AUBO SDK 接口手册

Version 1.0.0

目录

1. RpcClient 类 1
   1.1 setLogHandler() . . . 1
   1.2 connect() . . . 2
   1.3 disconnect() . . . 2
   1.4 hasConnected() . . . 2
   1.5 login() . . . 3
   1.6 logout() . . . 3
   1.7 hasLogined() . . . 4
   1.8 setRequestTimeout() . . . 4

2. RtdeClient 类 5
   2.1 setLogHandler() . . . 5
   2.2 connect() . . . 6
   2.3 hasConnected() . . . 6
   2.4 login() . . . 6
   2.5 hasLogined() . . . 7
   2.6 logout() . . . 7
   2.7 disconnect() . . . 8
   2.8 getProtocolVersion() . . . 8
   2.9 getInputMaps() . . . 8
   2.10 getOutputMaps() . . . 9
   2.11 setTopic() . . . 9
   2.12 removeTopic() . . . 10
   2.13 getRegisteredInputRecipe() . . . 10
   2.14 getRegisteredOutputRecipe() . . . 10
   2.15 subscribe() . . . 11
   2.16 publish() . . . 11

3. RtdeRecipe 类 13
   3.1 arcs::common_interface::RtdeRecipe 结构体参考 . . . 13

4. ScriptClient 类 15
   4.1 setLogHandler() . . . 15

4.2 connect() . . . 16
4.3 hasConnected() . . . 16
4.4 login() . . . 16
4.5 hasLogined() . . . 17
4.6 logout() . . . 17
4.7 disconnect() . . . 18
4.8 sendFile() . . . 18
4.9 sendString() . . . 18

5. ScriptWriter **类** 21
5.1 arcs::aubo_sdk::ScriptWriter 类参考 . . . 21

6. AuboApi **类** 23
6.1 getMath() . . . 23
6.2 getSystemInfo() . . . 23
6.3 getRuntimeMachine() . . . 24
6.4 getRegisterControl() . . . 24
6.5 getRobotNames() . . . 25
6.6 getRobotInterface() . . . 25
6.7 getSocket() . . . 26
6.8 getSerial() . . . 26

7. AuboException **类** 27
7.1 arcs::common_interface::AuboException 类参考 . . . 27

8. CircleParameters **结构体** 29
8.1 arcs::common_interface::CircleParameters 结构体参考 . . . 29

9. Math **类** 31
9.1 poseAdd() . . . 31
9.2 poseSub() . . . 31
9.3 poseTrans() . . . 32
9.4 poseInverse() . . . 33
9.5 rpyToQuaternion() . . . 34
9.6 quaternionToRpy() . . . 34
9.7 tcpOffsetIdentify() . . . 35

9.8 calibrateCoordinate() 35
9.9 calculateCircleFourthPoint() 36

10. RegisterControl 类 37
10.1 getBoolInput() 37
10.2 setBoolInput() 37
10.3 getInt32Input() 38
10.4 setInt32Input() 39
10.5 getFloatInput() 40
10.6 setFloatInput() 40
10.7 getDoubleInput() 41
10.8 setDoubleInput() 42
10.9 getBoolOutput() 42
10.10setBoolOutput() 43
10.11getInt32Output() 44
10.12setInt32Output() 44
10.13getFloatOutput() 45
10.14setFloatOutput() 46
10.15getDoubleOutput() 46
10.16setDoubleOutput() 47
10.17getInt16Register() 48
10.18setInt16Register() 49
10.19hasNamedVariable() 49
10.20getNamedVariableType() 50
10.21clearNamedVariable() 50
10.22modbusAddSignal() 51
10.23modbusDeleteSignal() 52
10.24modbusDeleteAllSignals() 52
10.25modbusGetSignalStatus() 53
10.26modbusGetSignalNames() 53
10.27modbusGetSignalTypes() 54
10.28modbusGetSignalValues() 54
10.29modbusGetSignalErrors() 54
10.30modbusSendCustomCommand() 55

10.31modbusSetDigitalInputAction() 56
10.32modbusSetOutputSignal() 56
10.33modbusSetSignalUpdateFrequency() 57

11. RobotInterface 类 59
11.1 getRobotConfig() 59
11.2 getMotionControl() 59
11.3 getForceControl() 60
11.4 getIoControl() 60
11.5 getSyncMove() 61
11.6 getRobotAlgorithm() 61
11.7 getRobotManage() 62
11.8 getRobotState() 62
11.9 getTrace() 63

12. RuntimeMachine 类 65
12.1 newTask() 65
12.2 deleteTask() 65
12.3 setPlanContext() 65
12.4 nop() 66
12.5 getExecutionStatus() 66
12.6 gotoLine() 66
12.7 getPlanContext() 67
12.8 getAdvancePlanContext() 68
12.9 getAdvancePtr() 68
12.10getMainPtr() 68
12.11loadProgram() 69
12.12runProgram() 69
12.13start() 70
12.14stop() 70
12.15abort() 71
12.16pause() 71
12.17step() 72
12.18resume() 72

12.19getStatus() . . . 73
12.20setBreakPoint() . . . 73
12.21removeBreakPoint() . . . 74
12.22clearBreakPoints() . . . 74
12.23timerStart() . . . 75
12.24timerStop() . . . 75
12.25timerReset() . . . 76
12.26timerDelete() . . . 76
12.27getTimer() . . . 77

13. SystemInfo 类 79
13.1 getControlSoftwareVersionCode() . . . 79
13.2 getInterfaceVersionCode() . . . 79
13.3 getControlSoftwareBuildDate() . . . 80
13.4 getControlSoftwareVersionHash() . . . 80
13.5 getControlSystemTime() . . . 81

14. Trace 类 83
14.1 alarm() . . . 83
14.2 textmsg() . . . 84
14.3 popup() . . . 84
14.4 peek() . . . 85

15. ForceControl 类 87
15.1 fcEnable() . . . 87
15.2 fcDisable() . . . 87
15.3 isFcEnabled() . . . 88
15.4 setTargetForce() . . . 88
15.5 setDynamicModel() . . . 89
15.6 setLpFilter() . . . 90

16. InputParser 类 91
16.1 arcs::aubo_sdk::InputParser 类参考 . . . 91

17. IoControl 类 93
17.1 getStandardDigitalInputNum() . . . 93

17.2 getToolDigitalInputNum() . . . 93
17.3 getConfigurableDigitalInputNum() . . . 94
17.4 getStandardDigitalOutputNum() . . . 94
17.5 getToolDigitalOutputNum() . . . 95
17.6 setToolIoInput() . . . 95
17.7 isToolIoInput() . . . 96
17.8 getConfigurableDigitalOutputNum() . . . 96
17.9 getStandardAnalogInputNum() . . . 97
17.10getToolAnalogInputNum() . . . 97
17.11getStandardAnalogOutputNum() . . . 98
17.12getToolAnalogOutputNum() . . . 98
17.13setDigitalInputActionDefault() . . . 98
17.14setStandardDigitalInputAction() . . . 99
17.15setToolDigitalInputAction() . . . 99
17.16setConfigurableDigitalInputAction() . . . 100
17.17getStandardDigitalInputAction() . . . 101
17.18getToolDigitalInputAction() . . . 101
17.19getConfigurableDigitalInputAction() . . . 102
17.20setDigitalOutputRunstateDefault() . . . 103
17.21setStandardDigitalOutputRunstate() . . . 103
17.22setToolDigitalOutputRunstate() . . . 104
17.23setConfigurableDigitalOutputRunstate() . . . 104
17.24getStandardDigitalOutputRunstate() . . . 105
17.25getToolDigitalOutputRunstate() . . . 106
17.26getConfigurableDigitalOutputRunstate() . . . 106
17.27setStandardAnalogOutputRunstate() . . . 107
17.28setToolAnalogOutputRunstate() . . . 108
17.29getStandardAnalogOutputRunstate() . . . 108
17.30getToolAnalogOutputRunstate() . . . 109
17.31setStandardAnalogInputDomain() . . . 110
17.32setToolAnalogInputDomain() . . . 110
17.33getStandardAnalogInputDomain() . . . 111
17.34getToolAnalogInputDomain() . . . 111

17.35setStandardAnalogOutputDomain() 112
17.36setToolAnalogOutputDomain() 113
17.37getStandardAnalogOutputDomain() 113
17.38getToolAnalogOutputDomain() 114
17.39setToolVoltageOutputDomain() 115
17.40getToolVoltageOutputDomain() 115
17.41setStandardDigitalOutput() 115
17.42setToolDigitalOutput() 116
17.43setConfigurableDigitalOutput() 117
17.44setStandardAnalogOutput() 117
17.45setToolAnalogOutput() 118
17.46getStandardDigitalInput() 118
17.47getStandardDigitalInputs() 119
17.48getToolDigitalInput() 120
17.49getToolDigitalInputs() 120
17.50getConfigurableDigitalInput() 121
17.51getConfigurableDigitalInputs() 121
17.52getStandardDigitalOutput() 122
17.53getStandardDigitalOutputs() 122
17.54getToolDigitalOutput() 123
17.55getToolDigitalOutputs() 123
17.56getConfigurableDigitalOutput() 124
17.57getConfigurableDigitalOutputs() 124
17.58getStandardAnalogInput() 125
17.59getToolAnalogInput() 125
17.60getStandardAnalogOutput() 126
17.61getToolAnalogOutput() 126
17.62getStaticLinkInputNum() 127
17.63getStaticLinkOutputNum() 128
17.64getStaticLinkInputs() 128
17.65getStaticLinkOutputs() 129

18. MotionControl **类** 131

18.1 getEqradius() 131

18.2 setSpeedFraction() . . . 131
18.3 getSpeedFraction() . . . 132
18.4 pathOffsetEnable() . . . 132
18.5 pathOffsetSet() . . . 132
18.6 pathOffsetDisable() . . . 133
18.7 jointOffsetEnable() . . . 134
18.8 jointOffsetSet() . . . 134
18.9 jointOffsetDisable() . . . 135
18.10getQueueSize() . . . 135
18.11getTrajectoryQueueSize() . . . 136
18.12getExecId() . . . 136
18.13getDuration() . . . 137
18.14getMotionLeftTime() . . . 137
18.15getProgress() . . . 138
18.16setWorkObjectHold() . . . 138
18.17getWorkObjectHold() . . . 139
18.18getPauseJointPositions() . . . 140
18.19setServoMode() . . . 140
18.20isServoModeEnabled() . . . 141
18.21servoJoint() . . . 141
18.22followJoint() . . . 142
18.23followLine() . . . 142
18.24speedJoint() . . . 143
18.25resumeSpeedJoint() . . . 143
18.26speedLine() . . . 144
18.27resumeSpeedLine() . . . 145
18.28moveSpline() . . . 146
18.29moveJoint() . . . 147
18.30resumeMoveJoint() . . . 148
18.31moveLine() . . . 148
18.32resumeMoveLine() . . . 149
18.33moveCircle() . . . 150
18.34setCirclePathMode() . . . 151

18.35moveCircle2() . . . 152
18.36pathBufferAlloc() . . . 152
18.37pathBufferAppend() . . . 153
18.38pathBufferEval() . . . 154
18.39pathBufferValid() . . . 155
18.40pathBufferFree() . . . 155
18.41pathBufferList() . . . 156
18.42movePathBuffer() . . . 156
18.43stopJoint() . . . 157
18.44resumeStopJoint() . . . 158
18.45stopLine() . . . 158
18.46resumeStopLine() . . . 159
18.47weaveStart() . . . 159
18.48weaveEnd() . . . 160

19. OutputBuilder 类 161
19.1 arcs::aubo_sdk::OutputBuilder 类参考 . . . 161

20. RobotAlgorithm 类 163
20.1 calibrateTcpForceSensor() . . . 163
20.2 payloadIdentify() . . . 163
20.3 calibWorkpieceCoordinatePara() . . . 164
20.4 forwardDynamics() . . . 165
20.5 forwardKinematics() . . . 166
20.6 forwardToolKinematics() . . . 166
20.7 inverseKinematics() . . . 167
20.8 inverseKinematicsAll() . . . 167
20.9 inverseToolKinematics() . . . 168
20.10inverseToolKinematicsAll() . . . 168
20.11pathMovej() . . . 169
20.12pathBlend3Points() . . . 170

21. RobotConfig 类 171
21.1 getName() . . . 171
21.2 getDof() . . . 171

21.3 getCycletime() . . . 172
21.4 setSlowDownFraction() . . . 172
21.5 getSlowDownFraction() . . . 172
21.6 getDefaultToolAcc() . . . 173
21.7 getDefaultToolSpeed() . . . 173
21.8 getDefaultJointAcc() . . . 174
21.9 getDefaultJointSpeed() . . . 174
21.10getRobotType() . . . 175
21.11getRobotSubType() . . . 175
21.12getControlBoxType() . . . 176
21.13setMountingPose() . . . 176
21.14getMountingPose() . . . 177
21.15setCollisionLevel() . . . 177
21.16getCollisionLevel() . . . 178
21.17setCollisionStopType() . . . 178
21.18getCollisionStopType() . . . 179
21.19setHomePosition() . . . 179
21.20getHomePosition() . . . 179
21.21setFreedriveDamp() . . . 180
21.22getFreedriveDamp() . . . 180
21.23getKinematicsParam() . . . 181
21.24getKinematicsCompensate() . . . 181
21.25setKinematicsCompensate() . . . 182
21.26setPersistentParameters() . . . 182
21.27setHardwareCustomParameters() . . . 183
21.28getHardwareCustomParameters() . . . 183
21.29setRobotZero() . . . 184
21.30getTcpForceSensorNames() . . . 184
21.31selectTcpForceSensor() . . . 185
21.32setTcpForceSensorPose() . . . 185
21.33getTcpForceSensorPose() . . . 185
21.34hasTcpForceSensor() . . . 186
21.35setTcpForceOffset() . . . 186

21.36getTcpForceOffset() . . . 187
21.37getBaseForceSensorNames() . . . 187
21.38selectBaseForceSensor() . . . 188
21.39hasBaseForceSensor() . . . 188
21.40setBaseForceOffset() . . . 188
21.41getBaseForceOffset() . . . 189
21.42getSafetyParametersCheckSum() . . . 190
21.43confirmSafetyParameters() . . . 190
21.44getJointMaxPositions() . . . 191
21.45getJointMinPositions() . . . 191
21.46getJointMaxSpeeds() . . . 192
21.47getJointMaxAccelerations() . . . 192
21.48getTcpMaxSpeeds() . . . 193
21.49getTcpMaxAccelerations() . . . 193
21.50setGravity() . . . 193
21.51getGravity() . . . 194
21.52setTcpOffset() . . . 194
21.53getTcpOffset() . . . 195
21.54setToolInertial() . . . 195
21.55setPayload() . . . 196
21.56getPayload() . . . 197
21.57toolSpaceInRange() . . . 197
21.58firmwareUpdate() . . . 198
21.59getFirmwareUpdateProcess() . . . 199

22. RobotManage 类 201
22.1 poweron() . . . 201
22.2 startup() . . . 201
22.3 poweroff() . . . 202
22.4 backdrive() . . . 202
22.5 freedrive() . . . 203
22.6 setSim() . . . 203
22.7 setOperationalMode() . . . 204
22.8 getOperationalMode() . . . 205

22.9 getRobotControlMode() . . . 205
22.10isFreedriveEnabled() . . . 206
22.11isBackdriveEnabled() . . . 207
22.12isSimulationEnabled() . . . 207
22.13setUnlockProtectiveStop() . . . 208
22.14restartInterfaceBoard() . . . 208

23. RobotMsg **结构体** 209
23.1 arcs::common_interface::RobotMsg 结构体参考 . . . 209

24. RobotSafetyParameterRang **结构体** 211
24.1 arcs::common_interface::RobotSafetyParameterRange 结构体参考 . . . 211

25. RobotState **类** 213
25.1 getRobotModeType() . . . 213
25.2 getSafetyModeType() . . . 213
25.3 isPowerOn() . . . 214
25.4 isSteady() . . . 214
25.5 isCollisionOccurred() . . . 214
25.6 isWithinSafetyLimits() . . . 215
25.7 getTcpPose() . . . 215
25.8 getTargetTcpPose() . . . 216
25.9 getToolPose() . . . 216
25.10getTcpSpeed() . . . 217
25.11getTcpForce() . . . 217
25.12getElbowPosistion() . . . 218
25.13getElbowVelocity() . . . 218
25.14getBaseForce() . . . 219
25.15getTcpTargetPose() . . . 219
25.16getTcpTargetSpeed() . . . 220
25.17getTcpTargetForce() . . . 220
25.18getJointState() . . . 221
25.19getJointServoMode() . . . 221
25.20getJointPositions() . . . 222
25.21getJointSpeeds() . . . 222

25.22getJointAccelerations() 223
25.23getJointTorqueSensors() 223
25.24getBaseForceSensor() 224
25.25getTcpForceSensors() 224
25.26getJointCurrents() 225
25.27getJointVoltages() 225
25.28getJointTemperatures() 226
25.29getJointUniqueIds() 226
25.30getJointFirmwareVersions() 227
25.31getJointHardwareVersions() 227
25.32getMasterBoardUniqueId() 228
25.33getMasterBoardFirmwareVersion() 228
25.34getMasterBoardHardwareVersion() 229
25.35getSlaveBoardUniqueId() 229
25.36getSlaveBoardFirmwareVersion() 230
25.37getSlaveBoardHardwareVersion() 230
25.38getToolUniqueId() 231
25.39getToolFirmwareVersion() 231
25.40getToolHardwareVersion() 232
25.41getPedestalUniqueId() 232
25.42getPedestalFirmwareVersion() 233
25.43getPedestalHardwareVersion() 233
25.44getJointTargetPositions() 234
25.45getJointTargetSpeeds() 234
25.46getJointTargetAccelerations() 235
25.47getJointTargetTorques() 235
25.48getJointTargetCurrents() 236
25.49getControlBoxTemperature() 236
25.50getMainVoltage() 237
25.51getMainCurrent() 237
25.52getRobotVoltage() 238
25.53getRobotCurrent() 238

1. RpcClient 类

1.1 setLogHandler()

void arcs::aubo_sdk::RpcClient::setLogHandler (
std::function< void(int, const char *, int, const std::string &)> *handler*)

设置日志处理器

此函数可设置自定义的日志处理函数来处理日志消息。
Aubo SDK 有一套默认的日志系统，按照默认的格式输出到默认的文件。如果用户不希望采用默认的格式或者不希望输出到默认的文件，那就可以通过这个接口重新自定义格式，或者输出路径。这个函数可以将用户自定义的日志系统与 Aubo SDK 默认的日志系统合并。

注解

setLogHandler 函数要放在即将触发的日志之前，否则会按照默认的形式输出日志。

参数

handler	日志处理函数 此日志处理函数的下定义如下: void handler(int level, const char* filename, int line, const std::string& message) level 表示日志等级 0: LOGLEVEL_FATAL 严重的错误 1: LOGLEVEL_ERROR 错误 2: LOGLEVEL_WARNING 警告 3: LOGLEVEL_INFO 通知 4: LOGLEVEL_DEBUG 调试 5: LOGLEVEL_BACKTRACE 跟踪 filename 表示文件名 line 表示代码行号 message 表示日志信息

返回

无

1.2 connect()

int arcs::aubo_sdk::RpcClient::connect (
const std::string & *ip*,
int *port*)

连接到RPC 服务

参数

ip	IP 地址
port	端口号，RPC 的端口号是 30004

返回值

0	RPC 连接成功
-8	RPC 连接失败，RPC 连接被拒绝
-15	RPC 连接失败，SDK 版本与Server 版本不兼容

1.3 disconnect()

int arcs::aubo_sdk::RpcClient::disconnect ( )

断开RPC 连接

返回值

0	成功
-1	失败

1.4 hasConnected()

bool arcs::aubo_sdk::RpcClient::hasConnected ( ) const

判断是否连接RPC

返回值

true	已连接RPC
false	未连接RPC

1.5 login()

int arcs::aubo_sdk::RpcClient::login (
const std::string & *usrname,*
const std::string & *passwd*)

登录

参数

usrname	用户名
passwd	密码

返回

0

1.6 logout()

int arcs::aubo_sdk::RpcClient::logout ( )

登出

返回

0

1.7 hasLogined()

bool arcs::aubo_sdk::RpcClient::hasLogined ( )

判断是否登录

返回值

true	已登录
false	未登录

1.8 setRequestTimeout()

int arcs::aubo_sdk::RpcClient::setRequestTimeout (
int *timeout* = 10)

设置RPC 请求超时时间

参数

timeout	请求超时时间，单位 ms

返回

0

2. RtdeClient 类

2.1 setLogHandler()

void arcs::aubo_sdk::RtdeClient::setLogHandler (
std::function< void(int, const char *, int, const std::string &)> *handler*
)

设置日志处理器

此函数可设置自定义的日志处理函数来处理日志消息。
Aubo SDK 有一套默认的日志系统，按照默认的格式输出到默认的文件。如果用户不希望采用默认的格式或者不希望输出到默认的文件，那就可以通过这个接口重新自定义格式，或者输出路径。这个函数可以将用户自定义的日志系统与 Aubo SDK 默认的日志系统合并。

