffsetDisable
MotionControl ( ), 301
pathOffsetEnable
MotionControl ( ), 302
pathOffsetLimits
MotionControl ( ), 303
pathOffsetSet
MotionControl ( ), 303
pathOffsetSupv
MotionControl ( ), 304
pause
RuntimeMachine ( ), 534
pauseRecord
RobotManage ( ), 453
pauseRecordCache
RobotManage ( ), 454
Payload
arcs::common_interface, 596
payloadCalculateFinished
RobotAlgorithm ( ), 375
payloadIdentify
RobotAlgorithm ( ), 375
payloadIdentify1
RobotAlgorithm ( ), 376
PayloadIdentifyMoveAxis
arcs::common_interface, 603
payloadIdentifyTrajGenFinished
RobotAlgorithm ( ), 377
PEDSTRAL_ERRORS
error_stack.h, 720
hal_error.h, 725
peek
Trace ( ), 583
plane
arcs::common_interface::RobotSafetyParameterRange, 669
arcs::common_interface::SpiralParameters, 689
planes
arcs::common_interface::RobotSafetyParameterRange, 669
points
arcs::common_interface::VibrationRecalibrationParameter, 698
popup
Trace ( ), 584
pose_to
arcs::common_interface::CircleParameters, 620
pose_via
arcs::common_interface::CircleParameters, 620
poseAdd
Math ( ), 52
poseAngleDistance
Math ( ), 53

poseDistance
Math ( ), 53
poseEqual
Math ( ), 54
poseInverse
Math ( ), 55
poseRotation
Math ( ), 55
poseSub
Math ( ), 56
poseToHomMatrix
Math ( ), 57
poseTrans
Math ( ), 57
poseTransInv
Math ( ), 58
position
arcs::common_interface::GripperStatus, 630
power
arcs::common_interface::RobotSafetyParameterRange, 669
poweroff
RobotManage ( ), 455
poweroffExtAxis
AxisInterface ( ), 38
poweron
RobotManage ( ), 455
poweronExtAxis
AxisInterface ( ), 38
preloadProgram
RuntimeMachine ( ), 534

qdmax
arcs::common_interface::RobotSafetyParameterRange, 669
qmax
arcs::common_interface::RobotSafetyParameterRange, 669
qmin
arcs::common_interface::RobotSafetyParameterRange, 669
quaternionToRpy
Math ( ), 59

r_velocity
arcs::common_interface::GripperStatus, 631
recordCacheFree
RobotManage ( ), 456
reduced_entry_distance
arcs::common_interface::RobotSafetyParameterRange, 669
reduced_entry_time
arcs::common_interface::RobotSafetyParameterRange, 670
RefFrameType
arcs::common_interface, 603
register_control.h
MB_ERR_ACKNOWLEDGE, 759
MB_ERR_DISCONNECTED, 759
MB_ERR_ILLEGAL_DATA_ADDRESS, 759
MB_ERR_ILLEGAL_DATA_VALUE, 759
MB_ERR_ILLEGAL_FUNCTION, 759
MB_ERR_NOT_INIT, 759
MB_ERR_SLAVE_DEVICE_BUSY, 759
MB_ERR_SLAVE_DEVICE_FAILURE, 759
ModbusErrorNum, 759
RegisterControl
arcs::common_interface::RegisterControl, 650
RegisterControl ( ), 62
addFixedModbusGroup, 66
addInt32RegEncoder, 67
addModbusEncoder, 67
clearNamedVariable, 68
deleteVirtualEncoder, 68
getBool, 69
getBoolInput, 69
getBoolOutput, 70
getDouble, 71
getDoubleInput, 72
getDoubleOutput, 73
getFloat, 73
getFloatInput, 74
getFloatOutput, 75
getInt16Register, 76
getInt16RegisterBit, 77
getInt32, 78
getInt32Input, 78
getInt32Output, 79
getModbusDeviceStatus, 80
getNamedVariableType, 81
getString, 81
getVecChar, 82
getVecDouble, 83
getVecFloat, 84
getVecInt32, 85
getWatchDogAction, 85
getWatchDogTimeout, 86
hasNamedVariable, 86
modbusAddSignal, 87
modbusDeleteAllSignals, 88
modbusDeleteSignal, 89
modbusGetSignalError, 89
modbusGetSignalErrors, 90
modbusGetSignalIndex, 90
modbusGetSignalNames, 91
modbusGetSignalStatus, 91
modbusGetSignalTypes, 92
modbusGetSignalValues, 92
modbusSendCustomCommand, 93
modbusSetDigitalInputAction, 94
modbusSetOutputRunstate, 95
modbusSetOutputRunstate1, 96
modbusSetOutputSignal, 97
modbusSetOutputSignal1, 97
modbusSetOutputSignalPulse, 98

