

airbot::AirbotSdk Class Reference

Core class for interacting with the Airbot robotic arm. [More...](#)

```
#include <airbot_sdk.hpp>
```

Public Member Functions

	AirbotSdk (const std::string &url="localhost", const unsigned int port=50051)	Constructor that initializes the SDK with the given connection details.
const std::shared_ptr< spdlog::logger >	GetLogger () const	Retrieves the logger used by the SDK.
bool	IsRunning () const	Checks if the SDK is running and connected to the robotic arm.
bool	Connect () const	Establishes a connection to the robotic arm.
bool	Disconnect ()	Disconnects from the robotic arm.
std::vector< std::pair< std::string, std::vector< std::string > > >	GetProductInfo () const	Retrieves product information (e.g., name, version) from the robotic arm.
State	GetState () const	Retrieves the current state of the robotic arm.
RobotMode	GetControlMode () const	Retrieves the current control mode of the robotic arm.
std::vector< std::pair< std::string, std::variant< int32_t, double, std::string, bool > > >	GetParams (std::vector< std::string > names) const	Retrieves the modes available for each robot component.
std::pair< std::array< double, 3 >, std::array< double, 4 > >	GetEndPose () const	Retrieves the end-effector pose (position and orientation) of the robotic arm.
std::array< double, 6 >	GetJointPos () const	Retrieves the current joint positions of the robotic arm.
std::array< double, 6 >	GetJointVel () const	Retrieves the current joint velocities of the robotic arm.
std::array< double, 6 >	GetJointEff () const	Retrieves the current joint efforts (torque) of the robotic arm.

	<code>std::vector< double ></code>	<code>GetEefPos () const</code>	Retrieves the current position of the end-effector (EEF).
	<code>std::vector< double ></code>	<code>GetEefVel () const</code>	Retrieves the current velocity of the end-effector (EEF).
	<code>std::vector< double ></code>	<code>GetEefEff () const</code>	Retrieves the current force/effort of the end-effector (EEF).
	<code>bool</code>	<code>SetParams (const std::vector< std::pair< std::string, std::variant< int32_t, double, std::string, bool > > &params) const</code>	Sets the parameters of the robotic arm.
	<code>bool</code>	<code>SetSpeedProfile (const SpeedProfile &profile) const</code>	Sets the speed profile for the robotic arm.
	<code>bool</code>	<code>SwitchMode (const RobotMode &mode) const</code>	Switches the control mode of the robotic arm.
	<code>bool</code>	<code>SwitchMode (const std::vector< std::string > &component_names, const std::vector< RobotMode > &modes) const</code>	Switches the control mode for multiple robot components.
	<code>bool</code>	<code>LoadApp (const std::string &name="record_replay_app/record_replay_app::RecordReplayApp", const std::vector< std::pair< std::string, std::string > > &params={}) const</code>	Loads an application onto the robotic arm.
	<code>bool</code>	<code>UnloadApp () const</code>	Unloads the currently loaded application from the robotic arm.
	<code>bool</code>	<code>MoveToJointPos (const std::array< double, 6 > &joint_pos) const</code>	Moves the robotic arm to the specified joint positions.
	<code>std::future< bool ></code>	<code>AsyncMoveToJointPos (const std::array< double, 6 > &joint_pos) const</code>	Asynchronously moves the robotic arm to the specified joint positions.
	<code>bool</code>	<code>MoveToCartPos (const std::pair< std::array< double, 3 >, std::array< double, 4 > > &cart_pos) const</code>	Moves the robotic arm to the specified Cartesian position and orientation.
	<code>std::future< bool ></code>	<code>AsyncMoveToCartPos (const std::pair< std::array< double, 3 >, std::array<</code>	