注解

setLogHandler 函数要放在即将触发的日志之前，否则会按照默认的形式输出日志。

参数

handler	日志处理函数 此日志处理函数的下定义如下: void handler(int level, const char* filename, int line, const std::string& message) level 表示日志等级 0: LOGLEVEL_FATAL 严重的错误 1: LOGLEVEL_ERROR 错误 2: LOGLEVEL_WARNING 警告 3: LOGLEVEL_INFO 通知 4: LOGLEVEL_DEBUG 调试 5: LOGLEVEL_BACKTRACE 跟踪 filename 表示文件名 line 表示代码行号 message 表示日志信息

返回

无

2.2 connect()

int arcs::aubo_sdk::RtdeClient::connect (
const std::string & *ip*,
int *port*)

连接到服务器

参数

ip	IP 地址
port	端口号，RTDE 端口号为 30010

返回值

0	连接成功
1	在执行函数前，已连接
-1	连接失败

2.3 hasConnected()

bool arcs::aubo_sdk::RtdeClient::hasConnected ( ) const

socket 是否已连接

返回值

true	已连接 socket
false	未连接 socket

2.4 login()

int arcs::aubo_sdk::RtdeClient::login (

const std::string & *usrname*,
const std::string & *passwd*)

登录

参数

usrname	用户名
passwd	密码

返回值

0	成功
-1	失败

2.5 hasLogined()

bool arcs::aubo_sdk::RtdeClient::hasLogined ( )

是否已经登录

返回值

true	已登录
false	未登录

2.6 logout()

int arcs::aubo_sdk::RtdeClient::logout ( )

登出

返回

0

2.7 disconnect()

int arcs::aubo_sdk::RtdeClient::disconnect ( )

断开连接

返回值

0	成功
-1	失败

2.8 getProtocolVersion()

int arcs::aubo_sdk::RtdeClient::getProtocolVersion ( )

获取协议版本号

返回

协议版本号

2.9 getInputMaps()

std::map< std::string, int > arcs::aubo_sdk::RtdeClient::getInputMaps ( )

获取输入列表

返回

输入列表

2.10 getOutputMaps()

std::map< std::string, int > arcs::aubo_sdk::RtdeClient::getOutputMaps ( )

获取输出列表

返回

输出列表

2.11 setTopic()

int arcs::aubo_sdk::RtdeClient::setTopic (
bool *to_server,*
const std::vector< std::string > & *names,*
double *freq,*
int *expected_chanel*)

设置话题

参数

to_server	数据流向。true 表示客户端给服务器发送消息，false 表示服务器给客户端发送消息
names	服务器推送的信息列表
freq	服务器推送信息的频率
expected_chanel	通道。取值范围：0~99，发布不同的话题走不同的通道

返回值

expected_chanel 参数的值	成功
-1	失败

2.12 removeTopic()

int arcs::aubo_sdk::RtdeClient::removeTopic (
bool *to_server,*
int *chanel*)

取消订阅

参数

to_server	数据流向 true 表示客户端给服务器发送消息，false 表示服务器给客户端发送消息
chanel	通道

返回值

0	成功
1	失败

2.13 getRegisteredInputRecipe()

std::unordered_map< int, common_interface::RtdeRecipe > arcs::aubo_sdk::↩ RtdeClient::getRegisteredInputRecipe ( )

获取已注册的输入菜单

返回

已注册的输入菜单

2.14 getRegisteredOutputRecipe()

std::unordered_map< int, common_interface::RtdeRecipe > arcs::aubo_sdk::↩ RtdeClient::getRegisteredOutputRecipe ( )

获取已注册的输出菜单

返回

已注册的输出菜单

2.15 subscribe()

int arcs::aubo_sdk::RtdeClient::subscribe (
int *chanel*,
std::function< void(InputParser &)> *callback*)

订阅来自服务器的数据

参数

chanel	通道
callback	回调函数，用于处理订阅的输入信息。 回调函数的定义如下：void callback(InputParser &parser)

返回

0

2.16 publish()

int arcs::aubo_sdk::RtdeClient::publish (
int *chanel*,
std::function< void(OutputBuilder &)> *callback*)

发布，表示向服务器推送数据

参数

chanel	通道
callback	回调函数，用于构建发布的输出信息。 回调函数的定义如下：void callback(OutputBuilder &builder)

返回值

0	成功
-1	失败

3. RtdeRecipe 类

3.1 arcs::common_interface::RtdeRecipe 结构体参考

Public **属性**

- bool to_server
- int chanel
- double frequency
- int trigger
- std::vector< std::string > segments

4. ScriptClient 类

4.1 setLogHandler()

void arcs::aubo_sdk::ScriptClient::setLogHandler (
std::function< void(int, const char *, int, const std::string &)> *handler*
)

设置日志处理器

此函数可设置自定义的日志处理函数来处理日志消息。
Aubo SDK 有一套默认的日志系统，按照默认的格式输出到默认的文件。如果用户不希望采用默认的格式或者不希望输出到默认的文件，那就可以通过这个接口重新自定义格式，或者输出路径。这个函数可以将用户自定义的日志系统与 Aubo SDK 默认的日志系统合并。

注解

setLogHandler 函数要放在即将触发的日志之前，否则会按照默认的形式输出日志。

参数

handler	日志处理函数 此日志处理函数的下定义如下: void handler(int level, const char* filename, int line, const std::string& message) level 表示日志等级 0: LOGLEVEL_FATAL 严重的错误 1: LOGLEVEL_ERROR 错误 2: LOGLEVEL_WARNING 警告 3: LOGLEVEL_INFO 通知 4: LOGLEVEL_DEBUG 调试 5: LOGLEVEL_BACKTRACE 跟踪 filename 表示文件名 line 表示代码行号 message 表示日志信息

返回

无

4.2 connect()

int arcs::aubo_sdk::ScriptClient::connect (
const std::string & *ip*,
int *port*)

连接到服务器

参数

ip	IP 地址
port	端口号。SCRIPT 端口号为 30004

返回值

0	连接成功
1	在执行函数前，已连接
-1	连接失败

4.3 hasConnected()

bool arcs::aubo_sdk::ScriptClient::hasConnected ( ) const

是否处于连接状态

返回值

true	已连接
false	未连接

4.4 login()

int arcs::aubo_sdk::ScriptClient::login (
const std::string & *usrname*,
const std::string & *passwd*)

登录

参数

usrname	用户名
passwd	密码

返回值

0	成功
-1	失败

4.5 hasLogined()

bool arcs::aubo_sdk::ScriptClient::hasLogined ( )

返回客户端是否登录

返回值

true	已登录
false	未登录

4.6 logout()

int arcs::aubo_sdk::ScriptClient::logout ( )

登出

返回

0

4.7 disconnect()

int arcs::aubo_sdk::ScriptClient::disconnect ( )

断开连接

返回值

0	成功
-1	失败

4.8 sendFile()

int arcs::aubo_sdk::ScriptClient::sendFile (
const std::string & *path*)

发送脚本文件

参数

path	脚本文件的路径

返回值

0	成功
-1	失败

4.9 sendString()

int arcs::aubo_sdk::ScriptClient::sendString (
const std::string & *script*)

发送脚本内容

调用本地的脚本

参数

script	脚本内容

返回值

0	成功
-1	失败

5. ScriptWriter 类

5.1 arcs::aubo_sdk::ScriptWriter 类参考

Public 成员函数

- virtual ScriptWriter & append (const std::string &line)=0
- virtual ScriptWriter & append (const char *buf, size_t len)=0
- virtual ScriptWriter & moveJoint ()=0
- virtual ScriptWriter & moveLine ()=0
- virtual ScriptWriter & ifCondition ()=0
- virtual ScriptWriter & elseCondition ()=0
- virtual ScriptWriter & elseIfCondition ()=0
- virtual ScriptWriter & whileCondition ()=0
- virtual ScriptWriter & end ()=0

6. AuboApi 类

6.1 getMath()

MathPtr arcs::common_interface::AuboApi::getMath ( )

获取纯数学相关接口

返回

MathPtr 对象的指针

Python 函数原型

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

C++ 示例

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

6.2 getSystemInfo()

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

获取系统信息

返回

SystemInfoPtr 对象的指针

Python 函数原型

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

C++ 示例

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

6.3 getRuntimeMachine()

RuntimeMachinePtr arcs::common_interface::AuboApi::getRuntimeMachine ( )

获取运行时接口

返回

RuntimeMachinePtr 对象的指针

Python 函数原型

getRuntimeMachine(self: pyaubo_sdk.AuboApi) -> pyaubo_sdk.RuntimeMachine

C++ 示例

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

6.4 getRegisterControl()

RegisterControlPtr arcs::common_interface::AuboApi::getRegisterControl ( )

对外寄存器接口

返回

RegisterControlPtr 对象的指针

Python 函数原型

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

C++ 示例

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

6.5 getRobotNames()

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

获取机器人列表

返回

机器人列表

Python 函数原型

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

C++ 示例

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

6.6 getRobotInterface()

RobotInterfacePtr arcs::common_interface::AuboApi::getRobotInterface (const std::string & *name*)

根据名字获取 RobotInterfacePtr 接口

参数

name	机器人名字

返回

RobotInterfacePtr 对象的指针

Python 函数原型

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

C++ 示例

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

6.7 getSocket()

SocketPtr arcs::common_interface::AuboApi::getSocket ( )

获取外部轴接口

获取独立 IO 模块接口

获取 socket

返回

SocketPtr 对象的指针

Python 函数原型

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

C++ 示例

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

6.8 getSerial()

SerialPtr arcs::common_interface::AuboApi::getSerial ( )

返回

SerialPtr 对象的指针

Python 函数原型

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

C++ 示例

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

7. AuboException 类

7.1 arcs::common_interface::AuboException 类参考

Public 成员函数

- 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

8. CircleParameters 结构体

8.1 arcs::common_interface::CircleParameters 结构体参考

Public **属性**

- std::vector< double > pose_via
- std::vector< double > pose_to
- double a
- double v
- double blend_radius
- double duration
- double helix
- double spiral
- double direction
- int loop_times

9. Math 类

9.1 poseAdd()

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

位姿相加。两个参数都包含三个位置参数（x、y、z），统称为P，以及三个旋转参数（R_x、R_y、R_z），统称为R。此函数根据以下方式计算结果 p_3，即给定位姿的相加：p_3.P = p_1.P + p_2.P, p_3.R = p_1.R * p_2.R

参数

p1	工具位姿 1
p2	工具位姿 2

返回

位置部分之和和旋转部分之积（pose）

Python 函数原型

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

Lua 函数原型

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

9.2 poseSub()

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

位姿相减

参数

p1	工具位姿 1
p2	工具位姿 2

返回

Python 函数原型

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

Lua 函数原型

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

9.3 poseTrans()

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

位姿转换

第一个参数 p_from 用于转换第二个参数 p_from_to，并返回结果。这意味着结果是从 p_from 的坐标系开始，然后在该坐标系中移动 p_from_to 后的位姿。

这个函数可以从两个不同的角度来看。一种是函数将 p_from_to 根据 p_from 的参数进行转换，即平移和旋转。另一种是函数被用于获取结果姿态，先对 p_from 进行移动，然后再对 p_from_to 进行移动。如果将姿态视为转换矩阵，它看起来像是:

T_world->to = T_world->from * T_from->to，T_x->to = T_x->from * T_from->to

这两个方程描述了姿态转换的基本原理，根据给定的起始姿态和相对于起始姿态的姿态变化，可以计算出目标姿态。

举个例子，已知B 相对于A 的位姿、C 相对于B 的位姿，求C 相对于A 的位姿。第一个参数是B 相对于A 的位姿，第二个参数是C 相对于B 的位姿，返回值是C 相对于A 的位姿。

参数

pose_from	起始位姿（空间向量）
pose_from_to	相对于起始位姿的姿态变化（空间向量）

返回

结果位姿 (空间向量)

Python 函数原型

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

Lua 函数原型

poseTrans(pose_from: table, pose_from_to: table) -> table

9.4 poseInverse()

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

获取位姿的逆

参数

pose	工具位姿（空间向量）

返回

位姿的逆转换（空间向量）

Python 函数原型

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

Lua 函数原型

poseInverse(pose: table) -> table

9.5 rpyToQuaternion()

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

欧拉角转四元数

参数

rpy	欧拉角

返回

四元数

Python 函数原型

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

Lua 函数原型

rpyToQuaternion(rpy: table) -> table

9.6 quaternionToRpy()

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

四元数转欧拉角

参数

quat	四元数

返回

欧拉角

Python 函数原型

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

Lua 函数原型

quaternionToRpy(quat: table) -> table

9.7 tcpOffsetIdentify()

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

参数

poses	位姿

返回

Python 函数原型

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

Lua 函数原型

tcpOffsetIdentify(poses: table) -> table

9.8 calibrateCoordinate()

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

三点法标定坐标系

参数

poses	位姿
type	类型: 0 - oxy 原点 x 轴正方向 xy 平面（y 轴正方向） 1 - oxz 原点 x 轴正方向 xz 平面（z 轴正方向） 2 - oyz 原点 y 轴正方向 yz 平面（z 轴正方向） 3 - oyx 原点 y 轴正方向 yx 平面（x 轴正方向） 4 - ozx 原点 z 轴正方向 zx 平面（x 轴正方向） 5 - ozy 原点 z 轴正方向 zy 平面（y 轴正方向）

返回

标定的坐标系是否有效

9.9 calculateCircleFourthPoint()

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

根据圆弧的三个点, 计算出拟合成的圆的另一半圆弧的中间点位置

参数

p1	圆弧的起始点
p2	圆弧的中间点
p3	圆弧的结束点

返回

拟合成的圆的另一半圆弧的中间点位置和计算结果是否有效

10. RegisterControl 类

10.1 getBoolInput()

bool arcs::common_interface::RegisterControl::getBoolInput (
uint32_t *address*)

从一个输入寄存器中读取布尔值，也可以通过现场总线进行访问。注意，它使用自己的内存空间。

参数

address	寄存器的地址（0:127）

返回

寄存器中保存的布尔值（true、false）

注解

布尔输入寄存器的较低范围[0:63]保留供FieldBus/PLC 接口使用。较高范围[64:127]无法通过FieldBus/PLC 接口访问，因为它保留供外部RTDE 客户端使用。

Python 函数原型

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

Lua 函数原型

getBoolInput(address: number) -> boolean

10.2 setBoolInput()

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

写入输入寄存器的布尔值

参数

address	寄存器的地址
value	布尔值

返回

注解

只有在实现 RTDE/Modbus Slave/PLC 服务端时使用

Python 函数原型

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

Lua 函数原型

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

10.3 getInt32Input()

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

从一个输入寄存器中读取整数值，也可以通过现场总线进行访问。注意，它使用自己的内存空间。

参数

address	寄存器的地址（0:47）

返回

寄存器中保存的整数值[-2,147,483,648 : 2,147,483,647]

注解

整数输入寄存器的较低范围[0:23]保留供FieldBus/PLC 接口使用。较高范围[24:47]无法通过FieldBus/PLC 接口访问，因为它保留供外部RTDE 客户端使用。

Python 函数原型

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

Lua 函数原型

getInt32Input(address: number) -> number

10.4 setInt32Input()

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

写入输入寄存器的整数值

参数

address	寄存器的地址
value	整数值

返回

注解

只有在实现 RTDE/Modbus Slave/PLC 服务端时使用

Python 函数原型

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

Lua 函数原型

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

10.5 getFloatInput()

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

从一个输入寄存器中读取浮点数，也可以通过现场总线进行访问。注意，它使用自己的内存空间。

参数

address	寄存器地址（0:47）

返回

寄存器中保存的浮点数值

注解

浮点数输入寄存器的较低范围[0:23]保留供现场总线/PLC 接口使用。较高范围[24:47]不能通过现场总线/PLC 接口访问，因为它们是为外部RTDE 客户端保留的。

Python 函数原型

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

Lua 函数原型

getFloatInput(address: number) -> number

10.6 setFloatInput()

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

写入输入寄存器的浮点值

参数

address	寄存器的地址
value	浮点数值

返回

注解

只有在实现 RTDE/Modbus Slave/PLC 服务端时使用

Python 函数原型

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

Lua 函数原型

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

10.7 getDoubleInput()

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

读取寄存器地址的浮点数值

参数

address	寄存器的地址

返回

Python 函数原型

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

Lua 函数原型

getDoubleInput(address: number) -> number

10.8 setDoubleInput()

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

写入输入寄存器的浮点数值

参数

address	寄存器的地址
value	浮点数值

返回

注解

只有在实现 RTDE/Modbus Slave/PLC 服务端时使用

Python 函数原型

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

Lua 函数原型

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

10.9 getBoolOutput()

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

从一个输出寄存器中读取布尔值，也可以通过现场总线进行访问。注意，它使用自己的内存空间。

参数

address	寄存器地址（0:127）

返回

寄存器中保存的布尔值（true, false）

注解

布尔输出寄存器的较低范围[0:63]保留供现场总线/PLC 接口使用。较高范围[64:127]不能通过现场总线/PLC 接口访问，因为它们是为外部RTDE 客户端保留的。

Python 函数原型

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

Lua 函数原型

getBoolOutput(address: number) -> boolean

10.10 setBoolOutput()

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

写入寄存器地址的布尔值

参数

address	寄存器的地址
value	布尔值

返回

Python 函数原型

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

Lua 函数原型

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

10.11 getInt32Output()

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

从一个输出寄存器中读取整数值，也可以通过现场总线进行访问。注意，它使用自己的内存空间。

参数

address	寄存器地址（0:47）

返回

寄存器中保存的整数值（-2,147,483,648：2,147,483,647）

注解

整数输出寄存器的较低范围[0:23]保留供现场总线/PLC 接口使用。较高范围[24:47]不能通过现场总线/PLC 接口访问，因为它们是为外部RTDE 客户端保留的。

Python 函数原型

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

Lua 函数原型

getInt32Output(address: number) -> number

10.12 setInt32Output()

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

写入输出寄存器的整数值

参数

address	寄存器的地址
value	整数值

返回

Python 函数原型

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

Lua 函数原型

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

10.13 getFloatOutput()

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

从一个输出寄存器中读取浮点数，也可以通过现场总线进行访问。注意，它使用自己的内存空间。

参数

address	寄存器地址（0:47）

返回

寄存器中保存的浮点数值（float）

注解

浮点数输出寄存器的较低范围[0:23]保留供现场总线/PLC 接口使用。较高范围[24:47]不能通过现场总线/PLC 接口访问，因为它们是为外部RTDE 客户端保留的。

Python 函数原型

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

Lua 函数原型

getFloatOutput(address: number) -> number

10.14 setFloatOutput()

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

写入输出寄存器的浮点数值

参数

address	寄存器的地址
value	浮点数值

返回

Python 函数原型

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

Lua 函数原型

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

10.15 getDoubleOutput()

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

读取输出寄存器的浮点数值

参数

address	寄存器的地址

返回

Python 函数原型

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

Lua 函数原型

getDoubleOutput(address: number) -> number

10.16 setDoubleOutput()

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

写入输出寄存器的浮点数值

参数

address	寄存器的地址
value	浮点数值

返回

Python 函数原型

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

Lua 函数原型

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

10.17 getInt16Register()

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

用于 Modbus Slave 读取寄存器的整数值

参数

address	寄存器的地址

返回

Python 函数原型

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

Lua 函数原型

getInt16Register(address: number) -> number

10.18 setInt16Register()

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

写入寄存器的整数值

参数

address	寄存器的地址
value	整数值

返回

Python 函数原型

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

Lua 函数原型

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

10.19 hasNamedVariable()

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

具名变量是否存在

参数

key	变量名

返回

10.20 getNamedVariableType()

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

获取具名变量的类型

参数

key	变量名

返回

10.21 clearNamedVariable()

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

参数

key	变量名

返回

Python 函数原型

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

Lua 函数原型

clearNamedVariable(key: string) -> nil

10.22 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*)

添加一个新的Modbus 信号以供控制器监视。不需要返回响应。

参数

device_info	设备信息。设备信息是RTU 格式，例如： ”serial_port,baud,parity,data_bit,stop_bit” \n (1)serial_port 参数指定串口的名称，例如，在Linux 上为”/dev/ttyS0” 或”/dev/ttyUSB0”，在Windows 上为”.\COM10”\n (2)baud 参数指定通信的波特率，例如 9600、19200、57600、115200 等 \n (3)parity 参数指定奇偶校验方式，N 表示无校验，E 表示偶校验，O 表示奇校验 \n (4)data_bit 参数指定数据位数，允许的值为 5、6、7 和 8 \n (5)stop_bit 参数指定停止位数，允许的值为 1 和 2 设备信息是TCP 格式，例如：”ip address,port” (1)ip address 参数指定服务器的IP 地址 (2)port 参数指定服务器监听的端口号
slave_number	通常不使用，设置为 255 即可，但可以在 0 到 255 之间自由选择
signal_address	指定新信号应该反映的线圈或寄存器的地址。请参考Modbus 单元的配置以获取此信息。
signal_type	指定要添加的信号类型。0 = 数字输入，1 = 数字输出，2 = 寄存器输入，3 = 寄存器输出。
signal_name	唯一标识信号的名词。如果提供的字符串与已添加的信号相等，则新信号将替换旧信号。字符串的长度不能超过 20 个字符。
sequential_mode	设置为True 会强制Modbus 客户端在发送下一个请求之前等待响应。某些 fieldbus 单元需要此模式。可选参数。