modbusSetSignalUpdateFrequency, 99
setBool, 100
setBoolInput, 101
setBoolOutput, 102
setDouble, 102
setDoubleInput, 103
setDoubleOutput, 104
setFloat, 105
setFloatInput, 106
setFloatOutput, 107
setInt16Register, 107
setInt16RegisterBit, 108
setInt32, 109
setInt32Input, 110
setInt32Output, 111
setString, 111
setVecChar, 112
setVecDouble, 113
setVecFloat, 114
setVecInt32, 115
setWatchDog, 115
variableUpdated, 116
RegisterControlPtr
arcs::common_interface, 596
releaseRobotBrake
RobotManage ( ), 457
remote_tool
arcs::common_interface::WObjectData, 700
removeBreakPoint
RuntimeMachine ( ), 535
removeCollisionObject
RobotAlgorithm ( ), 377
resetDamping
ForceControl( ), 132
resetLpFilter
ForceControl( ), 132
resetTcpMaxLinearVelocity
MotionControl ( ), 305
restartInterfaceBoard
RobotManage ( ), 458
restoPath
MotionControl ( ), 306
restrict_elbow
arcs::common_interface::RobotSafetyParameterRange, 670
ResultWithErrno
arcs::common_interface, 596
ResultWithErrno1
arcs::common_interface, 596
ResultWithErrno2
arcs::common_interface, 596
ResultWithErrno3
arcs::common_interface, 596
resume
RuntimeMachine ( ), 535
resumeMoveJoint
MotionControl ( ), 307
resumeMoveLine
MotionControl ( ), 308
resumeSpeedJoint
MotionControl ( ), 309
resumeSpeedLine
MotionControl ( ), 310
resumeStopJoint
MotionControl ( ), 311
resumeStopLine
MotionControl ( ), 312
returnValue2Str
arcs::common_interface, 608
RobotAlgorithm
arcs::common_interface::RobotAlgorithm, 653
RobotAlgorithm ( ), 349
addCollisionBox, 352
calcJacobian, 353
calibrateTcpForceSensor, 354
calibrateTcpForceSensor2, 355
calibrateTcpForceSensor3, 356
calibVibrationParams, 356
calibVibrationParams1, 356
calibWorkpieceCoordinatePara, 357
forwardDynamics, 357
forwardDynamics1, 358
forwardKinematics, 359
forwardKinematics1, 360
forwardKinematicsAll, 361
forwardToolKinematics, 362
frictionModelIdentify, 363
generatePayloadIdentifyTraj, 364
getPayloadIdentifyResult, 365
getRobotConfiguration, 365
inverseKinematics, 367
inverseKinematics1, 368
inverseKinematics2, 369
inverseKinematicsAll, 370
inverseKinematicsAll1, 371
inverseToolKinematics, 372
inverseToolKinematicsAll, 373
needVibrationRecalib, 373
pathBlend3Points, 374
pathMoveS, 374
payloadCalculateFinished, 375
payloadIdentify, 375
payloadIdentify1, 376
payloadIdentifyTrajGenFinished, 377
removeCollisionObject, 377
validatePath, 378
validatePath1, 379
RobotAlgorithmPtr
arcs::common_interface, 596
RobotConfig
arcs::common_interface::RobotConfig, 658
RobotConfig ( ), 381
attachRobotBaseTo, 384
attachWeldingGun, 384
calcSafetyParametersCheckSum, 385
confirmSafetyParameters, 385