		<code>double, 4 > > &cart_pos) const</code>	Asynchronously moves the robotic arm to the specified Cartesian position and orientation.
	<code>bool</code>	<code>MoveEefPos (const std::vector< double > &eef_pos) const</code>	Moves the robotic arm's end-effector to the specified position.
	<code>std::future< bool ></code>	<code>AsyncMoveEefPos (const std::vector< double > &eef_pos) const</code>	Asynchronously moves the robotic arm's end-effector to the specified position.
	<code>bool</code>	<code>MoveEefPos (const double &eef_pos) const</code>	Moves the robotic arm's end-effector to the specified position (single value).
	<code>std::future< bool ></code>	<code>AsyncMoveEefPos (const double &eef_pos) const</code>	Asynchronously moves the robotic arm's end-effector to the specified position (single value).
	<code>bool</code>	<code>MoveWithCartWaypoints (const std::vector< std::pair< std::array< double, 3 >, std::array< double, 4 > > > &cart_waypoints) const</code>	Moves the robotic arm along specified Cartesian waypoints.
	<code>std::future< bool ></code>	<code>AsyncMoveWithCartWaypoints (const std::vector< std::pair< std::array< double, 3 >, std::array< double, 4 > > > &cart_waypoints) const</code>	Asynchronously moves the robotic arm along specified Cartesian waypoints.
	<code>bool</code>	<code>MoveWithJointWaypoints (const std::vector< std::array< double, 6 > > &joint_waypoints) const</code>	Moves the robotic arm along specified joint waypoints.
	<code>std::future< bool ></code>	<code>AsyncMoveWithJointWaypoints (const std::vector< std::array< double, 6 > > &joint_waypoints) const</code>	Asynchronously moves the robotic arm along specified joint waypoints.
	<code>void</code>	<code>MitJointIntegratedControl (const std::array< double, 6 > &joint_pos, const std::array< double, 6 > &joint_vel, const std::array< double, 6 > &joint_eff, const std::array< double, 6 > &joint_kp, const std::array< double, 6 > &joint_kd) const</code>	Sends integrated MIT control commands to the robotic arm's joints.
	<code>void</code>	<code>ServoJointPos (const std::array< double, 6 > &joint_pos) const</code>	Directly controls the robotic arm's joints to move to the specified positions.

	<code>void</code> <code>ServoJointVel</code> (<code>const std::array< double, 6 > &joint_vel</code>) <code>const</code>	Directly controls the robotic arm's joints to move at the specified velocities.
	<code>void</code> <code>ServoCartPos</code> (<code>const std::pair< std::array< double, 3 >, std::array< double, 4 > > &cart_pos</code>) <code>const</code>	Directly controls the robotic arm's end-effector position.
	<code>void</code> <code>ServoCartTwist</code> (<code>const std::pair< std::array< double, 3 >, std::array< double, 3 > > &twist</code>) <code>const</code>	Directly controls the robotic arm's twist (linear and angular velocity) in Cartesian space.
	<code>void</code> <code>ServoEefPos</code> (<code>const std::vector< double > &eef_pos</code>) <code>const</code>	Directly controls the robotic arm's end-effector position.
	<code>void</code> <code>ServoEefPos</code> (<code>const double &eef_pos</code>) <code>const</code>	Directly controls the robotic arm's end-effector position (single value).
	<code>~AirbotSdk</code> (<code></code>)	Destructor that cleans up resources used by the SDK.

Detailed Description

Core class for interacting with the Airbot robotic arm.

This class provides all necessary methods for interacting with the Airbot robotic arm, such as connecting to the arm, retrieving status information, and controlling motion. It supports both synchronous and asynchronous operations for controlling the robotic arm's joints, end effector, and cartesian pose.

Example usage:

```
auto sdk = std::make_unique<AirbotSdk>("192.168.0.xxx", 50051);
if (sdk->IsRunning()) {
    sdk->GetLogger()->info("SDK start running:");
    sdk->GetLogger()->info("Product Info:");
    auto info = sdk->GetProductInfo();
    for (const auto& [key, value] : info) {
        sdk->GetLogger()->info("  {}: {}", key, value);
    }
}
```

Constructor & Destructor Documentation

◆ AirbotSdk()

```
airbot::AirbotSdk::AirbotSdk ( const std::string &url = "localhost",
                               const unsigned int port = 50051 )
```

Constructor that initializes the SDK with the given connection details.

Parameters

- url** The IP address or hostname of the robot (default: "localhost").
- port** The port number to connect to (default: 50051).

Initializes the SDK instance with the specified connection details. The SDK will not automatically connect to the robot upon initialization. Use `Connect()` to establish a connection.

◆ ~AirbotSdk()

```
airbot::AirbotSdk::~AirbotSdk ( )
```

Destructor that cleans up resources used by the SDK.