返回

Python 函数原型

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

Lua 函数原型

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

10.23 modbusDeleteSignal()

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

删除指定名称的信号。

参数

signal_name	要删除的信号的名称

返回

Python 函数原型

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

Lua 函数原型

modbusDeleteSignal(signal_name: string) -> nil

10.24 modbusDeleteAllSignals()

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

删除所有的 modbus 信号

返回

10.25 modbusGetSignalStatus()

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

读取特定信号的当前值。

参数

signal_name	要获取值的信号的名称

返回

对于数字信号：1 或 0。对于寄存器信号：表示为整数的寄存器值。如果值为-1，则表示该信号不存在。

Python 函数原型

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

Lua 函数原型

modbusGetSignalStatus(signal_name: string) -> nil

10.26 modbusGetSignalNames()

std::vector< std::string > arcs::common_interface::RegisterControl::modbus↩ GetSignalNames ( )

获取所有信号的名字集合

返回

所有信号的名字集合

10.27 modbusGetSignalTypes()

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

获取所有信号的类型集合

返回

所有信号的类型集合

10.28 modbusGetSignalValues()

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

获取所有信号的数值集合

返回

所有信号的数值集合

10.29 modbusGetSignalErrors()

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

获取所有信号的请求是否有错误 (0: 无错误, 其他: 有错误) 集合

返回

Modbus 错误码集合

10.30 modbusSendCustomCommand()

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

将用户指定的命令发送到指定IP 地址上的Modbus 单元。由于不会接收到响应，因此不能用于请求数据。用户负责提供对所提供的功能码有意义的数据。内置函数负责构建Modbus 帧，因此用户不需要关心命令的长度。

参数

device_info	设备信息。设备信息是RTU 格式，例如： "serial_port,baud,parity,data_bit,stop_bit" \n (1)serial_port 参数指定串口的名称，例如，在Linux 上为"/dev/ttyS0" 或"/dev/ttyUSB0"，在Windows 上为".\COM10"\n (2)baud 参数指定通信的波特率，例如 9600、19200、57600、115200 等 \n (3)parity 参数指定奇偶校验方式，N 表示无校验，E 表示偶校验，O 表示奇校验 \n (4)data_bit 参数指定数据位数，允许的值为 5、6、7 和 8 \n (5)stop_bit 参数指定停止位数，允许的值为 1 和 2 设备信息是TCP 格式，例如："ip address,port" (1)ip address 参数指定服务器的IP 地址 (2)port 参数指定服务器监听的端口号
slave_number	指定用于自定义命令的从站号
function_code	指定自定义命令的功能码

参数

data	必须是有效的字节值（0-255）

返回

Python 函数原型

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

Lua 函数原型

modbusSendCustomCommand(device_info: string, slave_number: number, function_↩ code: number, data: table) -> nil

10.31 modbusSetDigitalInputAction()

int arcs::common_interface::RegisterControl::modbusSetDigitalInputAction (
const std::string & *robot_name,*
const std::string & *signal_name,*
StandardInputAction *action*)

将选择的数字输入信号设置为“default”或“freedrive”

参数

robot_name	连接的机器人名称
signal_name	先前被添加的数字输入信号
action	动作类型。动作可以是“default”或“freedrive”

返回

Python 函数原型

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

Lua 函数原型

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

10.32 modbusSetOutputSignal()

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

将指定名称的输出寄存器信号设置为给定的值

参数

signal_name	提前被添加的输出寄存器信号
value	必须是有效的整数，范围是 0-65535

返回

Python 函数原型

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

Lua 函数原型

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

10.33 modbusSetSignalUpdateFrequency()

int arcs::common_interface::RegisterControl::modbusSetSignalUpdateFrequency (

const std::string & *signal_name,*
int *update_frequency*)

设置机器人向Modbus 控制器发送请求的频率，用于读取或写入信号值

参数

signal_name	提前被添加的输出数字信号
update_frequency	更新频率（以赫兹为单位），范围是 0-125

返回

Python 函数原型

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

Lua 函数原型

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

11. RobotInterface 类

11.1 getRobotConfig()

RobotConfigPtr arcs::common_interface::RobotInterface::getRobotConfig ( )

获取RobotConfig 接口

返回

RobotConfigPtr 对象的指针

Python 函数原型

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

C++ 示例

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

11.2 getMotionControl()

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

获取运动规划接口

返回

MotionControlPtr 对象的指针

Python 函数原型

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

C++ 示例

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

11.3 getForceControl()

ForceControlPtr arcs::common_interface::RobotInterface::getForceControl ( )

获取力控接口

返回

ForceControlPtr 对象的指针

Python 函数原型

getForceControl(self: pyaubo_sdk.RobotInterface) -> arcs::common_interface::ForceControl

C++ 示例

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

11.4 getIoControl()

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

获取IO 控制的接口

返回

IoControlPtr 对象的指针

Python 函数原型

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

C++ 示例

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

11.5 getSyncMove()

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

获取同步运动接口

返回

SyncMovePtr 对象的指针

Python 函数原型

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

C++ 示例

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

11.6 getRobotAlgorithm()

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

获取机器人实用算法接口

返回

RobotAlgorithmPtr 对象的指针

Python 函数原型

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

C++ 示例

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

11.7 getRobotManage()

RobotManagePtr arcs::common_interface::RobotInterface::getRobotManage ( )

获取机器人管理接口 (上电、启动、停止等)

返回

RobotManagePtr 对象的指针

Python 函数原型

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

C++ 示例

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

11.8 getRobotState()

RobotStatePtr arcs::common_interface::RobotInterface::getRobotState ( )

获取机器人状态接口

返回

RobotStatePtr 对象的指针

Python 函数原型

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

C++ 示例

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

11.9 getTrace()

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

获取告警信息接口

返回

TracePtr 对象的指针

Python 函数原型

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

C++ 示例

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

12. RuntimeMachine 类

12.1 newTask()

int arcs::common_interface::RuntimeMachine::newTask (
bool *daemon* = false **)**

返回 task_id

12.2 deleteTask()

int arcs::common_interface::RuntimeMachine::deleteTask (
int *tid*)

删除 task

12.3 setPlanContext()

int arcs::common_interface::RuntimeMachine::setPlanContext (
int *tid,*
int *lineno,*
const std::string & *comment*)

向 aubo_control 日志中添加注释

参数

tid	指令的线程ID
lineno	行号
comment	注释

返回

Python 函数原型

setPlanContext(self: pyaubo_sdk.RuntimeMachine, arg0: int, arg1: int, arg2: str) -> int

Lua 函数原型

setPlanContext(tid: number, lineno: number, comment: string) -> number

12.4 nop()

int arcs::common_interface::RuntimeMachine::nop ( )

空操作

返回

12.5 getExecutionStatus()

std::tuple< std::string, std::string > arcs::common_interface::RuntimeMachine::getExecutionStatus ( )

获取耗时的接口执行状态, 如 setPersistentParameters

返回

指令名字, 执行状态执行状态: EXECUTING/FINISHED

Python 函数原型

getExecutionStatus(self: pyaubo_sdk.RuntimeMachine) -> Tuple[str, str]

Lua 函数原型

getExecutionStatus() -> string

12.6 gotoLine()

int arcs::common_interface::RuntimeMachine::gotoLine (
int *lineno*)

跳转到指定行号

参数

lineno	行号

返回

Python 函数原型

gotoLine(self: pyaubo_sdk.RuntimeMachine, arg0: int) -> int

Lua 函数原型

gotoLine(lineno: number) -> number

12.7 getPlanContext()

std::tuple< int, int, std::string > arcs::common_interface::RuntimeMachine↩ ::getPlanContext (
int *tid* = -1)

获取当前运行上下文

参数

tid	任务编号如果指定 (不是-1)，返回对应任务的运行上下文；如果不指定 (是-1)，返回正在运行的线程的运行上下文

返回

当前运行上下文

Python 函数原型

getPlanContext(self: pyaubo_sdk.RuntimeMachine) -> Tuple[int, int, str]

Lua 函数原型

getPlanContext() -> number

12.8 getAdvancePlanContext()

std::tuple< int, int, std::string > arcs::common_interface::RuntimeMachine↩ ::getAdvancePlanContext (
int *tid* = -1)

获取提前运行规划器的上下文信息

参数

tid	任务编号如果指定 (不是-1)，返回对应任务运行规划器的上下文信息；如果不指定 (是-1)，返回正在运行的线程运行规划器的上下文信息

返回

提前运行规划器的上下文信息

12.9 getAdvancePtr()

int arcs::common_interface::RuntimeMachine::getAdvancePtr ( )

获取AdvanceRun 的程序指针

返回

AdvanceRun 的程序指针

12.10 getMainPtr()

int arcs::common_interface::RuntimeMachine::getMainPtr (
int *tid* = -1)

获取机器人运动的程序指针

参数

tid	任务编号如果指定 (不是-1)，返回对应任务的程序指针；如果不指定 (是-1)，返回正在运行线程的程序指针

返回

机器人运动的程序指针

12.11 loadProgram()

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

加载本地工程文件 Lua 脚本，只需要给出文件名字，不需要后缀，需要从 ${ARCS_↩ WS}/program 目录中查找

参数

program	程序文件名字

返回

12.12 runProgram()

int arcs::common_interface::RuntimeMachine::runProgram ( )

运行已经加载的工程文件

返回

12.13 start()

int arcs::common_interface::RuntimeMachine::start ( )

开始运行时

返回

Python 函数原型

start(self: pyaubo_sdk.RuntimeMachine) -> int

Lua 函数原型

start() -> number

12.14 stop()

int arcs::common_interface::RuntimeMachine::stop ( )

终止运行时

返回

Python 函数原型

stop(self: pyaubo_sdk.RuntimeMachine) -> int

Lua 函数原型

stop() -> number

12.15 abort()

int arcs::common_interface::RuntimeMachine::abort ( )

终止脚本运行 (在关节空间进行规划，以最快速度退出)

返回

Python 函数原型

abort(self: pyaubo_sdk.RuntimeMachine) -> int

Lua 函数原型

abort() -> number

12.16 pause()

int arcs::common_interface::RuntimeMachine::pause ( )

暂停脚本运行

返回

Python 函数原型

pause(self: pyaubo_sdk.RuntimeMachine) -> int

Lua 函数原型

pause() -> number

12.17 step()

int arcs::common_interface::RuntimeMachine::step ( )

单步运行

返回

Python 函数原型

step(self: pyaubo_sdk.RuntimeMachine) -> int

Lua 函数原型

step() -> number

12.18 resume()

int arcs::common_interface::RuntimeMachine::resume ( )

恢复脚本运行

返回

Python 函数原型

resume(self: pyaubo_sdk.RuntimeMachine) -> int

Lua 函数原型

resume() -> number

12.19 getStatus()

RuntimeState arcs::common_interface::RuntimeMachine::getStatus ( )

获取脚本运行的状态

返回

脚本运行的状态

Python 函数原型

getStatus(self: pyaubo_sdk.RuntimeMachine) -> arcs::common_interface::RuntimeState

Lua 函数原型

getStatus() -> number

12.20 setBreakPoint()

int arcs::common_interface::RuntimeMachine::setBreakPoint (
int *lineno*)

设置断点

参数

lineno	行号

返回

Python 函数原型

setBreakPoint(self: pyaubo_sdk.RuntimeMachine, arg0: int) -> int

Lua 函数原型

setBreakPoint(lineno: number) -> number

12.21 removeBreakPoint()

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

移除断点

参数

lineno	行号

返回

Python 函数原型

removeBreakPoint(self: pyaubo_sdk.RuntimeMachine, arg0: int) -> int

Lua 函数原型

removeBreakPoint(lineno: number) -> number

12.22 clearBreakPoints()

int arcs::common_interface::RuntimeMachine::clearBreakPoints ( )

清除所有断点

返回

Python 函数原型

clearBreakPoints(self: pyaubo_sdk.RuntimeMachine) -> int

Lua 函数原型

clearBreakPoints() -> number

12.23 timerStart()

int arcs::common_interface::RuntimeMachine::timerStart (
const std::string & *name*)

定时器开始

参数

name	定时器名称

返回

Python 函数原型

timerStart(self: pyaubo_sdk.RuntimeMachine, arg0: str) -> int

Lua 函数原型

timerStart(name: string) -> nil

12.24 timerStop()

int arcs::common_interface::RuntimeMachine::timerStop (
const std::string & *name*)

定时器结束

参数

name	定时器名称

返回

Python 函数原型

timerStop(self: pyaubo_sdk.RuntimeMachine, arg0: str) -> int

Lua 函数原型

timerStop(name: string) -> nil

12.25 timerReset()

int arcs::common_interface::RuntimeMachine::timerReset (
const std::string & *name*)

定时器重置

参数

name	定时器名称

返回

Python 函数原型

timerReset(self: pyaubo_sdk.RuntimeMachine, arg0: str) -> int

Lua 函数原型

timerReset(name: string) -> nil

12.26 timerDelete()

int arcs::common_interface::RuntimeMachine::timerDelete (
const std::string & *name*)

定时器删除

参数

name	定时器名称

返回

Python 函数原型

timerDelete(self: pyaubo_sdk.RuntimeMachine, arg0: str) -> int

Lua 函数原型

timerDelete(name: string) -> nil

12.27 getTimer()

double arcs::common_interface::RuntimeMachine::getTimer (
const std::string & *name*)

获取定时器数值

参数

name	定时器名称

返回

Python 函数原型

getTimer(self: pyaubo_sdk.RuntimeMachine, arg0: str) -> float

Lua 函数原型

getTimer(name: string) -> number

13. SystemInfo 类

13.1 getControlSoftwareVersionCode()

int arcs::common_interface::SystemInfo::getControlSoftwareVersionCode ( )

获取控制器软件版本号

返回

返回控制器软件版本号

Python 函数原型

getControlSoftwareVersionCode(self: pyaubo_sdk.SystemInfo) -> int

Lua 函数原型

getControlSoftwareVersionCode() -> number

C++ 示例

```
int control_version =
rpc_cli->getSystemInfo()->getControlSoftwareVersionCode();
```

13.2 getInterfaceVersionCode()

int arcs::common_interface::SystemInfo::getInterfaceVersionCode ( )

获取接口版本号

返回

返回接口版本号

Python 函数原型

getInterfaceVersionCode(self: pyaubo_sdk.SystemInfo) -> int

Lua 函数原型

getInterfaceVersionCode() -> number

C++ 示例

```
int interface_version =
rpc_cli->getSystemInfo()->getInterfaceVersionCode();
```

13.3 getControlSoftwareBuildDate()

std::string arcs::common_interface::SystemInfo::getControlSoftwareBuildDate ( )

获取控制器软件构建时间

返回

返回控制器软件构建时间

Python 函数原型

getControlSoftwareBuildDate(self: pyaubo_sdk.SystemInfo) -> str

Lua 函数原型

getControlSoftwareBuildDate() -> string

C++ 示例

```
std::string build_date =
rpc_cli->getSystemInfo()->getControlSoftwareBuildDate();
```

13.4 getControlSoftwareVersionHash()

std::string arcs::common_interface::SystemInfo::getControlSoftwareVersion↩ Hash ( )

获取控制器软件 git 版

返回

返回控制器软件 git 版本

Python 函数原型

getControlSoftwareVersionHash(self: pyaubo_sdk.SystemInfo) -> str

Lua 函数原型

getControlSoftwareVersionHash() -> string

C++ 示例

```
std::string git_version =
rpc_cli->getSystemInfo()->getControlSoftwareVersionHash();
```

13.5 getControlSystemTime()

uint64_t arcs::common_interface::SystemInfo::getControlSystemTime ( )

获取系统时间 (软件启动时间 us)

返回

返回系统时间 (软件启动时间 us)

Python 函数原型

getControlSystemTime(self: pyaubo_sdk.SystemInfo) -> int

Lua 函数原型

getControlSystemTime() -> number

C++ 示例

```
std::string system_time =
rpc_cli->getSystemInfo()->getControlSystemTime();
```

14. Trace 类

14.1 alarm()

int arcs::common_interface::Trace::alarm (
TraceLevel *level,*
int *code,*
const std::vector< std::string > & *args* = {})

向 aubo_control 日志注入告警信息

TraceLevel:
0 - FATAL
1 - ERROR
2 - WARNING
3 - INFO
4 - DEBUG
code 定义参考 error_stack

参数

level	日志等级
code	错误码
args	日志信息描述

返回

Python 函数原型

alarm(self: pyaubo_sdk.Trace, arg0: arcs::common_interface::TraceLevel, arg1: int, arg2: List[str]) -> int

Lua 函数原型

alarm(level: number, code: number, args: table) -> nil

14.2 textmsg()

int arcs::common_interface::Trace::textmsg (
const std::string & *msg*)

打印文本信息到日志中

参数

msg	文本信息

返回

Python 函数原型

textmsg(self: pyaubo_sdk.Trace, arg0: str) -> int

Lua 函数原型

textmsg(msg: string) -> nil

14.3 popup()

int arcs::common_interface::Trace::popup (
TraceLevel *level,*
const std::string & *title,*
const std::string & *msg,*
int *mode*)

向连接的 RTDE 客户端发送弹窗请求

参数

level	等级
title	标题
msg	信息
mode	模式 0: 普通模式 1: 阻塞模式 2: 输入模式 bool 3: 输入模式 int 4: 输入模式 double 5: 输入模式 string

返回

Python 函数原型

popup(self: pyaubo_sdk.Trace, arg0: arcs::common_interface::TraceLevel, arg1: str, arg2: str, arg3: int) -> int

Lua 函数原型

popup(level: number, title: string, msg: string, mode: number) -> nil

14.4 peek()

RobotMsgVector arcs::common_interface::Trace::peek (
size_t *num*,
uint64_t *last_time* = 0)

peek 最新的 AlarmInfo(上次一获取之后)

last_time 设置为 0 时，可以获取到所有的AlarmInfo

参数

num	数量
last_time	时间

返回

获取 last_time 这个时间之后的所有错误

Python 函数原型

peek(self: pyaubo_sdk.Trace, arg0: int, arg1: int) -> List[arcs::common_interface::RobotMsg]

Lua 函数原型

peek(num: number, last_time: number) -> table

15. ForceControl 类

15.1 fcEnable()

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

使能力控

返回值

0	成功

Python 函数原型

fcEnable(self: pyaubo_sdk.ForceControl) -> int

Lua 函数原型

fcEnable() -> nil

15.2 fcDisable()

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

失能力控

返回值

0	成功

Python 函数原型

fcDisable(self: pyaubo_sdk.ForceControl) -> int

Lua 函数原型

fcDisable() -> nil

15.3 isFcEnabled()

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

判断力控是否被使能

返回

使能返回 true，失能返回 false

Python 函数原型

isFcEnabled(self: pyaubo_sdk.ForceControl) -> bool

Lua 函数原型

isFcEnabled() -> boolean

15.4 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 **)**

设置力控参考 (目标) 值

参数

feature	参考几何特征，用于生成力控参考坐标系
compliance	柔性轴（方向）选择
wrench	目标力/力矩
limits	速度限制
type	力控参考坐标系类型

Python 函数原型

setTargetForce(self: pyaubo_sdk.ForceControl, arg0: List[float], arg1: List[bool], arg2↩
: List[float], arg3: List[float], arg4: arcs::common_interface::TaskFrameType) -> int

Lua 函数原型

setTargetForce(feature: table, compliance: table, wrench: table, limits: table, type: number) -> nil

15.5 setDynamicModel()

int arcs::common_interface::ForceControl::setDynamicModel (
const std::vector< double > & *m*,
const std::vector< double > & *d*,
const std::vector< double > & *k*)

设置力控动力学模型

参数

m	质量
d	阻尼
k	刚度

返回

Python 函数原型

setDynamicModel(self: pyaubo_sdk.ForceControl, arg0: List[float], arg1: List[float], arg2: List[float]) -> int

Lua 函数原型

setDynamicModel(m: table, d: table, k: table) -> nil

15.6 setLpFilter()

int arcs::common_interface::ForceControl::setLpFilter (
const std::vector< double > & *cutoff_freq*)

设置低通滤波器

参数

cutoff_freq	截止频率

返回

Python 函数原型

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

Lua 函数原型

setLpFilter(cutoff_freq: table) -> nil

16. InputParser 类

16.1 arcs::aubo_sdk::InputParser 类参考

Public 成员函数

- int popInt32 ()
- int64_t popInt64 ()
- double popDouble ()
- char popChar ()
- std::vector< int > popVectorInt ()
- std::vector< double > popVectorDouble ()
- std::vector< common_interface::JointStateType > popVectorJointStateType ()
- common_interface::RobotModeType popRobotModeType ()
- common_interface::SafetyModeType popSafetyModeType ()
- common_interface::RuntimeState popRuntimeState ()
- common_interface::RobotMsgVector popRobotMsgVector ()

17. IoControl 类

17.1 getStandardDigitalInputNum()

int arcs::common_interface::IoControl::getStandardDigitalInputNum ( )