enableEndCollisionCheck, 386
firmwareUpdate, 386
getBaseForceOffset, 387
getBaseForceSensorNames, 388
getCollisionLevel, 389
getCollisionStopType, 389
getCollisionThreshold, 390
getControlBoxType, 391
getCycletime, 392
getDefaultJointAcc, 392
getDefaultJointSpeed, 393
getDefaultToolAcc, 394
getDefaultToolSpeed, 395
getDof, 395
getFirmwareUpdateProcess, 396
getFreedriveDamp, 397
getGravity, 398
getHandguidDamp, 398
getHardwareCustomParameters, 399
getHomePosition, 400
getJointMaxAccelerations, 400
getJointMaxPositions, 401
getJointMaxSpeeds, 402
getJointMinPositions, 402
getKinematicsCompensate, 403
getKinematicsParam, 404
getLimitJointMaxAccelerations, 405
getLimitJointMaxPositions, 406
getLimitJointMaxSpeeds, 407
getLimitJointMinPositions, 407
getLimitTcpMaxSpeed, 408
getMountingPose, 409
getName, 410
getPayload, 410
getRobotEmergencyStopSource, 411
getRobotSubType, 411
getRobotType, 412
getSafeguardStopSource, 413
getSafeguardStopType, 413
getSafetyParametersCheckSum, 414
getSelectedTcpForceSensorName, 414
getSlowDownFraction, 415
getTcpForceOffset, 416
getTcpForceSensorNames, 416
getTcpForceSensorPose, 417
getTcpMaxAccelerations, 417
getTcpMaxSpeeds, 418
getTcpOffset, 419
hasBaseForceSensor, 419
hasTcpForceSensor, 420
isEndCollisionCheckEnabled, 421
selectBaseForceSensor, 422
selectTcpForceSensor, 422
setBaseForceOffset, 423
setCollisionLevel, 424
setCollisionStopType, 425
setCollisionThreshold, 426
setFreedriveDamp, 426
setGravity, 427
setHandguidDamp, 428
setHardwareCustomParameters, 429
setHomePosition, 430
setKinematicsCompensate, 430
setMountingPose, 431
setPayload, 432
setPersistentParameters, 433
setRobotZero, 433
setSlowDownFraction, 434
setTcpForceOffset, 435
setTcpForceSensorPose, 436
setTcpOffset, 436
setToolInertial, 437
setWorkObjectData, 438
toolSpaceInRange, 438
RobotConfigPtr
arcs::common_interface, 597
RobotControlModeType
arcs::common_interface, 604
RobotEmergencyStopType
arcs::common_interface, 604
RobotInterface
arcs::common_interface::RobotInterface, 660
RobotInterface( ), 117
getForceControl, 118
getIoControl, 118
getMotionControl, 119
getRobotAlgorithm, 119
getRobotConfig, 119
getRobotManage, 120
getRobotState, 120
getSyncMove, 120
getTrace, 121
RobotInterfacePtr
arcs::common_interface, 597
RobotManage
arcs::common_interface::RobotManage, 662
RobotManage ( ), 439
backdrive, 441
exitHandguideMode, 442
freedrive, 443
generateDiagnoseFile, 443
getHandguideFeature, 444
getHandguideFreeAxes, 445
getHandguideStatus, 445
getHandguideTrigger, 445
getOperationalMode, 446
getRecordCache, 446
getRobotControlMode, 447
handguideMode, 448
isBackdriveEnabled, 449
isFreedriveEnabled, 449
isHandguideEnabled, 450
isLinkModeEnabled, 451
isSimulationEnabled, 452
lockRobotBrake, 453
pauseRecord, 453

pauseRecordCache, 454
poweroff, 455
poweron, 455
recordCacheFree, 456
releaseRobotBrake, 457
restartInterfaceBoard, 458
setHandguideParams, 459
setLinkModeEnable, 459
setOperationalMode, 460
setSim, 461
setUnlockProtectiveStop, 462
startRecord, 463
startRecordCache, 464
startup, 464
stopRecord, 465
stopRecordCache, 466
RobotManagePtr
arcs::common_interface, 597
RobotModeType
arcs::common_interface, 604
RobotMsgVector
arcs::common_interface, 597
RobotSafetyParameterRange
arcs::common_interface::RobotSafetyParameterRange, 667
RobotState
arcs::common_interface::RobotState, 679
RobotState ( ), 467
getActualTcpOffset, 470
getBaseForce, 470
getControlBoxHumidity, 471
getControlBoxTemperature, 471
getElbowPosistion, 472
getElbowVelocity, 473
getHandleFirmwareVersion, 474
getHandleHardwareVersion, 474
getHandleMode, 475
getHandleState, 476
getHandleUniqueId, 476
getJointAccelerations, 477
getJointContactTorques, 478
getJointCurrents, 479
getJointFirmwareVersions, 479
getJointGravityTorques, 480
getJointHardwareVersions, 481
getJointPositions, 482
getJointPositionsHistory, 482
getJointServoMode, 483
getJointSpeeds, 483
getJointState, 484
getJointTargetAccelerations, 485
getJointTargetCurrents, 485
getJointTargetPositions, 486
getJointTargetSpeeds, 487
getJointTargetTorques, 488
getJointTemperatures, 488
getJointTorqueSensors, 489
getJointUniqueIds, 490
getJointVoltages, 491
getMainCurrent, 491
getMainVoltage, 492
getMasterBoardFirmwareVersion, 493
getMasterBoardHardwareVersion, 494
getMasterBoardUniqueId, 494
getPedestalBrakeResTemp, 495
getPedestalFirmwareVersion, 496
getPedestalHardwareVersion, 497
getPedestalUniqueId, 497
getPowerlossBoardFirmwareVersion, 498
getPowerlossBoardHardwareVersion, 499
getPowerlossBoardUniqueId, 500
getRobotCurrent, 500
getRobotModeType, 501
getRobotVoltage, 502
getSafetyModeType, 503
getSlaveBoardFirmwareVersion, 503
getSlaveBoardHardwareVersion, 504
getSlaveBoardUniqueId, 505
getSlowDownLevel, 506
getTargetTcpPose, 506
getTcpForce, 507
getTcpForceSensors, 507
getTcpForceSensorStatus, 508
getTcpPose, 509
getTcpSpeed, 510
getTcpTargetForce, 510
getTcpTargetPose, 511
getTcpTargetSpeed, 512
getToolCommMode, 513
getToolFirmwareVersion, 513
getToolHardwareVersion, 514
getToolPose, 515
getToolUniqueId, 516
isCollisionOccurred, 516
isPowerOn, 517
isSteady, 518
isTeachPendantEnabled, 518
isToolFlangeEnabled, 519
isWithinSafetyLimits, 520
RobotStatePtr
arcs::common_interface, 597
rpyToQuaternion
Math ( ), 60
rtm_error.h
RTM_ERRORS, 733
RTM_ERRORS
error_stack.h, 720
rtm_error.h, 733
Running
arcs::common_interface, 601
runProgram
RuntimeMachine ( ), 536
RuntimeMachine
arcs::common_interface::RuntimeMachine, 684
RuntimeMachine ( ), 521

abort, 523
arbitraryResume, 523
clearBreakPoints, 524
clearPreloadPrograms, 524
deleteTask, 525
detachTask, 525
enterCritical, 525
exitCritical, 526
getAdvancePlanContext, 526
getAdvancePtr, 527
getExecutionStatus, 527
getInterpPtr, 528
getMainPtr, 528
getPlanContext, 529
getPreloadProgram, 529
getStatus, 530
getTaskQueueSize, 530
getTimer, 531
getTriggInterrupts, 531
gotoLine, 532
isTaskAlive, 532
loadProgram, 533
newTask, 533
nop, 534
pause, 534
preloadProgram, 534
removeBreakPoint, 535
resume, 535
runProgram, 536
setBreakPoint, 536
setLabel, 537
setPlanContext, 537
setResumeWait, 538
start, 539
step, 539
stop, 540
switchTask, 540
timerDelete, 541
timerReset, 541
timerStart, 542
timerStop, 542
triggBegin, 543
triggEnd, 544
triggInterrupt, 544
RuntimeMachinePtr
arcs::common_interface, 597
RuntimeState
arcs::common_interface, 605

SafeguedStopType
arcs::common_interface, 605
SAFETY_CUBIC_NUM
type_def.h, 1193
safety_home
arcs::common_interface::RobotSafetyParameterRange, 670
safety_input_auto_safeguard_reset
arcs::common_interface::RobotSafetyParameterRange, 670
safety_input_auto_safeguard_stop
arcs::common_interface::RobotSafetyParameterRange, 670
safety_input_emergency_stop
arcs::common_interface::RobotSafetyParameterRange, 670
safety_input_handguide
arcs::common_interface::RobotSafetyParameterRange, 671
safety_input_operational_mode
arcs::common_interface::RobotSafetyParameterRange, 671
safety_input_reduced_mode
arcs::common_interface::RobotSafetyParameterRange, 671
safety_input_safeguard_reset
arcs::common_interface::RobotSafetyParameterRange, 671
safety_input_safeguard_stop
arcs::common_interface::RobotSafetyParameterRange, 671
safety_input_three_position_switch
arcs::common_interface::RobotSafetyParameterRange, 671
SAFETY_INTERFACE_BOARD_ERRORS
error_stack.h, 720
hal_error.h, 725
safety_output_emergency_stop
arcs::common_interface::RobotSafetyParameterRange, 672
safety_output_not_emergency_stop
arcs::common_interface::RobotSafetyParameterRange, 672
safety_output_not_reduced_mode
arcs::common_interface::RobotSafetyParameterRange, 672
safety_output_reduced_mode
arcs::common_interface::RobotSafetyParameterRange, 672
safety_output_robot_moving
arcs::common_interface::RobotSafetyParameterRange, 672
safety_output_robot_not_stopping
arcs::common_interface::RobotSafetyParameterRange, 672
safety_output_robot_steady
arcs::common_interface::RobotSafetyParameterRange, 673
safety_output_safe_home
arcs::common_interface::RobotSafetyParameterRange, 673
safety_output_safetyguard_stop
arcs::common_interface::RobotSafetyParameterRange, 673
SAFETY_PARAM_SELECT_NUM
type_def.h, 1193
SAFETY_PLANES_NUM
type_def.h, 1193