获取标准数字输入数量

返回

标准数字输入数量

Python 函数原型

getStandardDigitalInputNum(self: pyaubo_sdk.IoControl) -> int

Lua 函数原型

getStandardDigitalInputNum() -> number

17.2 getToolDigitalInputNum()

int arcs::common_interface::IoControl::getToolDigitalInputNum ( )

获取工具端数字输入数量

返回

工具端数字输入数量

Python 函数原型

getToolDigitalInputNum(self: pyaubo_sdk.IoControl) -> int

Lua 函数原型

getToolDigitalInputNum() -> number

17.3 getConfigurableDigitalInputNum()

int arcs::common_interface::IoControl::getConfigurableDigitalInputNum ( )

获取可配置数字输入数量

返回

可配置数字输入数量

Python 函数原型

getConfigurableDigitalInputNum(self: pyaubo_sdk.IoControl) -> int

Lua 函数原型

getConfigurableDigitalInputNum() -> number

17.4 getStandardDigitalOutputNum()

int arcs::common_interface::IoControl::getStandardDigitalOutputNum ( )

获取标准数字输出数量

返回

标准数字输出数量

Python 函数原型

getStandardDigitalOutputNum(self: pyaubo_sdk.IoControl) -> int

Lua 函数原型

getStandardDigitalOutputNum() -> number

17.5 getToolDigitalOutputNum()

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

获取工具端数字输出数量

返回

工具端数字输出数量

Python 函数原型

getToolDigitalOutputNum(self: pyaubo_sdk.IoControl) -> int

Lua 函数原型

getToolDigitalOutputNum() -> number

17.6 setToolIoInput()

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

设置指定的工具端IO 为输入/输出

工具端IO 比较特殊，IO 可以配置为输入或者输出

参数

index	表示IO 口的管脚，可用十进制或者十六进制表示。。例如，0x00000000 表示第一个管脚
input	表示指定IO 是否为输入。input 为 true 时，设置指定IO 为输入，否则为输出

返回

Python 函数原型

setToolIoInput(self: pyaubo_sdk.IoControl, arg0: int, arg1: bool) -> int

Lua 函数原型

setToolIoInput(index: number, input: boolean) -> nil

17.7 isToolIoInput()

bool arcs::common_interface::IoControl::isToolIoInput (
int *index*)

判断指定IO 是否为输入

参数

index	表示IO 口的管脚，可用十进制或者十六进制表示。。例如，0x00000000 表示第一个管脚

返回

当指定的IO 为输入时返回 true, 否则为 false

Python 函数原型

isToolIoInput(self: pyaubo_sdk.IoControl, arg0: int) -> bool

Lua 函数原型

isToolIoInput(index: number) -> boolean

17.8 getConfigurableDigitalOutputNum()

int arcs::common_interface::IoControl::getConfigurableDigitalOutputNum ( )

获取可配置数字输出数量

返回

可配置数字输出数量

Python 函数原型

getConfigurableDigitalOutputNum(self: pyaubo_sdk.IoControl) -> int

Lua 函数原型

getConfigurableDigitalOutputNum() -> number

17.9 getStandardAnalogInputNum()

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

获取标准模拟输入数量

返回

标准模拟输入数量

Python 函数原型

getStandardAnalogInputNum(self: pyaubo_sdk.IoControl) -> int

Lua 函数原型

getStandardAnalogInputNum() -> number

17.10 getToolAnalogInputNum()

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

获取工具端模拟输入数量

返回

工具端模拟输入数量

Python 函数原型

getToolAnalogInputNum(self: pyaubo_sdk.IoControl) -> int

Lua 函数原型

getToolAnalogInputNum() -> number

17.11 getStandardAnalogOutputNum()

int arcs::common_interface::IoControl::getStandardAnalogOutputNum ( )

获取标准模拟输出数量

返回

标准模拟输出数量

Python 函数原型

getStandardAnalogOutputNum(self: pyaubo_sdk.IoControl) -> int

Lua 函数原型

getStandardAnalogOutputNum() -> number

17.12 getToolAnalogOutputNum()

int arcs::common_interface::IoControl::getToolAnalogOutputNum ( )

获取工具端模拟输出数量

返回

工具端模拟输出数量

Python 函数原型

getToolAnalogOutputNum(self: pyaubo_sdk.IoControl) -> int

Lua 函数原型

getToolAnalogOutputNum() -> number

17.13 setDigitalInputActionDefault()

int arcs::common_interface::IoControl::setDigitalInputActionDefault ( )

设置所有数字输入动作为默认值

返回

Python 函数原型

setDigitalInputActionDefault(self: pyaubo_sdk.IoControl) -> int

Lua 函数原型

setDigitalInputActionDefault() -> nil

17.14 setStandardDigitalInputAction()

int arcs::common_interface::IoControl::setStandardDigitalInputAction (
int *index,*
StandardInputAction *action*)

设置标准数字输入触发动作

参数

index	表示IO 口的管脚，可用十进制或者十六进制表示。。例如，0x00000000 表示第一个管脚
action	触发动作

返回

Python 函数原型

setStandardDigitalInputAction(self: pyaubo_sdk.IoControl, arg0: int, arg1: arcs↩ ::common_interface::StandardInputAction) -> int

Lua 函数原型

setStandardDigitalInputAction(index: number, action: number) -> nil

17.15 setToolDigitalInputAction()

int arcs::common_interface::IoControl::setToolDigitalInputAction (
int *index,*
StandardInputAction *action*)

设置工具数字输入触发动作

参数

index	表示IO 口的管脚，可用十进制或者十六进制表示。。例如，0x00000000 表示第一个管脚
action	触发动作

返回

Python 函数原型

setToolDigitalInputAction(self: pyaubo_sdk.IoControl, arg0: int, arg1: arcs::common_↩ interface::StandardInputAction) -> int

Lua 函数原型

setToolDigitalInputAction(index: number, action: number) -> nil

17.16 setConfigurableDigitalInputAction()

int arcs::common_interface::IoControl::setConfigurableDigitalInputAction (
int *index*,
StandardInputAction *action*)

设置可配置数字输入触发动作

参数

index	表示IO 口的管脚，可用十进制或者十六进制表示。。例如，0x00000000 表示第一个管脚
action	触发动作

返回

注解

需要将可配置输入的安全输入动作设置为 SafetyInputAction::Unassigned 时这个函数的配置才会生效

Python 函数原型

setConfigurableDigitalInputAction(self: pyaubo_sdk.IoControl, arg0: int, arg1: arcs↩ ::common_interface::StandardInputAction) -> int

Lua 函数原型

setConfigurableDigitalInputAction(index: number, action: number) -> nil

17.17 getStandardDigitalInputAction()

StandardInputAction arcs::common_interface::IoControl::getStandardDigital↩ InputAction (
int *index*)

获取标准数字输入的输入触发动作

参数

index	表示IO 口的管脚，可用十进制或者十六进制表示。。例如，0x00000000 表示第一个管脚

返回

标准数字输入的输入触发动作

Python 函数原型

getStandardDigitalInputAction(self: pyaubo_sdk.IoControl, arg0: int) -> arcs::common↩ _interface::StandardInputAction

Lua 函数原型

getStandardDigitalInputAction(index: number) -> number

17.18 getToolDigitalInputAction()

StandardInputAction arcs::common_interface::IoControl::getToolDigitalInput↩ Action (
int *index*)

获取工具端数字输入的输入触发动作

参数

index	表示IO 口的管脚，可用十进制或者十六进制表示。。例如，0x00000000 表示第一个管脚

返回

工具端数字输入的输入触发动作

Python 函数原型

getToolDigitalInputAction(self: pyaubo_sdk.IoControl, arg0: int) -> arcs::common_↩ interface::StandardInputAction

Lua 函数原型

getToolDigitalInputAction(index: number) -> number

17.19 getConfigurableDigitalInputAction()

StandardInputAction arcs::common_interface::IoControl::getConfigurable↩ DigitalInputAction (
int *index*)

获取可配置数字输入的输入触发动作

参数

index	表示IO 口的管脚，可用十进制或者十六进制表示。。例如，0x00000000 表示第一个管脚

返回

输入触发动作

Python 函数原型

getConfigurableDigitalInputAction(self: pyaubo_sdk.IoControl, arg0: int) -> arcs↩ ::common_interface::StandardInputAction

Lua 函数原型

getConfigurableDigitalInputAction(index: number) -> number

17.20 setDigitalOutputRunstateDefault()

int arcs::common_interface::IoControl::setDigitalOutputRunstateDefault ( )

设置所有数字输出动作为默认值

返回

Python 函数原型

setDigitalOutputRunstateDefault(self: pyaubo_sdk.IoControl) -> int

Lua 函数原型

setDigitalOutputRunstateDefault() -> nil

17.21 setStandardDigitalOutputRunstate()

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

设置标准数字输出状态选择

参数

index	表示IO 口的管脚，可用十进制或者十六进制表示。。例如，0x00000000 表示第一个管脚
runstate	输出状态选择

返回

Python 函数原型

setStandardDigitalOutputRunstate(self: pyaubo_sdk.IoControl, arg0: int, arg1: arcs↩
::common_interface::StandardOutputRunState) -> int

Lua 函数原型

setStandardDigitalOutputRunstate(index: number, runstate: number) -> nil

17.22 setToolDigitalOutputRunstate()

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

设置工具端数字输出状态选择

参数

index	表示IO 口的管脚，可用十进制或者十六进制表示。。例如，0x00000000 表示第一个管脚
runstate	输出状态选择

返回

Python 函数原型

setToolDigitalOutputRunstate(self: pyaubo_sdk.IoControl, arg0: int, arg1: arcs↩
::common_interface::StandardOutputRunState) -> int

Lua 函数原型

setToolDigitalOutputRunstate(index: number, runstate: number) -> nil

17.23 setConfigurableDigitalOutputRunstate()

int arcs::common_interface::IoControl::setConfigurableDigitalOutputRunstate (
int *index,*
StandardOutputRunState *runstate*)

设置可配置数字输出状态选择

参数

index	表示IO 口的管脚，可用十进制或者十六进制表示。。例如，0x00000000 表示第一个管脚
runstate	输出状态选择

返回

Python 函数原型

setConfigurableDigitalOutputRunstate(self: pyaubo_sdk.IoControl, arg0: int, arg1: arcs↩
::common_interface::StandardOutputRunState) -> int

Lua 函数原型

setConfigurableDigitalOutputRunstate(index: number, runstate: number) -> nil

17.24 getStandardDigitalOutputRunstate()

**StandardOutputRunState arcs::common_interface::IoControl::getStandard↩
DigitalOutputRunstate (**
int *index*)

获取标准数字输出状态选择

参数

index	表示IO 口的管脚，可用十进制或者十六进制表示。。例如，0x00000000 表示第一个管脚

返回

输出状态选择

Python 函数原型

getStandardDigitalOutputRunstate(self: pyaubo_sdk.IoControl, arg0: int) -> arcs↩
::common_interface::StandardOutputRunState

Lua 函数原型

getStandardDigitalOutputRunstate(index: number) -> number

17.25 getToolDigitalOutputRunstate()

StandardOutputRunState arcs::common_interface::IoControl::getToolDigital↩ OutputRunstate (
int *index*)

获取工具端数字输出状态选择

参数

index	表示IO 口的管脚，可用十进制或者十六进制表示。。例如，0x00000000 表示第一个管脚

返回

输出状态选择

Python 函数原型

getToolDigitalOutputRunstate(self: pyaubo_sdk.IoControl, arg0: int) -> arcs::common↩ _interface::StandardOutputRunState

Lua 函数原型

getToolDigitalOutputRunstate(index: number) -> number

17.26 getConfigurableDigitalOutputRunstate()

StandardOutputRunState arcs::common_interface::IoControl::getConfigurable↩ DigitalOutputRunstate (
int *index*)

获取可配置数字输出状态选择

参数

index	表示IO 口的管脚，可用十进制或者十六进制表示。。例如，0x00000000 表示第一个管脚

返回

输出状态选择

Python 函数原型

getConfigurableDigitalOutputRunstate(self: pyaubo_sdk.IoControl, arg0: int) -> arcs↩
::common_interface::StandardOutputRunState

Lua 函数原型

getConfigurableDigitalOutputRunstate(index: number) -> number

17.27 setStandardAnalogOutputRunstate()

int arcs::common_interface::IoControl::setStandardAnalogOutputRunstate (
int *index*,
StandardOutputRunState *runstate*)

设置标准模拟输出状态选择

参数

index	表示IO 口的管脚，可用十进制或者十六进制表示。。例如，0x00000000 表示第一个管脚
runstate	输出状态选择

返回

Python 函数原型

setStandardAnalogOutputRunstate(self: pyaubo_sdk.IoControl, arg0: int, arg1: arcs↩
::common_interface::StandardOutputRunState) -> int

Lua 函数原型

setStandardAnalogOutputRunstate(index: number, runstate: number) -> nil

17.28 setToolAnalogOutputRunstate()

int arcs::common_interface::IoControl::setToolAnalogOutputRunstate (
int *index*,
StandardOutputRunState *runstate*)

设置工具端模拟输出状态选择

参数

index	表示IO 口的管脚，可用十进制或者十六进制表示。例如，0x00000000 表示第一个管脚
runstate	输出状态选择

返回

Python 函数原型

setToolAnalogOutputRunstate(self: pyaubo_sdk.IoControl, arg0: int, arg1: arcs↩
::common_interface::StandardOutputRunState) -> int

Lua 函数原型

setToolAnalogOutputRunstate(index: number, runstate: number) -> nil

17.29 getStandardAnalogOutputRunstate()

**StandardOutputRunState arcs::common_interface::IoControl::getStandard↩
AnalogOutputRunstate (**
int *index*)

获取标准模拟输出状态选择

参数

index	表示IO 口的管脚，可用十进制或者十六进制表示。例如，0x00000000 表示第一个管脚

返回

标准模拟输出状态选择

Python 函数原型

getStandardAnalogOutputRunstate(self: pyaubo_sdk.IoControl, arg0: int) -> arcs↩
::common_interface::StandardOutputRunState

Lua 函数原型

getStandardAnalogOutputRunstate(index: number) -> number

17.30 getToolAnalogOutputRunstate()

**StandardOutputRunState arcs::common_interface::IoControl::getToolAnalog↩
OutputRunstate (**
int *index*)

获取工具端模拟输出状态选择

参数

index	表示IO 口的管脚，可用十进制或者十六进制表示。例如，0x00000000 表示第一个管脚

返回

工具端模拟输出状态选择

Python 函数原型

getToolAnalogOutputRunstate(self: pyaubo_sdk.IoControl, arg0: int) -> arcs::common↩
_interface::StandardOutputRunState

Lua 函数原型

getToolAnalogOutputRunstate(index: number) -> number

17.31 setStandardAnalogInputDomain()

int arcs::common_interface::IoControl::setStandardAnalogInputDomain (
int *index,*
int *domain*)

设置标准模拟输入的范围

参数

index	表示IO 口的管脚，可用十进制或者十六进制表示。例如，0x00000000 表示第一个管脚
domain	输入的范围

返回

Python 函数原型

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

Lua 函数原型

setStandardAnalogInputDomain(index: number, domain: number) -> nil

17.32 setToolAnalogInputDomain()

int arcs::common_interface::IoControl::setToolAnalogInputDomain (
int *index,*
int *domain*)

设置工具端模拟输入的范围

参数

index	表示IO 口的管脚，可用十进制或者十六进制表示。例如，0x00000000 表示第一个管脚
domain	输入的范围

返回

Python 函数原型

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

Lua 函数原型

setToolAnalogInputDomain(index: number, domain: number) -> nil

17.33 getStandardAnalogInputDomain()

int arcs::common_interface::IoControl::getStandardAnalogInputDomain (
int *index*)

获取标准模式输入范围

参数

index	表示IO 口的管脚，可用十进制或者十六进制表示。例如，0x00000000 表示第一个管脚

返回

标准模式输入范围

Python 函数原型

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

Lua 函数原型

getStandardAnalogInputDomain(index: number) -> number

17.34 getToolAnalogInputDomain()

int arcs::common_interface::IoControl::getToolAnalogInputDomain (
int *index*)

获取工具端模式输入范围

参数

index	表示IO 口的管脚，可用十进制或者十六进制表示。例如，0x00000000 表示第一个管脚

返回

工具端模式输入范围

Python 函数原型

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

Lua 函数原型

getToolAnalogInputDomain(index: number) -> number

17.35 setStandardAnalogOutputDomain()

int arcs::common_interface::IoControl::setStandardAnalogOutputDomain (
int *index*,
int *domain*)

设置标准模拟输出的范围

参数

index	表示IO 口的管脚，可用十进制或者十六进制表示。例如，0x00000000 表示第一个管脚
domain	输出的范围

返回

Python 函数原型

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

Lua 函数原型

setStandardAnalogOutputDomain(index: number, domain: number) -> nil

17.36 setToolAnalogOutputDomain()

int arcs::common_interface::IoControl::setToolAnalogOutputDomain (
int *index,*
int *domain*)

设置工具端模拟输出范围

参数

index	表示IO 口的管脚，可用十进制或者十六进制表示。例如，0x00000000 表示第一个管脚
domain	输出的范围

返回

Python 函数原型

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

Lua 函数原型

setToolAnalogOutputDomain(index: number, domain: number) -> nil

17.37 getStandardAnalogOutputDomain()

int arcs::common_interface::IoControl::getStandardAnalogOutputDomain (
int *index*)

获取标准模拟输出范围

参数

index	表示IO 口的管脚，可用十进制或者十六进制表示。例如，0x00000000 表示第一个管脚

返回

标准模拟输出范围

Python 函数原型

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

Lua 函数原型

getStandardAnalogOutputDomain(index: number) -> number

17.38 getToolAnalogOutputDomain()

int arcs::common_interface::IoControl::getToolAnalogOutputDomain (
int *index*)

获取工具端模拟输出范围

参数

index	表示IO 口的管脚，可用十进制或者十六进制表示。例如，0x00000000 表示第一个管脚

返回

工具端模拟输出范围

Python 函数原型

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

Lua 函数原型

getToolAnalogOutputDomain(index: number) -> number

17.39 setToolVoltageOutputDomain()

int arcs::common_interface::IoControl::setToolVoltageOutputDomain (
int *domain*)

设置工具端电源电压

参数

domain	可选三个档位 0: 0V, 12: 12V, 24: 24V

返回

Python 函数原型

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

Lua 函数原型

setToolVoltageOutputDomain(domain: number) -> nil

17.40 getToolVoltageOutputDomain()

int arcs::common_interface::IoControl::getToolVoltageOutputDomain ( )

获取工具端电源电压

返回

工具端电源电压

Python 函数原型

getToolVoltageOutputDomain(self: pyaubo_sdk.IoControl) -> int

Lua 函数原型

getToolVoltageOutputDomain() -> number

17.41 setStandardDigitalOutput()

int arcs::common_interface::IoControl::setStandardDigitalOutput (
int *index,*
bool *value*)

设置标准数字输出

参数

index	表示IO 口的管脚，可用十进制或者十六进制表示。例如，0x00000000 表示第一个管脚
value	输出

返回

Python 函数原型

setStandardDigitalOutput(self: pyaubo_sdk.IoControl, arg0: int, arg1: bool) -> int

Lua 函数原型

setStandardDigitalOutput(index: number, value: boolean) -> nil

17.42 setToolDigitalOutput()

int arcs::common_interface::IoControl::setToolDigitalOutput (
int *index*,
bool *value*)

设置工具端数字输出

参数

index	表示IO 口的管脚，可用十进制或者十六进制表示。例如，0x00000000 表示第一个管脚
value	数字输出

返回

Python 函数原型

setToolDigitalOutput(self: pyaubo_sdk.IoControl, arg0: int, arg1: bool) -> int

Lua 函数原型

setToolDigitalOutput(index: number, value: boolean) -> nil

17.43 setConfigurableDigitalOutput()

int arcs::common_interface::IoControl::setConfigurableDigitalOutput (
int *index,*
bool *value*)

设置可配置数字输出

参数

index	表示IO 口的管脚，可用十进制或者十六进制表示。例如，0x00000000 表示第一个管脚
value	数字输出

返回

Python 函数原型

setConfigurableDigitalOutput(self: pyaubo_sdk.IoControl, arg0: int, arg1: bool) -> int

Lua 函数原型

setConfigurableDigitalOutput(index: number, value: boolean) -> nil

17.44 setStandardAnalogOutput()

int arcs::common_interface::IoControl::setStandardAnalogOutput (
int *index,*
double *value*)

设置标准模拟输出

参数

index	表示IO 口的管脚，可用十进制或者十六进制表示。例如，0x00000000 表示第一个管脚
value	模拟输出

返回

Python 函数原型

setStandardAnalogOutput(self: pyaubo_sdk.IoControl, arg0: int, arg1: float) -> int

Lua 函数原型

setStandardAnalogOutput(index: number, value: number) -> nil

17.45 setToolAnalogOutput()

int arcs::common_interface::IoControl::setToolAnalogOutput (
int *index,*
double *value*)