SafetyInputAction
arcs::common_interface, 605
SafetyModeType
arcs::common_interface, 605
SafetyOutputRunState
arcs::common_interface, 606
segments
arcs::common_interface::RtdeRecipe, 681
selectBaseForceSensor
RobotConfig ( ), 422
selectTcpForceSensor
RobotConfig ( ), 422
Serial
arcs::common_interface::Serial, 686
Serial ( ), 545
serialClose, 546
serialOpen, 546
serialReadByte, 547
serialReadByteList, 547
serialReadString, 548
serialSendAllString, 549
serialSendByte, 549
serialSendInt, 550
serialSendLine, 551
serialSendString, 551
serialClose
Serial ( ), 546
serialOpen
Serial ( ), 546
SerialPtr
arcs::common_interface, 597
serialReadByte
Serial ( ), 547
serialReadByteList
Serial ( ), 547
serialReadString
Serial ( ), 548
serialSendAllString
Serial ( ), 549
serialSendByte
Serial ( ), 549
serialSendInt
Serial ( ), 550
serialSendLine
Serial ( ), 551
serialSendString
Serial ( ), 551
server_error
arcs::common_interface, 600
servoCartesian
MotionControl ( ), 313
servoCartesianWithAxes
MotionControl ( ), 315
servoCartesianWithAxisGroup
arcs::common_interface::MotionControl, 646
servoJoint
MotionControl ( ), 316
servoJointWithAxes
MotionControl ( ), 317
servoJointWithAxisGroup
arcs::common_interface::MotionControl, 646
setAnalogOutputAfterEStopDefault
IoControl (IO), 214
setBaseForceOffset
RobotConfig ( ), 423
setBool
RegisterControl ( ), 100
setBoolInput
RegisterControl ( ), 101
setBoolOutput
RegisterControl ( ), 102
setBreakPoint
RuntimeMachine ( ), 536
setCirclePathMode
MotionControl ( ), 318
setCollisionLevel
RobotConfig ( ), 424
setCollisionStopType
RobotConfig ( ), 425
setCollisionThreshold
RobotConfig ( ), 426
setCondActive
ForceControl( ), 133
setCondAdvanced
ForceControl( ), 134
setCondCylinder
ForceControl( ), 135
setCondDistance
ForceControl( ), 136
setCondForce
ForceControl( ), 136
setCondOrient
ForceControl( ), 137
setCondPlane
ForceControl( ), 138
setCondSphere
ForceControl( ), 139
setCondTcpSpeed
ForceControl( ), 140
setConfigurableDigitalInputAction
IoControl (IO), 215
setConfigurableDigitalOutput
IoControl (IO), 216
setConfigurableDigitalOutputAfterEStop
IoControl (IO), 217
setConfigurableDigitalOutputPulse
IoControl (IO), 218
setConfigurableDigitalOutputRunstate
IoControl (IO), 219
setConveyorTrackCompensate
MotionControl ( ), 319
setConveyorTrackEncoder
MotionControl ( ), 320
setConveyorTrackLimit
MotionControl ( ), 321
setConveyorTrackSensorOffset

MotionControl ( ), 322
setConveyorTrackStartWindow
MotionControl ( ), 323
setConveyorTrackSyncSeparation
MotionControl ( ), 324
setConveyorTrackVelThreshold
MotionControl ( ), 325
setDamping
ForceControl( ), 141
setDigitalInputActionDefault
IoControl (IO), 220
setDigitalOutputAfterEStopDefault
IoControl (IO), 221
setDigitalOutputRunstate
IoControl (IO), 222
setDigitalOutputRunstateDefault
IoControl (IO), 222
setDouble
RegisterControl ( ), 102
setDoubleInput
RegisterControl ( ), 103
setDoubleOutput
RegisterControl ( ), 104
setDynamicModel
ForceControl( ), 142
setDynamicModel1
ForceControl( ), 143
setDynamicModelContact
ForceControl( ), 144
setDynamicModelInsert
ForceControl( ), 145
setDynamicModelSearch
ForceControl( ), 146
setEncDecoderType
IoControl (IO), 223
setEncTickCount
IoControl (IO), 224
setEndPath
MotionControl ( ), 326
setEqradius
MotionControl ( ), 327
setExtAxisAcceleration
AxisInterface ( ), 38
setExtAxisMaxAccelerationLimit
AxisInterface ( ), 38
setExtAxisMaxPositionLimit
AxisInterface ( ), 39
setExtAxisMaxVelocityLimit
AxisInterface ( ), 39
setExtAxisMountingPose
AxisInterface ( ), 39
setExtAxisType
AxisInterface ( ), 40
setExtAxisVelocity
AxisInterface ( ), 40
setExtAxisZero
AxisInterface ( ), 40
setExtMinPositionLimit
AxisInterface ( ), 40
setFloat
RegisterControl ( ), 105
setFloatInput
RegisterControl ( ), 106
setFloatOutput
RegisterControl ( ), 107
setFreedriveDamp
RobotConfig ( ), 426
setFuturePointSamplePeriod
MotionControl ( ), 328
setGravity
RobotConfig ( ), 427
setGroupOutput
IoControl (IO), 224
setHandguidDamp
RobotConfig ( ), 428
setHandguideParams
RobotManage ( ), 459
setHardwareCustomParameters
RobotConfig ( ), 429
setHomePosition
RobotConfig ( ), 430
setInt16Register
RegisterControl ( ), 107
setInt16RegisterBit
RegisterControl ( ), 108
setInt32
RegisterControl ( ), 109
setInt32Input
RegisterControl ( ), 110
setInt32Output
RegisterControl ( ), 111
setKinematicsCompensate
RobotConfig ( ), 430
setLabel
RuntimeMachine ( ), 537
setLinkModeEnable
RobotManage ( ), 459
setLookAheadSize
MotionControl ( ), 329
setLpFilter
ForceControl( ), 146
setMountingPose
RobotConfig ( ), 431
setOperationalMode
RobotManage ( ), 460
setPayload
RobotConfig ( ), 432
setPersistentParameters
RobotConfig ( ), 433
setPlanContext
RuntimeMachine ( ), 537
setReductionRatio
AxisInterface ( ), 41
setResumeStartPoint
MotionControl ( ), 329
setResumeWait

RuntimeMachine ( ), 538
setRobotZero
RobotConfig ( ), 433
setServoMode
MotionControl ( ), 330
setServoModeSelect
MotionControl ( ), 331
setSim
RobotManage ( ), 461
setSlowDownFraction
RobotConfig ( ), 434
setSoftFloatParams
ForceControl( ), 147
setSpeedFraction
MotionControl ( ), 332
setStandardAnalogInputDomain
IoControl (IO), 225
setStandardAnalogOutput
IoControl (IO), 226
setStandardAnalogOutputAfterEStop
IoControl (IO), 227
setStandardAnalogOutputDomain
IoControl (IO), 228
setStandardAnalogOutputRunstate
IoControl (IO), 229
setStandardDigitalInputAction
IoControl (IO), 230
setStandardDigitalOutput
IoControl (IO), 231
setStandardDigitalOutputAfterEStop
IoControl (IO), 232
setStandardDigitalOutputPulse
IoControl (IO), 233
setStandardDigitalOutputRunstate
IoControl (IO), 234
setString
RegisterControl ( ), 111
setSupvForce
ForceControl( ), 148
setSupvOrient
ForceControl( ), 149
setSupvPosBox
ForceControl( ), 150
setSupvPosCylinder
ForceControl( ), 150
setSupvPosSphere
ForceControl( ), 151
setSupvReoriSpeed
ForceControl( ), 152
setSupvTcpSpeed
ForceControl( ), 153
setTargetForce
ForceControl( ), 154
setTcpForceOffset
RobotConfig ( ), 435
setTcpForceSensorPose
RobotConfig ( ), 436
setTcpMaxLinearVelocity
MotionControl ( ), 333
setTcpOffset
RobotConfig ( ), 436
setTimeOptimalEnable
MotionControl ( ), 334
setToolAnalogInputDomain
IoControl (IO), 235
setToolAnalogOutput
IoControl (IO), 236
setToolAnalogOutputDomain
IoControl (IO), 237
setToolAnalogOutputRunstate
IoControl (IO), 238
setToolDigitalInputAction
IoControl (IO), 239
setToolDigitalOutput
IoControl (IO), 240
setToolDigitalOutputPulse
IoControl (IO), 241
setToolDigitalOutputRunstate
IoControl (IO), 242
setToolInertial
RobotConfig ( ), 437
setToolIoInput
IoControl (IO), 243
setToolVoltageOutputDomain
IoControl (IO), 244
setUnlockProtectiveStop
RobotManage ( ), 462
setVecChar
RegisterControl ( ), 112
setVecDouble
RegisterControl ( ), 113
setVecFloat
RegisterControl ( ), 114
setVecInt32
RegisterControl ( ), 115
setWatchDog
RegisterControl ( ), 115
setWorkObjectData
RobotConfig ( ), 438
setWorkObjectHold
MotionControl ( ), 334
shape
arcs::common_interface::Enveloping, 622
size
arcs::common_interface::RobotSafetyParameterRange, 673
Socket
arcs::common_interface::Socket, 687
Socket (socket), 552
socketClose, 553
socketHasConnected, 554
socketOpen, 554
socketReadAllString, 555
socketReadAsciiFloat, 556
socketReadBinaryInteger, 556
socketReadByteList, 557