设置工具端模拟输出

参数

index	表示IO 口的管脚，可用十进制或者十六进制表示。例如，0x00000000 表示第一个管脚
value	模拟输出

返回

Python 函数原型

setToolAnalogOutput(self: pyaubo_sdk.IoControl, arg0: int, arg1: float) -> int

Lua 函数原型

setToolAnalogOutput(index: number, value: number) -> nil

17.46 getStandardDigitalInput()

bool arcs::common_interface::IoControl::getStandardDigitalInput (
int *index*)

获取标准数字输入

参数

index	表示IO 口的管脚，可用十进制或者十六进制表示。例如，0x00000000 表示第一个管脚

返回

高电平返回 true; 低电平返回 false

Python 函数原型

getStandardDigitalInput(self: pyaubo_sdk.IoControl, arg0: int) -> bool

Lua 函数原型

getStandardDigitalInput(index: number) -> boolean

17.47 getStandardDigitalInputs()

uint32_t arcs::common_interface::IoControl::getStandardDigitalInputs ( )

获取所有的标准数字输入

返回

所有的标准数字输入例如,当返回值是 2863267846 时, 换成 2 进制后是 10101010101010100000000000000110。后 16 位就是所有的标准数字输入状态值，最后一位表示DI00 的输入状态值, 倒数第二位表示DI01 的输入状态值，以此类推。
1 表示高电平状态，0 表示低电平状态

Python 函数原型

getStandardDigitalInputs(self: pyaubo_sdk.IoControl) -> int

Lua 函数原型

getStandardDigitalInputs() -> number

17.48 getToolDigitalInput()

bool arcs::common_interface::IoControl::getToolDigitalInput (
int *index*)

获取工具端数字输入

参数

index	表示IO 口的管脚，可用十进制或者十六进制表示。例如，0x00000000 表示第一个管脚

返回

高电平返回 true; 低电平返回 false

Python 函数原型

getToolDigitalInput(self: pyaubo_sdk.IoControl, arg0: int) -> bool

Lua 函数原型

getToolDigitalInput(index: number) -> boolean

17.49 getToolDigitalInputs()

uint32_t arcs::common_interface::IoControl::getToolDigitalInputs ( )

获取所有的工具端数字输入

返回

所有的工具端数字输入

Python 函数原型

getToolDigitalInputs(self: pyaubo_sdk.IoControl) -> int

Lua 函数原型

getToolDigitalInputs() -> number

17.50 getConfigurableDigitalInput()

bool arcs::common_interface::IoControl::getConfigurableDigitalInput (
int *index*)

获取可配置数字输入

参数

index	表示IO 口的管脚，可用十进制或者十六进制表示。例如，0x00000000 表示第一个管脚

返回

高电平返回 true; 低电平返回 false

Python 函数原型

getConfigurableDigitalInput(self: pyaubo_sdk.IoControl, arg0: int) -> bool

Lua 函数原型

getConfigurableDigitalInput(index: number) -> boolean

17.51 getConfigurableDigitalInputs()

uint32_t arcs::common_interface::IoControl::getConfigurableDigitalInputs ( )

获取所有的可配置数字输入

返回

所有的可配置数字输入

Python 函数原型

getConfigurableDigitalInputs(self: pyaubo_sdk.IoControl) -> int

Lua 函数原型

getConfigurableDigitalInputs() -> number

17.52 getStandardDigitalOutput()

bool arcs::common_interface::IoControl::getStandardDigitalOutput (
int *index*)

获取标准数字输出

参数

index	表示IO 口的管脚，可用十进制或者十六进制表示。例如，0x00000000 表示第一个管脚

返回

高电平返回 true; 低电平返回 false

Python 函数原型

getStandardDigitalOutput(self: pyaubo_sdk.IoControl, arg0: int) -> bool

Lua 函数原型

getStandardDigitalOutput(index: number) -> boolean

17.53 getStandardDigitalOutputs()

uint32_t arcs::common_interface::IoControl::getStandardDigitalOutputs ( )

获取所有的标准数字输出

返回

所有的标准数字输出例如,当返回值是 2863267846 时, 换成 2 进制后是 10101010101010100000000000000110。后 16 位就是所有的标准数字输出状态值，最后一位表示DI00 的输出状态值, 倒数第二位表示DI01 的输出状态值，以此类推。
1 表示高电平状态，0 表示低电平状态.

Python 函数原型

getStandardDigitalOutputs(self: pyaubo_sdk.IoControl) -> int

Lua 函数原型

getStandardDigitalOutputs() -> number

17.54 getToolDigitalOutput()

bool arcs::common_interface::IoControl::getToolDigitalOutput (
int *index*)

获取工具端数字输出

参数

index	表示IO 口的管脚，可用十进制或者十六进制表示。例如，0x00000000 表示第一个管脚

返回

高电平返回 true; 低电平返回 false

Python 函数原型

getToolDigitalOutput(self: pyaubo_sdk.IoControl, arg0: int) -> bool

Lua 函数原型

getToolDigitalOutput(index: number) -> boolean

17.55 getToolDigitalOutputs()

uint32_t arcs::common_interface::IoControl::getToolDigitalOutputs ( )

获取所有的工具端数字输出

返回

所有的工具端数字输出

Python 函数原型

getToolDigitalOutputs(self: pyaubo_sdk.IoControl) -> int

Lua 函数原型

getToolDigitalOutputs() -> number

17.56 getConfigurableDigitalOutput()

bool arcs::common_interface::IoControl::getConfigurableDigitalOutput (int *index*)

获取可配值数字输出

参数

index	表示IO 口的管脚，可用十进制或者十六进制表示。例如，0x00000000 表示第一个管脚

返回

高电平返回 true; 低电平返回 false

Python 函数原型

getConfigurableDigitalOutput(self: pyaubo_sdk.IoControl, arg0: int) -> bool

Lua 函数原型

getConfigurableDigitalOutput(index: number) -> boolean

17.57 getConfigurableDigitalOutputs()

uint32_t arcs::common_interface::IoControl::getConfigurableDigitalOutputs ( )

获取所有的可配值数字输出

返回

所有的可配值数字输出

Python 函数原型

getConfigurableDigitalOutputs(self: pyaubo_sdk.IoControl) -> int

Lua 函数原型

getConfigurableDigitalOutputs() -> number

17.58 getStandardAnalogInput()

double arcs::common_interface::IoControl::getStandardAnalogInput (
int *index*)

获取标准模拟输入

参数

index	表示IO 口的管脚，可用十进制或者十六进制表示。例如，0x00000000 表示第一个管脚

返回

标准模拟输入值

Python 函数原型

getStandardAnalogInput(self: pyaubo_sdk.IoControl, arg0: int) -> float

Lua 函数原型

getStandardAnalogInput(index: number) -> number

17.59 getToolAnalogInput()

double arcs::common_interface::IoControl::getToolAnalogInput (
int *index*)

获取工具端模拟输入

参数

index	表示IO 口的管脚，可用十进制或者十六进制表示。例如，0x00000000 表示第一个管脚

返回

工具端模拟输入

Python 函数原型

getToolAnalogInput(self: pyaubo_sdk.IoControl, arg0: int) -> float

Lua 函数原型

getToolAnalogInput(index: number) -> number

17.60 getStandardAnalogOutput()

double arcs::common_interface::IoControl::getStandardAnalogOutput (
int *index*)

获取标准模拟输出

参数

index	表示IO 口的管脚，可用十进制或者十六进制表示。例如，0x00000000 表示第一个管脚

返回

标准模拟输出

Python 函数原型

getStandardAnalogOutput(self: pyaubo_sdk.IoControl, arg0: int) -> float

Lua 函数原型

getStandardAnalogOutput(index: number) -> number

17.61 getToolAnalogOutput()

double arcs::common_interface::IoControl::getToolAnalogOutput (
int *index*)

获取工具端模拟输出

参数

index	表示IO 口的管脚，可用十进制或者十六进制表示。例如，0x00000000 表示第一个管脚

返回

工具端模拟输出

Python 函数原型

getToolAnalogOutput(self: pyaubo_sdk.IoControl, arg0: int) -> float

Lua 函数原型

getToolAnalogOutput(index: number) -> number

17.62 getStaticLinkInputNum()

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

获取联动输入数量

返回

联动输入数量

注解

兼容接口板，仅供读数显示

Python 函数原型

getStaticLinkInputNum(self: pyaubo_sdk.IoControl) -> int

Lua 函数原型

getStaticLinkInputNum() -> number

17.63 getStaticLinkOutputNum()

int arcs::common_interface::IoControl::getStaticLinkOutputNum ( )

获取联动输出数量

返回

联动输出数量

注解

兼容接口板，仅供读数显示

Python 函数原型

getStaticLinkOutputNum(self: pyaubo_sdk.IoControl) -> int

Lua 函数原型

getStaticLinkOutputNum() -> number

17.64 getStaticLinkInputs()

uint32_t arcs::common_interface::IoControl::getStaticLinkInputs ( )

获取所有的联动输入

返回

所有的联动输入

注解

兼容接口板，仅供读数显示

Python 函数原型

getStaticLinkInputs(self: pyaubo_sdk.IoControl) -> int

Lua 函数原型

getStaticLinkInputs() -> number

17.65 getStaticLinkOutputs()

uint32_t arcs::common_interface::IoControl::getStaticLinkOutputs ( )

获取所有的联动输出

返回

所有的联动输出

注解

兼容接口板，仅供读数显示

Python 函数原型

getStaticLinkOutputs(self: pyaubo_sdk.IoControl) -> int

Lua 函数原型

getStaticLinkOutputs() -> number

18. MotionControl 类

18.1 getEqradius()

double arcs::common_interface::MotionControl::getEqradius ( )

获取等效半径，单位 m

moveLine/moveCircle 时，末端姿态旋转的角度等效到末端位置移动

返回

等效半径

Python 函数原型

getEqradius(self: pyaubo_sdk.MotionControl) -> float

Lua 函数原型

getEqradius() -> number

18.2 setSpeedFraction()

int arcs::common_interface::MotionControl::setSpeedFraction (
double *fraction*)

动态调整机器人运行速度和加速度比例 (0., 1.]

参数

fraction	机器人运行速度和加速度比例

返回

Python 函数原型

setSpeedFraction(self: pyaubo_sdk.MotionControl, arg0: float) -> int

Lua 函数原型

setSpeedFraction(fraction: number) -> nil

18.3 getSpeedFraction()

double arcs::common_interface::MotionControl::getSpeedFraction ( )

获取速度和加速度比例，默认为 1 可以通过 setSpeedFraction 接口设置

返回

速度和加速度比例

Python 函数原型

getSpeedFraction(self: pyaubo_sdk.MotionControl) -> float

Lua 函数原型

getSpeedFraction() -> number

18.4 pathOffsetEnable()

int arcs::common_interface::MotionControl::pathOffsetEnable ( )

路径偏移使能

返回

Python 函数原型

pathOffsetEnable(self: pyaubo_sdk.MotionControl) -> int

Lua 函数原型

pathOffsetEnable() -> number

18.5 pathOffsetSet()

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

设置路径偏移

参数

offset	在各方向的位姿偏移
type	类型

返回

Python 函数原型

pathOffsetSet(self: pyaubo_sdk.MotionControl, arg0: List[float], arg1: int) -> int

Lua 函数原型

pathOffsetSet(offset: table, type: number) -> nil

18.6 pathOffsetDisable()

int arcs::common_interface::MotionControl::pathOffsetDisable ( )

路径偏移失能

返回

Python 函数原型

pathOffsetDisable(self: pyaubo_sdk.MotionControl) -> int

Lua 函数原型

pathOffsetDisable() -> nil

18.7 jointOffsetEnable()

int arcs::common_interface::MotionControl::jointOffsetEnable ( )

关节偏移使能

返回

Python 函数原型

jointOffsetEnable(self: pyaubo_sdk.MotionControl) -> int

Lua 函数原型

jointOffsetEnable() -> nil

18.8 jointOffsetSet()

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

设置关节偏移

参数

offset	在各关节的位姿偏移
type	类型

返回

Python 函数原型

jointOffsetSet(self: pyaubo_sdk.MotionControl, arg0: List[float], arg1: int) -> int

Lua 函数原型

jointOffsetSet(offset: table, type: number) -> nil

18.9 jointOffsetDisable()

int arcs::common_interface::MotionControl::jointOffsetDisable ( )

关节偏移失能

返回

Python 函数原型

jointOffsetDisable(self: pyaubo_sdk.MotionControl) -> int

Lua 函数原型

jointOffsetDisable() -> nil

18.10 getQueueSize()

int arcs::common_interface::MotionControl::getQueueSize ( )

获取已经入队的运动指令段数量

返回

已经入队的运动指令段数量

Python 函数原型

getQueueSize(self: pyaubo_sdk.MotionControl) -> int

Lua 函数原型

getQueueSize() -> number

18.11 getTrajectoryQueueSize()

int arcs::common_interface::MotionControl::getTrajectoryQueueSize ( )

获取已经入队的运动规划插补点数量

返回

已经入队的运动规划插补点数量

Python 函数原型

getTrajectoryQueueSize(self: pyaubo_sdk.MotionControl) -> int

Lua 函数原型

getTrajectoryQueueSize() -> number

18.12 getExecId()

int arcs::common_interface::MotionControl::getExecId ( )

获取当前正在插补的运动指令段的ID

返回

当前正在插补的运动指令段的ID

返回值

-1	表示轨迹队列为空 像 movePathBuffer 运动中的 buffer 或者规划器 (moveJoint 和 moveLine 等) 里的队列都属于轨迹队列

Python 函数原型

getExecId(self: pyaubo_sdk.MotionControl) -> int

Lua 函数原型

getExecId() -> number

18.13 getDuration()

double arcs::common_interface::MotionControl::getDuration (
int *id*)

获取指定ID 的运动指令段的预期执行时间

参数

id	运动指令段ID

返回

预期执行时间

Python 函数原型

getDuration(self: pyaubo_sdk.MotionControl, arg0: int) -> float

Lua 函数原型

getDuration(id: number) -> number

18.14 getMotionLeftTime()

double arcs::common_interface::MotionControl::getMotionLeftTime (
int *id*)

获取指定ID 的运动指令段的剩余执行时间

参数

id	运动指令段ID

返回

剩余执行时间

Python 函数原型

getMotionLeftTime(self: pyaubo_sdk.MotionControl, arg0: int) -> float

Lua 函数原型

getMotionLeftTime(id: number) -> number

18.15 getProgress()

double arcs::common_interface::MotionControl::getProgress ( )

获取当前运动指令段的执行进度

返回

执行进度

Python 函数原型

getProgress(self: pyaubo_sdk.MotionControl) -> float

Lua 函数原型

getProgress() -> number

18.16 setWorkObjectHold()

int arcs::common_interface::MotionControl::setWorkObjectHold (
const std::string & *module_name,*
const std::vector< double > & *mounting_pose*)

当工件安装在另外一台机器人的末端或者外部轴上时，指定其名字和安装位置

参数

module_name	控制模块名字
mounting_pose	抓取的相对位置，如果是机器人，则相对于机器人末端中心点(非TCP 点)

返回

Python 函数原型

setWorkObjectHold(self: pyaubo_sdk.MotionControl, arg0: str, arg1: List[float]) -> int

Lua 函数原型

setWorkObjectHold(module_name: string, mounting_pose: table) -> nil

18.17 getWorkObjectHold()

std::tuple< std::string, std::vector< double > > arcs::common_interface::MotionControl::getWorkObjectHold ( )

getWorkObjectHold

返回

Python 函数原型

getWorkObjectHold(self: pyaubo_sdk.MotionControl) -> Tuple[str, List[float]]

Lua 函数原型

getWorkObjectHold() -> table

18.18 getPauseJointPositions()

std::vector< double > arcs::common_interface::MotionControl::getPauseJoint↩ Positions ( )

getPauseJointPositions

注解

获取暂停点关节位置常用于运行工程时发生碰撞后继续运动过程中 (先通过 resumeMove↩ Joint 或 resumeMoveLine 运动到暂停位置, 再恢复工程)

返回

暂停关节位置

Python 函数原型

getPauseJointPositions(self: pyaubo_sdk.MotionControl) -> List[float]

Lua 函数原型

getPauseJointPositions() -> table

18.19 setServoMode()

int arcs::common_interface::MotionControl::setServoMode (
bool *enable*)

设置伺服模式

参数

enable	是否使能

返回

Python 函数原型

setServoMode(self: pyaubo_sdk.MotionControl, arg0: bool) -> int

Lua 函数原型

setServoMode(enable: boolean) -> nil

18.20 isServoModeEnabled()

bool arcs::common_interface::MotionControl::isServoModeEnabled ( )

判断伺服模式是否使能

返回

已使能返回 true，反之则返回 false

Python 函数原型

isServoModeEnabled(self: pyaubo_sdk.MotionControl) -> bool

Lua 函数原型

isServoModeEnabled() -> boolean

18.21 servoJoint()

int arcs::common_interface::MotionControl::servoJoint (
const std::vector< double > & *q*,
double *a*,
double *v*,
double *t*,
double *lookahead_time*,
double *gain*)

关节空间伺服

返回

Python 函数原型

servoJoint(self: pyaubo_sdk.MotionControl, arg0: List[float], arg1: float, arg2: float, arg3: float, arg4: float, arg5: float) -> int

Lua 函数原型

servoJoint(q: table, a: number, v: number, t: number, lookahead_time: number, gain: number) -> nil

18.22 followJoint()

int arcs::common_interface::MotionControl::followJoint (
const std::vector< double > & *q*)

关节空间跟随

返回

Python 函数原型

followJoint(self: pyaubo_sdk.MotionControl, arg0: List[float]) -> int

Lua 函数原型

followJoint(q: table) -> nil

18.23 followLine()

int arcs::common_interface::MotionControl::followLine (
const std::vector< double > & *pose*)

笛卡尔空间跟随

返回

Python 函数原型

followLine(self: pyaubo_sdk.MotionControl, arg0: List[float]) -> int

Lua 函数原型

followLine(pose: table) -> nil

18.24 speedJoint()

int arcs::common_interface::MotionControl::speedJoint (
const std::vector< double > & *qd,*
double *a,*
double *t*)

关节空间速度跟随

当机械臂还没达到目标速度的时候，给一个新的目标速度，机械臂会立刻达到新的目标速度

参数

q	目标关节速度, 单位 rad/s
a	主轴的加速度, 单位 rad/s^2
t	函数返回所需要的时间, 单位 s 如果 t = 0，当达到目标速度的时候，函数将返回；反之，则经过 t 时间后，函数返回，不管是否达到目标速度。 如果没有达到目标速度，会减速到零。如果达到了目标速度就是按照目标速度匀速运动。

返回

Python 函数原型

speedJoint(self: pyaubo_sdk.MotionControl, arg0: List[float], arg1: float, arg2: float) -> int

Lua 函数原型

speedJoint(q: table, a: number, t: number) -> nil

18.25 resumeSpeedJoint()

int arcs::common_interface::MotionControl::resumeSpeedJoint (
const std::vector< double > & *qd,*
double *a,*
double *t*)

关节空间速度跟随 (机械臂运行工程时发生碰撞, 通过此接口移动到安全位置)

当机械臂还没达到目标速度的时候，给一个新的目标速度，机械臂会立刻达到新的目标速度

参数

qd	目标关节速度, 单位 rad/s
a	主轴的加速度, 单位 rad/s^2
t	函数返回所需要的时间, 单位 s 如果 t = 0，当达到目标速度的时候，函数将返回；反之，则经过 t 时间后，函数返回，不管是否达到目标速度。如果没有达到目标速度，会减速到零。如果达到了目标速度就是按照目标速度匀速运动。

返回

Python 函数原型

resumeSpeedJoint(self: pyaubo_sdk.MotionControl, arg0: List[float], arg1: float, arg2: float) -> int

Lua 函数原型

resumeSpeedJoint(q: table, a: number, t: number) -> nil

18.26 speedLine()

int arcs::common_interface::MotionControl::speedLine (
const std::vector< double > & *xd,*
double *a,*
double *t*)

笛卡尔空间速度跟随

当机械臂还没达到目标速度的时候，给一个新的目标速度，机械臂会立刻达到新的目标速度

参数

xd	工具速度, 单位 m/s
a	工具位置加速度, 单位 m/s^2
t	函数返回所需要的时间, 单位 s 如果 t = 0，当达到目标速度的时候，函数将返回；反之，则经过 t 时间后，函数返回，不管是否达到目标速度。如果没有达到目标速度，会减速到零。如果达到了目标速度就是按照目标速度匀速运动。

返回

Python 函数原型

speedLine(self: pyaubo_sdk.MotionControl, arg0: List[float], arg1: float, arg2: float) -> int

Lua 函数原型

speedLine(pose: table, a: number, t: number) -> nil

18.27 resumeSpeedLine()

int arcs::common_interface::MotionControl::resumeSpeedLine (
const std::vector< double > & *xd,*
double *a,*
double *t*)

笛卡尔空间速度跟随 (机械臂运行工程时发生碰撞, 通过此接口移动到安全位置)

当机械臂还没达到目标速度的时候，给一个新的目标速度，机械臂会立刻达到新的目标速度

参数

xd	工具速度, 单位 m/s
a	工具位置加速度, 单位 m/s^2
t	函数返回所需要的时间, 单位 s 如果 t = 0，当达到目标速度的时候，函数将返回；反之，则经过 t 时间后，函数返回，不管是否达到目标速度。如果没有达到目标速度，会减速到零。如果达到了目标速度就是按照目标速度匀速运动。