socketReadString, 558
socketSendAllString, 559
socketSendByte, 560
socketSendInt, 560
socketSendLine, 561
socketSendString, 562
socketClose
Socket (socket), 553
socketHasConnected
Socket (socket), 554
socketOpen
Socket (socket), 554
SocketPtr
arcs::common_interface, 597
socketReadAllString
Socket (socket), 555
socketReadAsciiFloat
Socket (socket), 556
socketReadBinaryInteger
Socket (socket), 556
socketReadByteList
Socket (socket), 557
socketReadString
Socket (socket), 558
socketSendAllString
Socket (socket), 559
socketSendByte
Socket (socket), 560
socketSendInt
Socket (socket), 560
socketSendLine
Socket (socket), 561
socketSendString
Socket (socket), 562
softFloatDisable
ForceControl( ), 155
softFloatEnable
ForceControl( ), 155
source
arcs::common_interface::RobotMsg, 664
speedChangeDisable
ForceControl( ), 156
speedChangeEnable
ForceControl( ), 156
speedChangeTune
ForceControl( ), 157
speedExtJoint
AxisInterface ( ), 41
speedFractionCritical
MotionControl ( ), 335
speedFractionCritical1
MotionControl ( ), 336
speedJoint
MotionControl ( ), 337
speedLine
MotionControl ( ), 338
Sphere
arcs::common_interface, 598
spiral
arcs::common_interface::CircleParameters, 620
arcs::common_interface::SpiralParameters, 689
StandardInputAction
arcs::common_interface, 606
StandardOutputRunState
arcs::common_interface, 606
start
RuntimeMachine ( ), 539
Starting
arcs::common_interface, 601
startMove
MotionControl ( ), 340
startRecord
RobotManage ( ), 463
startRecordCache
RobotManage ( ), 464
startup
RobotManage ( ), 464
status_code
arcs::common_interface::GripperStatus, 631
step
RuntimeMachine ( ), 539
stiff_param
arcs::common_interface::VibrationRecalibrationParameter, 698
stiff_section
arcs::common_interface::VibrationRecalibrationParameter, 698
stl_path
arcs::common_interface::Enveloping, 622
stop
RuntimeMachine ( ), 540
stop_distance
arcs::common_interface::RobotSafetyParameterRange, 673
stop_time
arcs::common_interface::RobotSafetyParameterRange, 673
stopExtJoint
AxisInterface ( ), 41
stopFollowAnotherAxis
AxisInterface ( ), 42
stopJoint
MotionControl ( ), 340
stopLine
MotionControl ( ), 341
stopMove
MotionControl ( ), 342
Stopped
arcs::common_interface, 601
stopRecord
RobotManage ( ), 465
stopRecordCache
RobotManage ( ), 466
storePath

MotionControl ( ), 343
str2ErrorCode
arcs::error_stack, 609
Stropping
arcs::common_interface, 601
switchTask
RuntimeMachine ( ), 540
SyncMove
arcs::common_interface::SyncMove, 691
SyncMove ( ), 563
axisGroupAdd, 565
axisGroupAddAxis, 565
axisGroupDelete, 566
axisGroupGetActualPositions, 566
axisGroupGetAxisIndex, 567
axisGroupGetAxisName, 567
axisGroupGetTargetPositions, 567
axisGroupMoveJoint, 568
axisGroupOffsetPositions, 568
axisGroupSpeedJoint, 569
axisGroupUpdateAxis, 569
frameAdd, 570
frameAttach, 570
frameConvertPose, 571
frameDelete, 571
frameDeleteAll, 572
frameExist, 572
frameGetChildren, 572
frameGetParent, 573
frameGetPose, 573
frameMove, 574
isSyncMoveOn, 574
syncMoveOff, 575
syncMoveOn, 575
syncMoveResume, 576
syncMoveSegment, 576
syncMoveSuspend, 577
syncMoveUndo, 577
waitSyncTasks, 577
syncMoveOff
SyncMove ( ), 575
syncMoveOn
SyncMove ( ), 575
SyncMovePtr
arcs::common_interface, 598
syncMoveResume
SyncMove ( ), 576
syncMoveSegment
SyncMove ( ), 576
syncMoveSuspend
SyncMove ( ), 577
syncMoveUndo
SyncMove ( ), 577
system_error.h
SYSTEM_ERRORS, 738
SYSTEM_ERRORS
error_stack.h, 721
system_error.h, 738
SystemInfo
arcs::common_interface::SystemInfo, 692
SystemInfo ( ), 578
getControlSoftwareBuildDate, 579
getControlSoftwareFullVersion, 579
getControlSoftwareVersionCode, 580
getControlSoftwareVersionHash, 580
getControlSystemTime, 581
getInterfaceVersionCode, 581
SystemInfoPtr
arcs::common_interface, 598