返回

Python 函数原型

resumeSpeedLine(self: pyaubo_sdk.MotionControl, arg0: List[float], arg1: float, arg2: float) -> int

Lua 函数原型

resumeSpeedLine(pose: table, a: number, t: number) -> nil

18.28 moveSpline()

int arcs::common_interface::MotionControl::moveSpline (
const std::vector< double > & *q*,
double *a*,
double *v*,
double *duration*)

在关节空间做样条插值

参数

q	关节角度，如果传入参数维度为 0，表示样条运动结束
a	加速度, 单位 rad/s^2, 最大值可通过RobotConfig 类中的接口 getJointMaxAccelerations() 来获取
v	速度，单位 rad/s, 最大值可通过RobotConfig 类中的接口 getJointMaxSpeeds() 来获取
duration	运行时间，单位 s 通常当给定了速度和加速度，便能够确定轨迹的运行时间。如果想延长轨迹的运行时间，便要设置 duration 这个参数。duration 可以延长轨迹运动的时间，但是不能缩短轨迹时间。 当 duration = 0 的时候，表示不指定运行时间，即没有明确完成运动的时间，将根据速度与加速度计算运行时间。如果 duration 不等于 0，a 和 v 的值将被忽略。

返回

Python 函数原型

moveSpline(self: pyaubo_sdk.MotionControl, arg0: List[float], arg1: float, arg2: float, arg3: float) -> int

Lua 函数原型

moveSpline(q: table, a: number, v: number, duration: number) -> nil

18.29 moveJoint()

int arcs::common_interface::MotionControl::moveJoint (
const std::vector< double > & *q*,
double *a*,
double *v*,
double *blend_radius*,
double *duration*)

添加关节运动

参数

q	关节角, 单位 rad
a	加速度, 单位 rad/s^2, 最大值可通过RobotConfig 类中的接口 getJointMaxAccelerations() 来获取
v	速度，单位 rad/s, 最大值可通过RobotConfig 类中的接口 getJointMaxSpeeds() 来获取
blend_radius	交融半径, 单位 m
duration	运行时间，单位 s 通常当给定了速度和加速度，便能够确定轨迹的运行时间。如果想延长轨迹的运行时间，便要设置 duration 这个参数。duration 可以延长轨迹运动的时间，但是不能缩短轨迹时间。 当 duration = 0 的时候，表示不指定运行时间，即没有明确完成运动的时间，将根据速度与加速度计算运行时间。如果 duration 不等于 0，a 和 v 的值将被忽略。

返回

Python 函数原型

moveJoint(self: pyaubo_sdk.MotionControl, arg0: List[float], arg1: float, arg2: float, arg3: float, arg4: float) -> int

Lua 函数原型

moveJoint(q: table, a: number, v: number, blend_radius: number, duration: number) -> nil

18.30 resumeMoveJoint()

int arcs::common_interface::MotionControl::resumeMoveJoint (
const std::vector< double > & *q*,
double *a*,
double *v*,
double *duration*)

通过关节运动移动到暂停点的位置

参数

q	关节角, 单位 rad
a	加速度, 单位 rad/s^2
v	速度，单位 rad/s
duration	运行时间，单位 s 通常当给定了速度和加速度，便能够确定轨迹的运行时间。如果想延长轨迹的运行时间，便要设置 duration 这个参数。duration 可以延长轨迹运动的时间，但是不能缩短轨迹时间。 当 duration = 0 的时候，表示不指定运行时间，即没有明确完成运动的时间，将根据速度与加速度计算运行时间。如果 duration 不等于 0，a 和 v 的值将被忽略。

返回

Python 函数原型

resumeMoveJoint(self: pyaubo_sdk.MotionControl, arg0: List[float], arg1: float, arg2: float, arg3: float) -> int

Lua 函数原型

resumeMoveJoint(q: table, a: number, v: number, duration: number) -> nil

18.31 moveLine()

int arcs::common_interface::MotionControl::moveLine (
const std::vector< double > & *pose*,
double *a*,

double *v,*
double *blend_radius,*
double *duration*)

添加直线运动

参数

pose	目标位姿
a	加速度, 单位 m/s^2，最大值可通过RobotConfig 类中的接口 getTcpMaxAccelerations() 来获取
v	速度, 单位 m/s，最大值可通过RobotConfig 类中的接口 getTcpMaxSpeeds() 来获取
blend_radius	交融半径，单位 m，值介于 0.001 和 1 之间
duration	运行时间，单位 s 通常当给定了速度和加速度，便能够确定轨迹的运行时间。如果想延长轨迹的运行时间，便要设置 duration 这个参数。duration 可以延长轨迹运动的时间，但是不能缩短轨迹时间。 当 duration = 0 的时候，表示不指定运行时间，即没有明确完成运动的时间，将根据速度与加速度计算运行时间。如果 duration 不等于 0，a 和 v 的值将被忽略。

返回

Python 函数原型

moveLine(self: pyaubo_sdk.MotionControl, arg0: List[float], arg1: float, arg2: float, arg3: float, arg4: float) -> int

Lua 函数原型

moveLine(pose: table, a: number, v: number, blend_radius: number, duration: number) -> nil

18.32 resumeMoveLine()

int arcs::common_interface::MotionControl::resumeMoveLine (
const std::vector< double > & *pose,*
double *a,*
double *v,*
double *duration*)

通过直线运动移动到暂停点的位置

参数

pose	目标位姿
a	加速度, 单位 m/s^2
v	速度, 单位 m/s
duration	运行时间，单位 s 通常当给定了速度和加速度，便能够确定轨迹的运行时间。如果想延长轨迹的运行时间，便要设置 duration 这个参数。duration 可以延长轨迹运动的时间，但是不能缩短轨迹时间。 当 duration = 0 的时候，表示不指定运行时间，即没有明确完成运动的时间，将根据速度与加速度计算运行时间。如果 duration 不等于 0，a 和 v 的值将被忽略。

返回

Python 函数原型

resumeMoveLine(self: pyaubo_sdk.MotionControl, arg0: List[float], arg1: float, arg2: float, arg3: float) -> int

Lua 函数原型

resumeMoveLine(pose: table, a: number, v: number,duration: number) -> nil

18.33 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*)

添加圆弧运动

参数

via_pose	圆弧运动途中点的位姿
end_pose	圆弧运动结束点的位姿

a	加速度, 单位: m/s^2
v	速度，单位: m/s
blend_radius	交融半径，单位: m
duration	运行时间，单位: s 通常当给定了速度和加速度，便能够确定轨迹的运行时间。如果想延长轨迹的运行时间，便要设置 duration 这个参数。duration 可以延长轨迹运动的时间，但是不能缩短轨迹时间。 当 duration = 0 的时候，表示不指定运行时间，即没有明确完成运动的时间，将根据速度与加速度计算运行时间。如果 duration 不等于 0，a 和 v 的值将被忽略。

返回

Python 函数原型

moveCircle(self: pyaubo_sdk.MotionControl, arg0: List[float], arg1: List[float], arg2: float, arg3: float, arg4: float, arg5: float) -> int

Lua 函数原型

moveCircle(via_pose: table, end_pose: table, a: number, v: number, blend_radius: number, duration: number) -> nil

18.34 setCirclePathMode()

int arcs::common_interface::MotionControl::setCirclePathMode (
int *mode*)

设置圆弧路径模式

参数

mode	模式 0: 工具姿态相对于圆弧路径点坐标系保持不变 1: 姿态线性变化, 绕着空间定轴转动, 从起始点姿态变化到目标点姿态 2: 从起点姿态开始经过中间点姿态, 变化到目标点姿态

返回

Python 函数原型

setCirclePathMode(self: pyaubo_sdk.MotionControl, arg0: int) -> int

Lua 函数原型

setCirclePathMode(mode: number) -> nil

18.35 moveCircle2()

int arcs::common_interface::MotionControl::moveCircle2 (
const CircleParameters & *param*)

高级圆弧或者圆周运动

参数

param	圆周运动参数

返回

Python 函数原型

moveCircle2(self: pyaubo_sdk.MotionControl, arg0: arcs::common_interface::CircleParameters) -> int

Lua 函数原型

moveCircle2(param: table) -> nil

18.36 pathBufferAlloc()

int arcs::common_interface::MotionControl::pathBufferAlloc (
const std::string & *name,*
int *type,*
int *size*)

新建一个路径点缓存

参数

name	指定路径的名字
type	路径的类型 1: toppra 时间最优路径规划 2: cubic_spline(录制的轨迹) 3: 关节B 样条插值，最少三个点
size	缓存区大小

返回

新建成功返回 AUBO_OK(0)

Python 函数原型

pathBufferAlloc(self: pyaubo_sdk.MotionControl, arg0: str, arg1: int, arg2: int) -> int

Lua 函数原型

pathBufferAlloc(name: string, type: number, size: number) -> nil

18.37 pathBufferAppend()

int arcs::common_interface::MotionControl::pathBufferAppend (
const std::string & *name,*
const std::vector< std::vector< double > > & *waypoints*)

向路径缓存添加路点

参数

name	路径缓存的名字
waypoints	路点

返回

Python 函数原型

pathBufferAppend(self: pyaubo_sdk.MotionControl, arg0: str, arg1: List[List[float]]) -> int

Lua 函数原型

pathBufferAppend(name: string, waypoints: table) -> nil

18.38 pathBufferEval()

int arcs::common_interface::MotionControl::pathBufferEval (
const std::string & *name,*
const std::vector< double > & *a,*
const std::vector< double > & *v,*
double *t*)

计算、优化等耗时操作，传入的参数相同时不会重新计算

参数

name	通过 pathBufferAlloc 新建的路径点缓存的名字
a	关节加速度限制
v	关节速度限制
t	时间 pathBufferAlloc 这个接口分配内存的时候要指定类型，根据 pathBufferAlloc 这个接口的类型: 类型为 1 时, 表示运动持续时间 类型为 2 时, 表示采样时间间隔 类型为 3 时,t 参数设置为 0

返回

Python 函数原型

pathBufferEval(self: pyaubo_sdk.MotionControl, arg0: str, arg1: List[float], arg2↩
: List[float], arg3: float) -> int

Lua 函数原型

pathBufferEval(name: string, a: table, v: table, t: number) -> nil

18.39 pathBufferValid()

bool arcs::common_interface::MotionControl::pathBufferValid (
const std::string & *name*)

指定名字的 buffer 是否有效

buffer 需要满足三个条件有效:
1、buffer 存在，已经分配过内存
2、传进 buffer 的点要大于等于 buffer 的大小
3、要执行一次 pathBufferEval

参数

name	buffer 的名字

返回

有效返回 true; 无效返回 false

Python 函数原型

pathBufferValid(self: pyaubo_sdk.MotionControl, arg0: str) -> bool

Lua 函数原型

pathBufferValid(name: string) -> boolean

18.40 pathBufferFree()

int arcs::common_interface::MotionControl::pathBufferFree (
const std::string & *name*)

释放路径缓存

参数

name	缓存路径的名字

返回

Python 函数原型

pathBufferFree(self: pyaubo_sdk.MotionControl, arg0: str) -> int

Lua 函数原型

pathBufferFree(name: string) -> nil

18.41 pathBufferList()

std::vector< std::string > arcs::common_interface::MotionControl::pathBufferList ( )

列出所有缓存路径的名字

返回

返回所有缓存路径的名字

Python 函数原型

pathBufferList(self: pyaubo_sdk.MotionControl) -> List[str]

Lua 函数原型

pathBufferList() -> table

18.42 movePathBuffer()

int arcs::common_interface::MotionControl::movePathBuffer (
const std::string & *name*)

执行缓存的路径

参数

name	缓存路径的名字

返回

Python 函数原型

movePathBuffer(self: pyaubo_sdk.MotionControl, arg0: str) -> int

Lua 函数原型

movePathBuffer(name: string) -> nil

18.43 stopJoint()

int arcs::common_interface::MotionControl::stopJoint (
double *acc*)

关节空间停止运动

参数

acc	关节加速度，单位: rad/s^2

返回

Python 函数原型

stopJoint(self: pyaubo_sdk.MotionControl, arg0: float) -> int

Lua 函数原型

stopJoint(acc: number) -> nil

18.44 resumeStopJoint()

int arcs::common_interface::MotionControl::resumeStopJoint (
double *acc*)

关节空间停止运动 (机械臂运行工程时发生碰撞, 通过 resumeSpeedJoint 接口移动到安全位置后需要停止时调用此接口)

参数

acc	关节加速度，单位: rad/s^2

返回

Python 函数原型

resumeStopJoint(self: pyaubo_sdk.MotionControl, arg0: float) -> int

Lua 函数原型

resumeStopJoint(acc: number) -> nil

18.45 stopLine()

int arcs::common_interface::MotionControl::stopLine (
double *acc,*
double *acc_rot*)

笛卡尔空间停止运动

参数

acc	工具加速度, 单位: m/s^2
acc_rot	

返回

Python 函数原型

stopLine(self: pyaubo_sdk.MotionControl, arg0: float, arg1: float) -> int

Lua 函数原型

stopLine(acc: number, acc_rot: number) -> nil

18.46 resumeStopLine()

int arcs::common_interface::MotionControl::resumeStopLine (
double *acc,*
double *acc_rot*)

笛卡尔空间停止运动 (机械臂运行工程时发生碰撞, 通过 resumeSpeedLine 接口移动到安全位置后需要停止时调用此接口)

参数

acc	位置加速度, 单位: m/s^2
acc_rot	姿态加速度，单位: rad/s^2

返回

Python 函数原型

resumeStopLine(self: pyaubo_sdk.MotionControl, arg0: float, arg1: float) -> int

Lua 函数原型

resumeStopLine(acc: number, acc_rot: number) -> nil

18.47 weaveStart()

int arcs::common_interface::MotionControl::weaveStart (
const std::string & *params*)

开始摆动: weaveStart 和 weaveEnd 的 moveLine/moveCircle 将根据 params 进行摆动

参数

params	Json 字符串用于定义摆动参数 { ”type”: <string>, // ”SINE” ”SPIRAL” ”TRIANGLE” ”TRAPEZOIDAL” ”step”: <num>, ”amplitude”: {<num>,<num>}, ”hold_distance”: {<num>,<num>}, ”angle”: <num> ”direction”: <num> }

返回

18.48 weaveEnd()

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

结束摆动

返回

19. OutputBuilder 类

19.1 arcs::aubo_sdk::OutputBuilder 类参考

Public 成员函数

- OutputBuilder & push (int val)
- OutputBuilder & push (double val)
- OutputBuilder & push (const std::vector< double > &val)
- OutputBuilder & push (const std::vector< int > &val)
- OutputBuilder & push (const std::string &val)
- OutputBuilder & push (char val)
- OutputBuilder & push (const common_interface::RtdeRecipe &val)

20. RobotAlgorithm 类

20.1 calibrateTcpForceSensor()

ForceSensorCalibResult arcs::common_interface::RobotAlgorithm::calibrate↩ TcpForceSensor (
const std::vector< std::vector< double > > & *forces,*
const std::vector< std::vector< double > > & *poses*)

力传感器标定算法 (三点标定法)

参数

force	大小为 6 的集合。前三个数是力，后三个数是力矩
q	TCP 位姿

返回

标定结果

Python 函数原型

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

Lua 函数原型

calibrateTcpForceSensor(force: table, q: table) -> table

20.2 payloadIdentify()

ForceSensorCalibResult arcs::common_interface::RobotAlgorithm::payload↩ Identify (
const std::vector< std::vector< double > > & *q,*
const std::vector< std::vector< double > > & *forces*)

负载辨识算法接口

输入关节角度和 (工具端) 力矩传感器读数，输出质量，质心偏移等等

参数

q	关节角度集合
forces	对应的末端力/力矩集合

返回

辨识的结果

Python 函数原型

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

Lua 函数原型

payloadIdentify(q: table, forces: table) -> table

20.3 calibWorkpieceCoordinatePara()

ResultWithErrno arcs::common_interface::RobotAlgorithm::calibWorkpieceCoordinatePara (
const std::vector< std::vector< double > > & *q*,
int *type*)

工件坐标系标定算法接口 (需要在调用之前正确的设置机器人的TCP 偏移) 输入多组关节角度和标定类型，输出工件坐标系位姿 (相对于机器人基坐标系)

参数

q	关节角度
type	标定类型

返回

计算结果 (工件坐标系位姿) 以及错误代码

Python 函数原型

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

Lua 函数原型

calibWorkpieceCoordinatePara(q: table, type: number) -> table

20.4 forwardDynamics()

ResultWithErrno arcs::common_interface::RobotAlgorithm::forwardDynamics (
const std::vector< double > & *q,*
const std::vector< double > & *torqs*)

动力学正解

参数

q	关节角度
torqs	力矩

返回

计算结果以及错误码

Python 函数原型

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

Lua 函数原型

forwardDynamics(q: table, torqs: table) -> number

20.5 forwardKinematics()

ResultWithErrno arcs::common_interface::RobotAlgorithm::forwardKinematics (

const std::vector< double > & *q*)

运动学正解输入关节角度，输出TCP 位姿

参数

q	关节角度

返回

计算结果 (TCP 位姿) 以及错误码

Python 函数原型

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

Lua 函数原型

forwardKinematics(q: table) -> table

20.6 forwardToolKinematics()

ResultWithErrno arcs::common_interface::RobotAlgorithm::forwardTool↩ Kinematics (

const std::vector< double > & *q*)

运动学正解 (忽略 TCP 偏移值)

参数

q	关节角度

返回

计算结果 (法兰盘中心的位姿) 以及错误码

20.7 inverseKinematics()

ResultWithErrno arcs::common_interface::RobotAlgorithm::inverseKinematics (

const std::vector< double > & *qnear,*
const std::vector< double > & *pose*)

运动学逆解输入TCP 位姿和参考关节角度，输出关节角度

参数

qnear	参考关节角度
pose	TCP 位姿

返回

计算结果（关节角）以及错误码

Python 函数原型

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

Lua 函数原型

inverseKinematics(qnear: table, pose: table) -> table

20.8 inverseKinematicsAll()

ResultWithErrno1 arcs::common_interface::RobotAlgorithm::inverseKinematics↩ All (

const std::vector< double > & *pose*)

求出所有的逆解

参数

pose	TCP 位姿

返回

计算结果（关节角）以及错误码

20.9 inverseToolKinematics()

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

运动学逆解 (忽略 TCP 偏移值)

参数

qnear	参考关节角
pose	法兰盘中心的位姿

返回

计算结果（关节角）以及错误码

20.10 inverseToolKinematicsAll()

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

运动学逆解

参数

qnear	参考关节角
pose	法兰盘中心的位姿

返回

计算结果（关节角）以及错误码

20.11 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*)

求解 movej 之间的轨迹点

参数

q1	movej 的起点
r1	在 q1 处的交融半径
q2	movej 的终点
r2	在 q2 处的交融半径
d	采样距离

返回

q1～q2 之间不包含交融段的笛卡尔空间离散轨迹点 (x,y,z) 集合

Python 函数原型

pathMovej(self: pyaubo_sdk.RobotAlgorithm, arg0: List[float], arg1: float, arg2: List[float], arg3: float, arg4: float) -> List[List[float]]

Lua 函数原型

pathMovej(q1: table, r1: number, q2: table, r2: number, d: number) -> table

20.12 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*)

求解交融的轨迹点

参数

type	0-movej and movej 1-movej and movel 2-movel and movej 2-movel and movel
q_start	交融前路径的起点
q_via	在 q1 处的交融半径
q_to	交融后路径的终点
r	在 q_via 处的交融半径
d	采样距离

返回

q_via 处的交融段笛卡尔空间离散轨迹点 (x,y,z) 集合

Python 函数原型

pathBlend3Points(self: pyaubo_sdk.RobotAlgorithm, arg0: int, arg1: List[float], arg2↩ : List[float], arg3: List[float], arg4: float, arg5: float) -> List[List[float]]

Lua 函数原型

pathBlend3Points(type: number, q_start: table, q_via: table, q_to: table, r: number, d: number) -> table

21. RobotConfig 类

21.1 getName()

std::string arcs::common_interface::RobotConfig::getName ( )

获取机器人的名字

返回

机器人的名字

Python 函数原型

getName(self: pyaubo_sdk.RobotConfig) -> str

Lua 函数原型

getName() -> string

21.2 getDof()

int arcs::common_interface::RobotConfig::getDof ( )

获取机器人的自由度 (从硬件抽象层读取)

返回

机器人的自由度

Python 函数原型

getDof(self: pyaubo_sdk.RobotConfig) -> int

Lua 函数原型

getDof() -> number

21.3 getCycletime()

double arcs::common_interface::RobotConfig::getCycletime ( )

获取机器人的伺服控制周期 (从硬件抽象层读取)

返回

机器人的伺服控制周期

Python 函数原型

getCycletime(self: pyaubo_sdk.RobotConfig) -> float

Lua 函数原型

getCycletime() -> number

21.4 setSlowDownFraction()

int arcs::common_interface::RobotConfig::setSlowDownFraction (
int *level,*
double *fraction*)

预设缓速模式下的速度缩减比例

参数

level	缓速等级 1, 2
fraction	缩减比例

返回

21.5 getSlowDownFraction()

double arcs::common_interface::RobotConfig::getSlowDownFraction (
int *level*)

获取预设的缓速模式下的速度缩减比例

参数

level	缓速等级 1, 2

返回

预设的缓速模式下的速度缩减比例

21.6 getDefaultToolAcc()

double arcs::common_interface::RobotConfig::getDefaultToolAcc ( )

获取默认的工具端加速度，单位 m/s^2

返回

默认的工具端加速度

Python 函数原型

getDefaultToolAcc(self: pyaubo_sdk.RobotConfig) -> float

Lua 函数原型

getDefaultToolAcc() -> number

21.7 getDefaultToolSpeed()

double arcs::common_interface::RobotConfig::getDefaultToolSpeed ( )

获取默认的工具端速度，单位 m/s

返回

默认的工具端速度

Python 函数原型

getDefaultToolSpeed(self: pyaubo_sdk.RobotConfig) -> float

Lua 函数原型

getDefaultToolSpeed() -> number

21.8 getDefaultJointAcc()

double arcs::common_interface::RobotConfig::getDefaultJointAcc ( )

获取默认的关节加速度，单位 rad/s^2

返回

默认的关节加速度

Python 函数原型

getDefaultJointAcc(self: pyaubo_sdk.RobotConfig) -> float

Lua 函数原型

getDefaultJointAcc() -> number

21.9 getDefaultJointSpeed()

double arcs::common_interface::RobotConfig::getDefaultJointSpeed ( )

获取默认的关节速度，单位 rad/s

返回

默认的关节速度

Python 函数原型

getDefaultJointSpeed(self: pyaubo_sdk.RobotConfig) -> float

Lua 函数原型

getDefaultJointSpeed() -> number

21.10 getRobotType()

std::string arcs::common_interface::RobotConfig::getRobotType ( )

获取机器人类型代码

返回

机器人类型代码

Python 函数原型

getRobotType(self: pyaubo_sdk.RobotConfig) -> str

Lua 函数原型

getRobotType() -> string

21.11 getRobotSubType()

std::string arcs::common_interface::RobotConfig::getRobotSubType ( )

获取机器人子类型代码

返回

机器人子类型代码

Python 函数原型

getRobotSubType(self: pyaubo_sdk.RobotConfig) -> str

Lua 函数原型

getRobotSubType() -> string

21.12 getControlBoxType()

std::string arcs::common_interface::RobotConfig::getControlBoxType ( )

获取控制柜类型代码

返回

控制柜类型代码

Python 函数原型

getControlBoxType(self: pyaubo_sdk.RobotConfig) -> str

Lua 函数原型

getControlBoxType() -> string

21.13 setMountingPose()

int arcs::common_interface::RobotConfig::setMountingPose (
const std::vector< double > & *pose*)

设置安装位姿 (机器人的基坐标系相对于世界坐标系) world->base

一般在多机器人系统中使用，默认为 [0,0,0,0,0,0]

参数

pose	安装位姿

返回

Python 函数原型

setMountingPose(self: pyaubo_sdk.RobotConfig, arg0: List[float]) -> int

Lua 函数原型

setMountingPose(pose: table) -> nil

21.14 getMountingPose()

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

获取安装位姿 (机器人的基坐标系相对于世界坐标系)

返回

安装位姿

Python 函数原型

getMountingPose(self: pyaubo_sdk.RobotConfig) -> List[float]

Lua 函数原型

getMountingPose() -> table

21.15 setCollisionLevel()

int arcs::common_interface::RobotConfig::setCollisionLevel (int *level*)

设置碰撞灵敏度等级数值越大越灵敏

参数

level	碰撞灵敏度等级 0: 关闭碰撞检测功能 1~9: 碰撞灵敏等级

返回

Python 函数原型

setCollisionLevel(self: pyaubo_sdk.RobotConfig, arg0: int) -> int

Lua 函数原型

setCollisionLevel(level: number) -> nil

21.16 getCollisionLevel()

int arcs::common_interface::RobotConfig::getCollisionLevel ( )

获取碰撞灵敏度等级

返回

碰撞灵敏度等级

Python 函数原型

getCollisionLevel(self: pyaubo_sdk.RobotConfig) -> int

Lua 函数原型

getCollisionLevel() -> number

21.17 setCollisionStopType()

int arcs::common_interface::RobotConfig::setCollisionStopType (
int *type*)

设置碰撞停止类型

参数

type	类型 0: 碰撞停机 1: 碰撞之后进入拖动模式

返回

Python 函数原型

setCollisionStopType(self: pyaubo_sdk.RobotConfig, arg0: int) -> int

Lua 函数原型

setCollisionStopType(type: number) -> nil

21.18 getCollisionStopType()

int arcs::common_interface::RobotConfig::getCollisionStopType ( )

获取碰撞停止类型

返回

返回碰撞停止类型

Python 函数原型

getCollisionStopType(self: pyaubo_sdk.RobotConfig) -> int

Lua 函数原型

getCollisionStopType() -> number

21.19 setHomePosition()

int arcs::common_interface::RobotConfig::setHomePosition (
const std::vector< double > & *positions*)

设置机器人的 Home 位置

参数

positions	关节角度

返回

21.20 getHomePosition()

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

获取机器人 Home 位置

返回

机器人 Home 位置

21.21 setFreedriveDamp()

int arcs::common_interface::RobotConfig::setFreedriveDamp (
const std::vector< double > & *damp*)

设置拖动阻尼

参数

damp	阻尼

返回

成功 0，失败-1

Python 函数原型

setFreedriveDamp(self: pyaubo_sdk.RobotConfig, arg0: List[float]) -> int

Lua 函数原型

setFreedriveDamp(damp: table) -> number

21.22 getFreedriveDamp()

std::vector< double > arcs::common_interface::RobotConfig::getFreedrive↩
Damp ( )

获取拖动阻尼

返回

拖动阻尼

Python 函数原型

getFreedriveDamp(self: pyaubo_sdk.RobotConfig) -> List[float]

Lua 函数原型

getFreedriveDamp() -> table

21.23 getKinematicsParam()

std::unordered_map< std::string, std::vector< double > > arcs::common_interface::RobotConfig::getKinematicsParam (
bool *real*)

获取机器人DH 参数 alpha a d theta beta

参数

real	读取真实参数 (理论值 + 补偿值) 或者理论参数

返回

返回机器人DH 参数

Python 函数原型

getKinematicsParam(self: pyaubo_sdk.RobotConfig, arg0: bool) -> Dict[str, List[float]]

Lua 函数原型

getKinematicsParam(real: boolean) -> table

21.24 getKinematicsCompensate()

std::unordered_map< std::string, std::vector< double > > arcs::common_interface::RobotConfig::getKinematicsCompensate (
double *ref_temperature*)

获取指定温度下的DH 参数补偿值:alpha a d theta beta

参数

ref_temperature	参考温度 ℃，默认 20℃

返回

DH 参数补偿值

Python 函数原型

getKinematicsCompensate(self: pyaubo_sdk.RobotConfig, arg0: float) -> Dict[str, List[float]]

Lua 函数原型

getKinematicsCompensate(ref_temperature: number) -> table

21.25 setKinematicsCompensate()

int arcs::common_interface::RobotConfig::setKinematicsCompensate (
const std::unordered_map< std::string, std::vector< double > > & *param*)

设置标准 DH 补偿到机器人

参数

param	

返回

21.26 setPersistentParameters()

int arcs::common_interface::RobotConfig::setPersistentParameters (
const std::string & *param*)

设置需要保存到接口板底座的参数

参数

param	补偿数据

返回

Python 函数原型

setPersistentParameters(self: pyaubo_sdk.RobotConfig, arg0: str) -> int

Lua 函数原型

setPersistentParameters(param: string) -> nil

21.27 setHardwareCustomParameters()

int arcs::common_interface::RobotConfig::setHardwareCustomParameters (
const std::string & *param*)

设置硬件抽象层自定义参数

目的是为了做不同硬件之间的兼容

参数

param	自定义参数

返回

21.28 getHardwareCustomParameters()

std::string arcs::common_interface::RobotConfig::getHardwareCustomParameters (
const std::string & *param*)

获取硬件抽象层自定义参数

参数

param	自定义参数

返回

参数

21.29 setRobotZero()

int arcs::common_interface::RobotConfig::setRobotZero ( )

设置机器人关节零位

返回

Python 函数原型

setRobotZero(self: pyaubo_sdk.RobotConfig) -> int

Lua 函数原型

setRobotZero() -> nil

21.30 getTcpForceSensorNames()

std::vector< std::string > arcs::common_interface::RobotConfig::getTcpForce↩ SensorNames ( )

获取可用的末端力矩传感器的名字

返回

可用的末端力矩传感器的名字

Python 函数原型

getTcpForceSensorNames(self: pyaubo_sdk.RobotConfig) -> List[str]

Lua 函数原型

getTcpForceSensorNames() -> table

21.31 selectTcpForceSensor()

int arcs::common_interface::RobotConfig::selectTcpForceSensor (
const std::string & *name*)

设置末端力矩传感器如果存在内置的末端力矩传感器，默认将使用内置的力矩传感器

参数

name	末端力矩传感器的名字

返回

Python 函数原型

selectTcpForceSensor(self: pyaubo_sdk.RobotConfig, arg0: str) -> int

Lua 函数原型

selectTcpForceSensor(name: string) -> nil

21.32 setTcpForceSensorPose()

int arcs::common_interface::RobotConfig::setTcpForceSensorPose (
const std::vector< double > & *sensor_pose*)

设置传感器安装位姿

参数

sensor_pose	传感器安装位姿

21.33 getTcpForceSensorPose()

std::vector< double > arcs::common_interface::RobotConfig::getTcpForce↩ SensorPose ( )

获取传感器安装位姿

返回

传感器安装位姿

21.34 hasTcpForceSensor()

bool arcs::common_interface::RobotConfig::hasTcpForceSensor ( )

是否安装了末端力矩传感器

返回

安装返回 true; 没有安装返回 false

Python 函数原型

hasTcpForceSensor(self: pyaubo_sdk.RobotConfig) -> bool

Lua 函数原型

hasTcpForceSensor() -> boolean

21.35 setTcpForceOffset()

int arcs::common_interface::RobotConfig::setTcpForceOffset (
const std::vector< double > & *force_offset*)

设置末端力矩偏移

参数

force_offset	末端力矩偏移

返回

Python 函数原型

setTcpForceOffset(self: pyaubo_sdk.RobotConfig, arg0: List[float]) -> int

Lua 函数原型

setTcpForceOffset(force_offset: table) -> nil

21.36 getTcpForceOffset()

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

获取末端力矩偏移

返回

末端力矩偏移

Python 函数原型

getTcpForceOffset(self: pyaubo_sdk.RobotConfig) -> List[float]

Lua 函数原型

getTcpForceOffset() -> table

21.37 getBaseForceSensorNames()

std::vector< std::string > arcs::common_interface::RobotConfig::getBase↩ ForceSensorNames ( )

获取可用的底座力矩传感器的名字

返回

可用的底座力矩传感器的名字

Python 函数原型

getBaseForceSensorNames(self: pyaubo_sdk.RobotConfig) -> List[str]

Lua 函数原型

getBaseForceSensorNames() -> table

21.38 selectBaseForceSensor()

int arcs::common_interface::RobotConfig::selectBaseForceSensor (
const std::string & *name*)

设置底座力矩传感器如果存在内置的底座力矩传感器，默认将使用内置的力矩传感器

参数

name	底座力矩传感器的名字

返回

Python 函数原型

selectBaseForceSensor(self: pyaubo_sdk.RobotConfig, arg0: str) -> int

Lua 函数原型

selectBaseForceSensor(name: string) -> nil

21.39 hasBaseForceSensor()

bool arcs::common_interface::RobotConfig::hasBaseForceSensor ( )

是否安装了底座力矩传感器

返回

安装返回 true; 没有安装返回 false

Python 函数原型

hasBaseForceSensor(self: pyaubo_sdk.RobotConfig) -> bool

Lua 函数原型

hasBaseForceSensor() -> boolean

21.40 setBaseForceOffset()

int arcs::common_interface::RobotConfig::setBaseForceOffset (
const std::vector< double > & *force_offset*)

设置底座力矩偏移

参数

force_offset	底座力矩偏移

返回

Python 函数原型

setBaseForceOffset(self: pyaubo_sdk.RobotConfig, arg0: List[float]) -> int

Lua 函数原型

setBaseForceOffset(force_offset: table) -> nil

21.41 getBaseForceOffset()

std::vector< double > arcs::common_interface::RobotConfig::getBaseForce↩ Offset ( )

获取底座力矩偏移

返回

底座力矩偏移

Python 函数原型

getBaseForceOffset(self: pyaubo_sdk.RobotConfig) -> List[float]

Lua 函数原型

getBaseForceOffset() -> table

21.42 getSafetyParametersCheckSum()

uint32_t arcs::common_interface::RobotConfig::getSafetyParametersCheckSum ( )

获取安全参数校验码 CRC32

返回

安全参数校验码

Python 函数原型

getSafetyParametersCheckSum(self: pyaubo_sdk.RobotConfig) -> int

Lua 函数原型

getSafetyParametersCheckSum() -> number

21.43 confirmSafetyParameters()

int arcs::common_interface::RobotConfig::confirmSafetyParameters (const RobotSafetyParameterRange & *parameters*)

发起确认安全配置参数请求: 将安全配置参数写入到安全接口板 flash 或文件

参数

parameters	安全配置参数

返回

Python 函数原型

confirmSafetyParameters(self: pyaubo_sdk.RobotConfig, arg0: arcs::common_interface::RobotSafetyParamete
-> int

Lua 函数原型

21.44 getJointMaxPositions()

std::vector< double > arcs::common_interface::RobotConfig::getJointMax↩ Positions ( )

获取关节最大位置（物理极限）

返回

关节最大位置

Python 函数原型

getJointMaxPositions(self: pyaubo_sdk.RobotConfig) -> List[float]

Lua 函数原型

getJointMaxPositions() -> table

21.45 getJointMinPositions()

std::vector< double > arcs::common_interface::RobotConfig::getJointMin↩ Positions ( )

获取关节最小位置（物理极限）

返回

关节最小位置

Python 函数原型

getJointMinPositions(self: pyaubo_sdk.RobotConfig) -> List[float]

Lua 函数原型

getJointMinPositions() -> table

21.46 getJointMaxSpeeds()

std::vector< double > arcs::common_interface::RobotConfig::getJointMax↩ Speeds ( )

获取关节最大速度（物理极限）

返回

关节最大速度

Python 函数原型

getJointMaxSpeeds(self: pyaubo_sdk.RobotConfig) -> List[float]

Lua 函数原型

getJointMaxSpeeds() -> table

21.47 getJointMaxAccelerations()

std::vector< double > arcs::common_interface::RobotConfig::getJointMax↩ Accelerations ( )

获取关节最大加速度（物理极限）

返回

关节最大加速度

Python 函数原型

getJointMaxAccelerations(self: pyaubo_sdk.RobotConfig) -> List[float]

Lua 函数原型

getJointMaxAccelerations() -> table

21.48 getTcpMaxSpeeds()

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

获取TCP 最大速度（物理极限）

返回

TCP 最大速度

Python 函数原型

getTcpMaxSpeeds(self: pyaubo_sdk.RobotConfig) -> List[float]

Lua 函数原型

getTcpMaxSpeeds() -> table

21.49 getTcpMaxAccelerations()

std::vector< double > arcs::common_interface::RobotConfig::getTcpMax↩ Accelerations ( )

获取TCP 最大加速度（物理极限）

返回

TCP 最大加速度

Python 函数原型

getTcpMaxAccelerations(self: pyaubo_sdk.RobotConfig) -> List[float]

Lua 函数原型

getTcpMaxAccelerations() -> table

21.50 setGravity()

int arcs::common_interface::RobotConfig::setGravity (
const std::vector< double > & *gravity*)

设置机器人安装姿态

参数

gravity	安装姿态

返回

Python 函数原型

setGravity(self: pyaubo_sdk.RobotConfig, arg0: List[float]) -> int

Lua 函数原型

setGravity(gravity: table) -> nil

21.51 getGravity()

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

获取机器人的安装姿态

如果机器人底座安装了姿态传感器，则从传感器读取数据，否则按照用户设置

返回

安装姿态

Python 函数原型

getGravity(self: pyaubo_sdk.RobotConfig) -> List[float]

Lua 函数原型

getGravity() -> table

21.52 setTcpOffset()

int arcs::common_interface::RobotConfig::setTcpOffset (
const std::vector< double > & *offset*)

设置TCP 偏移

参数

offset	TCP 偏移

返回

Python 函数原型

setTcpOffset(self: pyaubo_sdk.RobotConfig, arg0: List[float]) -> int

Lua 函数原型

setTcpOffset(offset: table) -> nil

21.53 getTcpOffset()

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

获取TCP 偏移

返回

TCP 偏移

Python 函数原型

getTcpOffset(self: pyaubo_sdk.RobotConfig) -> List[float]

Lua 函数原型

getTcpOffset() -> table

21.54 setToolInertial()

int arcs::common_interface::RobotConfig::setToolInertial (
double *m*,
const std::vector< double > & *com*,
const std::vector< double > & *inertial*)

设置工具端质量、质心及惯量

参数

m	工具端质量
com	质心
inertial	惯量

返回

Python 函数原型

setToolInertial(self: pyaubo_sdk.RobotConfig, arg0: float, arg1: List[float], arg2↩ : List[float]) -> int

Lua 函数原型

setToolInertial(m: number, com: table, inertial: table) -> nil

21.55 setPayload()

int arcs::common_interface::RobotConfig::setPayload (
double *m*,
const std::vector< double > & *cog*,
const std::vector< double > & *aom*,
const std::vector< double > & *inertia*)

设置末端负载

参数

m	质量
cog	重心
aom	
inertia	惯量

返回

Python 函数原型

setPayload(self: pyaubo_sdk.RobotConfig, arg0: float, arg1: List[float], arg2: List[float], arg3: List[float]) -> int

Lua 函数原型

setPayload(m: number, cog: table, aom: table, inertia: table) -> nil

21.56 getPayload()

Payload arcs::common_interface::RobotConfig::getPayload ( )

获取末端负载

返回

末端负载

Python 函数原型

getPayload(self: pyaubo_sdk.RobotConfig) -> Tuple[float, List[float], List[float], List[float]]

Lua 函数原型

getPayload() -> number

21.57 toolSpaceInRange()

bool arcs::common_interface::RobotConfig::toolSpaceInRange (const std::vector< double > & *pose*)

末端位姿是否在安全范围之内

参数

pose	末端位姿

返回

在安全范围内返回 true; 反之返回 false

Python 函数原型

toolSpaceInRange(self: pyaubo_sdk.RobotConfig, arg0: List[float]) -> bool

Lua 函数原型

toolSpaceInRange(pose: table) -> boolean

21.58 firmwareUpdate()

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

发起固件升级请求，控制器软件将进入固件升级模式

参数

fw	固件路径 pm://param/model/xx.bin /absolute/path/to/xx.bin relative/path/to/xx.bin

返回

成功返回 0
失败返回错误码 AUBO_BAD_STATE: 当前运行时状态不处于 Stopped, 固件升级请求被拒绝

Python 函数原型

firmwareUpdate(self: pyaubo_sdk.RobotConfig, arg0: str) -> int

Lua 函数原型

firmwareUpdate(fw: string) -> nil

21.59 getFirmwareUpdateProcess()

std::tuple< std::string, double > arcs::common_interface::RobotConfig::get↩ FirmwareUpdateProcess ( )

获取固件升级的进程

返回

升级进程
std::string 代表步骤名称
double 代表进度 (0~1)，完成之后，返回 ("", 1)

Python 函数原型

getFirmwareUpdateProcess(self: pyaubo_sdk.RobotConfig) -> Tuple[str, float]

Lua 函数原型

getFirmwareUpdateProcess() -> table

22. RobotManage 类

22.1 poweron()

int arcs::common_interface::RobotManage::poweron ( )

发起机器人上电请求

返回

Python 函数原型

poweron(self: pyaubo_sdk.RobotManage) -> int

Lua 函数原型

poweron() -> number

C++ 示例

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

22.2 startup()

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

发起机器人启动请求

返回

Python 函数原型

startup(self: pyaubo_sdk.RobotManage) -> int

Lua 函数原型

startup() -> number

C++ 示例

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

22.3 poweroff()

int arcs::common_interface::RobotManage::poweroff ( )

发起机器人断电请求

返回

Python 函数原型

poweroff(self: pyaubo_sdk.RobotManage) -> int

Lua 函数原型

poweroff() -> number

C++ 示例

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

22.4 backdrive()

int arcs::common_interface::RobotManage::backdrive (bool *enable*)

发起机器人反向驱动请求

参数

enable	true 进入反向驱动模式；false 退出反向驱动模式

返回

Python 函数原型

backdrive(self: pyaubo_sdk.RobotManage, arg0: bool) -> int

Lua 函数原型

backdrive(enable: boolean) -> number

C++ 示例

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

22.5 freedrive()

int arcs::common_interface::RobotManage::freedrive (bool *enable*)