TaskFrameType
arcs::common_interface, 606
TaskSet
arcs::common_interface, 598
tcp_force
arcs::common_interface::RobotSafetyParameterRange, 674
tcp_speed
arcs::common_interface::RobotSafetyParameterRange, 674
tcpOffsetIdentify
Math ( ), 61
temperature
arcs::common_interface::GripperStatus, 631
textmsg
Trace ( ), 585
timerDelete
RuntimeMachine ( ), 541
timerReset
RuntimeMachine ( ), 541
timerStart
RuntimeMachine ( ), 542
timerStop
RuntimeMachine ( ), 542
timestamp
arcs::common_interface::RobotMsg, 664
to_server
arcs::common_interface::RtdeRecipe, 681
Tool
arcs::common_interface, 604
tool_azimuth
arcs::common_interface::RobotSafetyParameterRange, 674
TOOL_CONFIGURATION_NUM
type_def.h, 1193
tool_deviation
arcs::common_interface::RobotSafetyParameterRange, 674
TOOL_ERRORS
error_stack.h, 721
hal_error.h, 726
tool_inclination
arcs::common_interface::RobotSafetyParameterRange, 674
tool_orientation
arcs::common_interface::RobotSafetyParameterRange, 674

toolContact
ForceControl( ), 158
tools
arcs::common_interface::RobotSafetyParameterRange, 675
toolSpaceInRange
RobotConfig ( ), 438
torque
arcs::common_interface::GripperStatus, 631
tp_3pe_for_handguide
arcs::common_interface::RobotSafetyParameterRange, 675
Trace
arcs::common_interface::Trace, 694
Trace ( ), 582
alarm, 583
peek, 583
popup, 584
textmsg, 585
TraceLevel
arcs::common_interface, 607
TracePtr
arcs::common_interface, 598
trackCartesian
MotionControl ( ), 344
trackJoint
MotionControl ( ), 344
transferRefFrame
Math ( ), 61
triggBegin
RuntimeMachine ( ), 543
triggEnd
RuntimeMachine ( ), 544
trigger
arcs::common_interface::RtdeRecipe, 682
trigger_planes
arcs::common_interface::RobotSafetyParameterRange, 675
triggInterrupt
RuntimeMachine ( ), 544
type
arcs::common_interface::AuboException, 615
type_def.h
CARTESIAN_DOF, 1186
DECL_TO_STRING_FUNC, 1186
ENUM_AuboErrorCodes_DECLARES, 1186
ENUM_AxisModeType_DECLARES, 1186
ENUM_EnvelopingShape_DECLARES, 1186
ENUM_HandleModeType_DECLARES, 1187
ENUM_HandleStateType_DECLARES, 1187
ENUM_ITEM, 1187, 1188
ENUM_JointServoModeType_DECLARES, 1188
ENUM_JointStateType_DECLARES, 1188
ENUM_OperationalModeType_DECLARES, 1189
ENUM_PayloadIdentifyMoveAxis_DECLARES, 1189
ENUM_RobotControlModeType_DECLARES, 1189
ENUM_RobotEmergencyStopType_DECLARES, 1189
ENUM_RobotModeType_DECLARES, 1190
ENUM_RuntimeState_DECLARES, 1190
ENUM_SafeguedStopType_DECLARES, 1190
ENUM_SafetyInputAction_DECLARES, 1190
ENUM_SafetyModeType_DECLARES, 1191
ENUM_SafetyOutputRunState_DECLARES, 1191
ENUM_StandardInputAction_DECLARES, 1191
ENUM_StandardOutputRunState_DECLARES, 1192
ENUM_TaskFrameType_DECLARES, 1192
ENUM_TraceLevel_DECLARES, 1192
SAFETY_CUBIC_NUM, 1193
SAFETY_PARAM_SELECT_NUM, 1193
SAFETY_PLANES_NUM, 1193
TOOL_CONFIGURATION_NUM, 1193

unwindEncDeltaTickCount
IoControl (IO), 245
upper_joint_bound
arcs::common_interface::TrajConfig, 696
user_coord
arcs::common_interface::WObjectData, 700

v
arcs::common_interface::CircleParameters, 620
validatePath
RobotAlgorithm ( ), 378
validatePath1
RobotAlgorithm ( ), 379
variableUpdated
RegisterControl ( ), 116
Vector3d
arcs::common_interface, 598
Vector3f
arcs::common_interface, 598
Vector4d
arcs::common_interface, 598
Vector4f
arcs::common_interface, 599
Vector6f
arcs::common_interface, 599
velocity
arcs::common_interface::GripperStatus, 631
voltage
arcs::common_interface::GripperStatus, 631

waitSyncTasks
SyncMove ( ), 577

weaveEnd
MotionControl ( ), 345
weaveStart
MotionControl ( ), 346
weaveUpdateParameters
MotionControl ( ), 348
what
arcs::common_interface::AuboException, 615