发起机器人自由驱动请求

参数

enable	true 进入拖动模式；false 退出拖动模式

返回

Python 函数原型

freedrive(self: pyaubo_sdk.RobotManage, arg0: bool) -> int

Lua 函数原型

freedrive(enable: boolean) -> number

C++ 示例

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

22.6 setSim()

int arcs::common_interface::RobotManage::setSim (bool *enable*)

发起机器人进入/退出仿真模式请求

参数

enable	true 进入仿真模式；false 退出仿真模式

返回

Python 函数原型

setSim(self: pyaubo_sdk.RobotManage, arg0: bool) -> int

Lua 函数原型

setSim(enable: boolean) -> number

C++ 示例

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

22.7 setOperationalMode()

int arcs::common_interface::RobotManage::setOperationalMode (OperationalModeType *mode*)

设置机器人操作模式

参数

mode	操作模式

返回

Python 函数原型

setOperationalMode(self: pyaubo_sdk.RobotManage, arg0: arcs::common_interface::↩ OperationalModeType) -> int

Lua 函数原型

setOperationalMode(mode: number) -> number

C++ 示例

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

22.8 getOperationalMode()

OperationalModeType arcs::common_interface::RobotManage::getOperational↩ Mode ( )

获取机器人操作模式

返回

机器人操作模式

Python 函数原型

getOperationalMode(self: pyaubo_sdk.RobotManage) -> arcs::common_interface::↩ OperationalModeType

Lua 函数原型

getOperationalMode() -> number

C++ 示例

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

22.9 getRobotControlMode()

RobotControlModeType arcs::common_interface::RobotManage::getRobot↩ ControlMode ( )

获取控制模式

返回

控制模式

Python 函数原型

getRobotControlMode(self: pyaubo_sdk.RobotManage) -> arcs::common_interface::↩
RobotControlModeType

Lua 函数原型

getRobotControlMode() -> number

C++ 示例

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

22.10 isFreedriveEnabled()

bool arcs::common_interface::RobotManage::isFreedriveEnabled ( )

是否使能了拖动示教模式

返回

使能返回 true; 反之返回 false

Python 函数原型

isFreedriveEnabled(self: pyaubo_sdk.RobotManage) -> bool

Lua 函数原型

isFreedriveEnabled() -> boolean

C++ 示例

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

22.11 isBackdriveEnabled()

bool arcs::common_interface::RobotManage::isBackdriveEnabled ( )

是否使能了反向驱动模式

返回

使能返回 true; 反之返回 false

Python 函数原型

isBackdriveEnabled(self: pyaubo_sdk.RobotManage) -> bool

Lua 函数原型

isBackdriveEnabled() -> boolean

C++ 示例

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

22.12 isSimulationEnabled()

bool arcs::common_interface::RobotManage::isSimulationEnabled ( )

是否使能了仿真模式

返回

使能返回 true; 反之返回 false

Python 函数原型

isSimulationEnabled(self: pyaubo_sdk.RobotManage) -> bool

Lua 函数原型

isSimulationEnabled() -> boolean

C++ 示例

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

22.13 setUnlockProtectiveStop()

int arcs::common_interface::RobotManage::setUnlockProtectiveStop ( )

清除防护停机，包括碰撞停机

返回

Python 函数原型

setUnlockProtectiveStop(self: pyaubo_sdk.RobotManage) -> int

Lua 函数原型

setUnlockProtectiveStop() -> number

C++ 示例

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

22.14 restartInterfaceBoard()

int arcs::common_interface::RobotManage::restartInterfaceBoard ( )

重置安全接口板，一般在机器人断电之后需要重置时调用，比如机器人急停、故障等之后

返回

Python 函数原型

restartInterfaceBoard(self: pyaubo_sdk.RobotManage) -> int

Lua 函数原型

restartInterfaceBoard() -> number

C++ 示例

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

23. RobotMsg 结构体

23.1 arcs::common_interface::RobotMsg 结构体参考

Public **属性**

- uint64_t timestamp
- TraceLevel level
- int code
- std::string source
- std::vector< std::string > args

24. RobotSafetyParameterRang 结构体

24.1 arcs::common_interface::RobotSafetyParameterRange 结构体参考

Public **属性**

- uint32_t crc32 { 0 }
- struct {
  REAL power
  REAL momentum
  REAL stop_time
  REAL stop_distance
  REAL tcp_speed
  REAL elbow_speed
  REAL tcp_force
  REAL elbow_force
  std::vector< REAL > qmin
  std::vector< REAL > qmax
  std::vector< REAL > qdmax
  std::vector< REAL > joint_torque
  Vector3f tool_orientation
  REAL tool_deviation
  Vector4f planes [SAFETY_PLANES_NUM]
  } params [SAFETY_PARAM_SELECT_NUM]
- struct {
  Vector4f plane
  bool restrict_elbow
  int param_select
  } trigger_planes [SAFETY_PLANES_NUM]
- Vector4f tools [TOOL_CONFIGURATION_NUM]
- REAL tool_inclination { 0. }
- REAL tool_azimuth { 0. }

- std::vector< REAL > safety_home
- uint32_t safety_input_emergency_stop
- uint32_t safety_input_safegurd_stop
- uint32_t safety_input_safeguard_reset
- uint32_t safety_input_auto_safegurd_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_safe_home
- uint32_t safety_output_reduced_mode
- uint32_t safety_output_not_reduced_mode
- uint32_t safety_output_emergency_stop
- uint32_t safety_output_robot_moving
- uint32_t safety_output_robot_steady

25. RobotState 类

25.1 getRobotModeType()

RobotModeType arcs::common_interface::RobotState::getRobotModeType ( )

获取机器人的模式状态

返回

机器人的模式状态

Python 函数原型

getRobotModeType(self: pyaubo_sdk.RobotState) -> arcs::common_interface::Robot↩ ModeType

Lua 函数原型

getRobotModeType() -> number

25.2 getSafetyModeType()

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

获取安全模式

返回

安全模式

Python 函数原型

getSafetyModeType(self: pyaubo_sdk.RobotState) -> arcs::common_interface::Safety↩ ModeType

Lua 函数原型

getSafetyModeType() -> number

25.3 isPowerOn()

bool arcs::common_interface::RobotState::isPowerOn ( )

获取机器人通电状态

返回

机器人通电状态

25.4 isSteady()

bool arcs::common_interface::RobotState::isSteady ( )

机器人是否已经停止下来

返回

停止返回 true; 反之返回 false

Python 函数原型

isSteady(self: pyaubo_sdk.RobotState) -> bool

Lua 函数原型

isSteady() -> boolean

25.5 isCollisionOccurred()

bool arcs::common_interface::RobotState::isCollisionOccurred ( )

机器人是否发生了碰撞

返回

发生碰撞返回 true；反之返回 false

25.6 isWithinSafetyLimits()

bool arcs::common_interface::RobotState::isWithinSafetyLimits ( )

机器人是否已经在安全限制之内

返回

在安全限制之内返回 true; 反之返回 false

Python 函数原型

isWithinSafetyLimits(self: pyaubo_sdk.RobotState) -> bool

Lua 函数原型

isWithinSafetyLimits() -> boolean

25.7 getTcpPose()

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

获取当前的TCP 位姿

返回

TCP 的位姿

Python 函数原型

getTcpPose(self: pyaubo_sdk.RobotState) -> List[float]

Lua 函数原型

getTcpPose() -> table

25.8 getTargetTcpPose()

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

获取（由算法规划的）下发的目标位姿

返回

当前目标位姿

Python 函数原型

getTargetTcpPose(self: pyaubo_sdk.RobotState) -> List[float]

Lua 函数原型

getTargetTcpPose() -> table

25.9 getToolPose()

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

获取工具端的位姿（不带TCP 偏移）

返回

工具端的位姿

Python 函数原型

getToolPose(self: pyaubo_sdk.RobotState) -> List[float]

Lua 函数原型

getToolPose() -> table

25.10 getTcpSpeed()

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

获取TCP 速度

返回

TCP 速度

Python 函数原型

getTcpSpeed(self: pyaubo_sdk.RobotState) -> List[float]

Lua 函数原型

getTcpSpeed() -> table

25.11 getTcpForce()

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

获取TCP 的力/力矩

返回

TCP 的力/力矩

Python 函数原型

getTcpForce(self: pyaubo_sdk.RobotState) -> List[float]

Lua 函数原型

getTcpForce() -> table

25.12 getElbowPosistion()

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

获取肘部的位置

返回

肘部的位置

Python 函数原型

getElbowPosistion(self: pyaubo_sdk.RobotState) -> List[float]

Lua 函数原型

getElbowPosistion() -> table

25.13 getElbowVelocity()

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

获取肘部速度

返回

肘部速度

Python 函数原型

getElbowVelocity(self: pyaubo_sdk.RobotState) -> List[float]

Lua 函数原型

getElbowVelocity() -> table

25.14 getBaseForce()

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

获取基座力/力矩

返回

基座力/力矩

Python 函数原型

getBaseForce(self: pyaubo_sdk.RobotState) -> List[float]

Lua 函数原型

getBaseForce() -> table

25.15 getTcpTargetPose()

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

获取上一次发送的TCP 目标位姿

返回

TCP 目标位姿

Python 函数原型

getTcpTargetPose(self: pyaubo_sdk.RobotState) -> List[float]

Lua 函数原型

getTcpTargetPose() -> table

25.16 getTcpTargetSpeed()

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

获取TCP 目标速度

返回

TCP 目标速度

Python 函数原型

getTcpTargetSpeed(self: pyaubo_sdk.RobotState) -> List[float]

Lua 函数原型

getTcpTargetSpeed() -> table

25.17 getTcpTargetForce()

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

获取TCP 目标力/力矩

返回

TCP 目标力/力矩

Python 函数原型

getTcpTargetForce(self: pyaubo_sdk.RobotState) -> List[float]

Lua 函数原型

getTcpTargetForce() -> table

25.18 getJointState()

std::vector< JointStateType > arcs::common_interface::RobotState::getJoint↩ State ( )

获取机械臂关节标志

返回

机械臂关节标志

Python 函数原型

getJointState(self: pyaubo_sdk.RobotState) -> List[arcs::common_interface::JointState↩ Type]

Lua 函数原型

getJointState() -> table

25.19 getJointServoMode()

std::vector< JointServoModeType > arcs::common_interface::RobotState::get↩ JointServoMode ( )

获取关节的伺服状态

返回

关节的伺服状态

Python 函数原型

getJointServoMode(self: pyaubo_sdk.RobotState) -> List[arcs::common_interface::Joint↩ ServoModeType]

Lua 函数原型

getJointServoMode() -> table

25.20 getJointPositions()

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

获取机械臂关节角度

返回

机械臂关节角度

Python 函数原型

getJointPositions(self: pyaubo_sdk.RobotState) -> List[float]

Lua 函数原型

getJointPositions() -> table

25.21 getJointSpeeds()

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

获取机械臂关节速度

返回

机械臂关节速度

Python 函数原型

getJointSpeeds(self: pyaubo_sdk.RobotState) -> List[float]

Lua 函数原型

getJointSpeeds() -> table

25.22 getJointAccelerations()

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

获取机械臂关节加速度

返回

机械臂关节加速度

Python 函数原型

getJointAccelerations(self: pyaubo_sdk.RobotState) -> List[float]

Lua 函数原型

getJointAccelerations() -> table

25.23 getJointTorqueSensors()

std::vector< double > arcs::common_interface::RobotState::getJointTorque↩ Sensors ( )

获取机械臂关节力矩

返回

机械臂关节力矩

Python 函数原型

getJointTorqueSensors(self: pyaubo_sdk.RobotState) -> List[float]

Lua 函数原型

getJointTorqueSensors() -> table

25.24 getBaseForceSensor()

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

获取底座力传感器读数

返回

底座力传感器读数

Python 函数原型

getBaseForceSensor(self: pyaubo_sdk.RobotState) -> List[float]

Lua 函数原型

getBaseForceSensor() -> table

25.25 getTcpForceSensors()

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

获取TCP 力传感器读数

返回

TCP 力传感器读数

Python 函数原型

getTcpForceSensors(self: pyaubo_sdk.RobotState) -> List[float]

Lua 函数原型

getTcpForceSensors() -> table

25.26 getJointCurrents()

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

获取机械臂关节电流

返回

机械臂关节电流

Python 函数原型

getJointCurrents(self: pyaubo_sdk.RobotState) -> List[float]

Lua 函数原型

getJointCurrents() -> table

25.27 getJointVoltages()

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

获取机械臂关节电压

返回

机械臂关节电压

Python 函数原型

getJointVoltages(self: pyaubo_sdk.RobotState) -> List[float]

Lua 函数原型

getJointVoltages() -> table

25.28 getJointTemperatures()

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

获取机械臂关节温度

返回

机械臂关节温度

Python 函数原型

getJointTemperatures(self: pyaubo_sdk.RobotState) -> List[float]

Lua 函数原型

getJointTemperatures() -> table

25.29 getJointUniqueIds()

std::vector< std::string > arcs::common_interface::RobotState::getJoint↩ UniqueIds ( )

获取关节全球唯一ID

返回

关节全球唯一ID

Python 函数原型

getJointUniqueIds(self: pyaubo_sdk.RobotState) -> List[str]

Lua 函数原型

getJointUniqueIds() -> table

25.30 getJointFirmwareVersions()

std::vector< int > arcs::common_interface::RobotState::getJointFirmware↩ Versions ( )

获取关节固件版本

返回

关节固件版本

Python 函数原型

getJointFirmwareVersions(self: pyaubo_sdk.RobotState) -> List[int]

Lua 函数原型

getJointFirmwareVersions() -> table

25.31 getJointHardwareVersions()

std::vector< int > arcs::common_interface::RobotState::getJointHardware↩ Versions ( )

获取关节硬件版本

返回

关节硬件版本

Python 函数原型

getJointHardwareVersions(self: pyaubo_sdk.RobotState) -> List[int]

Lua 函数原型

getJointHardwareVersions() -> table

25.32 getMasterBoardUniqueId()

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

获取MasterBoard 全球唯一ID

返回

MasterBoard 全球唯一ID

Python 函数原型

getMasterBoardUniqueId(self: pyaubo_sdk.RobotState) -> str

Lua 函数原型

getMasterBoardUniqueId() -> string

25.33 getMasterBoardFirmwareVersion()

int arcs::common_interface::RobotState::getMasterBoardFirmwareVersion ( )

获取MasterBoard 固件版本

返回

MasterBoard 固件版本

Python 函数原型

getMasterBoardFirmwareVersion(self: pyaubo_sdk.RobotState) -> int

Lua 函数原型

getMasterBoardFirmwareVersion() -> number

25.34 getMasterBoardHardwareVersion()

int arcs::common_interface::RobotState::getMasterBoardHardwareVersion ( )

获取MasterBoard 硬件版本

返回

MasterBoard 硬件版本

Python 函数原型

getMasterBoardHardwareVersion(self: pyaubo_sdk.RobotState) -> int

Lua 函数原型

getMasterBoardHardwareVersion() -> number

25.35 getSlaveBoardUniqueId()

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

获取SlaveBoard 全球唯一ID

返回

SlaveBoard 全球唯一ID

Python 函数原型

getSlaveBoardUniqueId(self: pyaubo_sdk.RobotState) -> str

Lua 函数原型

getSlaveBoardUniqueId() -> string

25.36 getSlaveBoardFirmwareVersion()

int arcs::common_interface::RobotState::getSlaveBoardFirmwareVersion ( )

获取SlaveBoard 固件版本

返回

SlaveBoard 固件版本

Python 函数原型

getSlaveBoardFirmwareVersion(self: pyaubo_sdk.RobotState) -> int

Lua 函数原型

getSlaveBoardFirmwareVersion() -> number

25.37 getSlaveBoardHardwareVersion()

int arcs::common_interface::RobotState::getSlaveBoardHardwareVersion ( )

获取SlaveBoard 硬件版本

返回

SlaveBoard 硬件版本

Python 函数原型

getSlaveBoardHardwareVersion(self: pyaubo_sdk.RobotState) -> int

Lua 函数原型

getSlaveBoardHardwareVersion() -> number

25.38 getToolUniqueId()

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

获取工具端全球唯一ID

返回

工具端全球唯一ID

Python 函数原型

getToolUniqueId(self: pyaubo_sdk.RobotState) -> str

Lua 函数原型

getToolUniqueId() -> string

25.39 getToolFirmwareVersion()

int arcs::common_interface::RobotState::getToolFirmwareVersion ( )

获取工具端固件版本

返回

工具端固件版本

Python 函数原型

getToolFirmwareVersion(self: pyaubo_sdk.RobotState) -> int

Lua 函数原型

getToolFirmwareVersion() -> number

25.40 getToolHardwareVersion()

int arcs::common_interface::RobotState::getToolHardwareVersion ( )

获取工具端硬件版本

返回

工具端硬件版本

Python 函数原型

getToolHardwareVersion(self: pyaubo_sdk.RobotState) -> int

Lua 函数原型

getToolHardwareVersion() -> number

25.41 getPedestalUniqueId()

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

获取底座全球唯一ID

返回

底座全球唯一ID

Python 函数原型

getPedestalUniqueId(self: pyaubo_sdk.RobotState) -> str

Lua 函数原型

getPedestalUniqueId() -> string

25.42 getPedestalFirmwareVersion()

int arcs::common_interface::RobotState::getPedestalFirmwareVersion ( )

获取底座固件版本

返回

底座固件版本

Python 函数原型

getPedestalFirmwareVersion(self: pyaubo_sdk.RobotState) -> int

Lua 函数原型

getPedestalFirmwareVersion() -> number

25.43 getPedestalHardwareVersion()

int arcs::common_interface::RobotState::getPedestalHardwareVersion ( )

获取底座硬件版本

返回

底座硬件版本

Python 函数原型

getPedestalHardwareVersion(self: pyaubo_sdk.RobotState) -> int

Lua 函数原型

getPedestalHardwareVersion() -> number

25.44 getJointTargetPositions()

std::vector< double > arcs::common_interface::RobotState::getJointTarget↩ Positions ( )

获取机械臂关节目标位置角度

返回

机械臂关节目标位置角度

Python 函数原型

getJointTargetPositions(self: pyaubo_sdk.RobotState) -> List[float]

Lua 函数原型

getJointTargetPositions() -> table

25.45 getJointTargetSpeeds()

std::vector< double > arcs::common_interface::RobotState::getJointTarget↩ Speeds ( )

获取机械臂关节目标速度

返回

机械臂关节目标速度

Python 函数原型

getJointTargetSpeeds(self: pyaubo_sdk.RobotState) -> List[float]

Lua 函数原型

getJointTargetSpeeds() -> table

25.46 getJointTargetAccelerations()

std::vector< double > arcs::common_interface::RobotState::getJointTarget↩ Accelerations ( )

获取机械臂关节目标加速度

返回

机械臂关节目标加速度

Python 函数原型

getJointTargetAccelerations(self: pyaubo_sdk.RobotState) -> List[float]

Lua 函数原型

getJointTargetAccelerations() -> table

25.47 getJointTargetTorques()

std::vector< double > arcs::common_interface::RobotState::getJointTarget↩ Torques ( )

获取机械臂关节目标力矩

返回

机械臂关节目标力矩

Python 函数原型

getJointTargetTorques(self: pyaubo_sdk.RobotState) -> List[float]

Lua 函数原型

getJointTargetTorques() -> table

25.48 getJointTargetCurrents()

std::vector< double > arcs::common_interface::RobotState::getJointTarget↩ Currents ( )

获取机械臂关节目标电流

返回

机械臂关节目标电流

Python 函数原型

getJointTargetCurrents(self: pyaubo_sdk.RobotState) -> List[float]

Lua 函数原型

getJointTargetCurrents() -> table

25.49 getControlBoxTemperature()

double arcs::common_interface::RobotState::getControlBoxTemperature ( )

获取控制柜温度

返回

控制柜温度

Python 函数原型

getControlBoxTemperature(self: pyaubo_sdk.RobotState) -> float

Lua 函数原型

getControlBoxTemperature() -> number

25.50 getMainVoltage()

double arcs::common_interface::RobotState::getMainVoltage ( )

获取母线电压

返回

母线电压

Python 函数原型

getMainVoltage(self: pyaubo_sdk.RobotState) -> float

Lua 函数原型

getMainVoltage() -> number

25.51 getMainCurrent()

double arcs::common_interface::RobotState::getMainCurrent ( )

获取母线电流

返回

母线电流

Python 函数原型

getMainCurrent(self: pyaubo_sdk.RobotState) -> float

Lua 函数原型

getMainCurrent() -> number

25.52 getRobotVoltage()

double arcs::common_interface::RobotState::getRobotVoltage ( )

获取机器人电压

返回

机器人电压

Python 函数原型

getRobotVoltage(self: pyaubo_sdk.RobotState) -> float

Lua 函数原型

getRobotVoltage() -> number

25.53 getRobotCurrent()

double arcs::common_interface::RobotState::getRobotCurrent ( )

获取机器人电流

返回

机器人电流

Python 函数原型

getRobotCurrent(self: pyaubo_sdk.RobotState) -> float

Lua 函数原型

getRobotCurrent() -> number