Cleans up any resources used by the SDK, including disconnecting from the robotic arm if necessary.

Member Function Documentation

◆ AsyncMoveEefPos() [1/2]

```
std::future< bool > airbot::AirbotSdk::AsyncMoveEefPos ( const double &eef_pos ) const
```

Asynchronously moves the robotic arm's end-effector to the specified position (single value).

Parameters

- eef_pos** The target position value for the end-effector.

Returns

A future representing the result of the move operation.

Asynchronously moves the robotic arm's end-effector to the specified position. The target position must be provided as a single double value. The function returns a future that will contain the result of the move operation.

◆ AsyncMoveEefPos() [2/2]

```
std::future< bool > airbot::AirbotSdk::AsyncMoveEefPos ( const std::vector< double > &eef_pos ) const
```

Asynchronously moves the robotic arm's end-effector to the specified position.

Parameters

- eef_pos** A vector containing the target end-effector position.

Returns

A future representing the result of the move operation.

Asynchronously moves the robotic arm's end-effector to the specified position. The target position must be provided as a vector of doubles. The function returns a future that will contain the result of the move operation.

◆ AsyncMoveToCartPos()

```
std::future< bool >
airbot::AirbotSdk::AsyncMoveToCartPos ( const std::pair< std::array< double, 3 >, std::array< double, 4 > > &cart_pos ) const
```

Asynchronously moves the robotic arm to the specified Cartesian position and orientation.

Parameters

cart_pos A pair containing the position (3D vector) and orientation (4D quaternion).

Returns

A future representing the result of the move operation.

Asynchronously moves the robotic arm to the specified Cartesian position and orientation. The position and orientation must be provided as a pair of arrays. The function returns a future that will contain the result of the move operation.

◆ AsyncMoveToJointPos()

```
std::future< bool > airbot::AirbotSdk::AsyncMoveToJointPos ( const std::array< double, 6 > &joint_pos ) const
```

Asynchronously moves the robotic arm to the specified joint positions.

Parameters

joint_pos An array containing the joint positions to move to.

Returns

A future representing the result of the move operation.

Asynchronously moves the robotic arm to the specified joint positions. The joint positions must be provided as an array of doubles. The function returns a future that will contain the result of the move operation.

◆ AsyncMoveWithCartWaypoints()

```
std::future< bool >
airbot::AirbotSdk::AsyncMoveWithCartWaypoints ( const std::vector< std::pair< std::array< double, 3 >, std::array< double, 4
```

Asynchronously moves the robotic arm along specified Cartesian waypoints.

Parameters

cart_waypoints A vector of waypoints represented as pairs of position (3D vector) and orientation (4D quaternion).

Returns

A future representing the result of the move operation.

Asynchronously moves the robotic arm along a series of Cartesian waypoints. Each waypoint is represented by a pair of arrays: one for the position (x, y, z) and one for the orientation (quaternion: x, y, z, w). The function returns a future that will contain the result of the move operation.

◆ AsyncMoveWithJointWaypoints()

```
std::future< bool >
airbot::AirbotSdk::AsyncMoveWithJointWaypoints ( const std::vector< std::array< double, 6 > > &joint_waypoints ) const
```

Asynchronously moves the robotic arm along specified joint waypoints.

Parameters

joint_waypoints A vector of joint positions to move through.

Returns

A future representing the result of the move operation.

Asynchronously moves the robotic arm along a series of joint waypoints. Each waypoint is represented by an array of joint positions. The function returns a future that will contain the result of the move operation.

◆ Connect()

```
bool airbot::AirbotSdk::Connect ( ) const
```

Establishes a connection to the robotic arm.

Returns

True if the connection is successful, false otherwise.

Establishes a connection to the robotic arm using the connection details provided during SDK initialization.

◆ Disconnect ()

```
bool airbot::AirbotSdk::Disconnect ( )
```

Disconnects from the robotic arm.

Returns

True if the disconnection is successful, false otherwise.

Disconnects the SDK from the robotic arm and releases any associated resources.

◆ GetControlMode ()

```
RobotMode airbot::AirbotSdk::GetControlMode ( ) const
```

Retrieves the current control mode of the robotic arm.

Returns

The current robot control mode.

Retrieves the current control mode of the robotic arm, such as planning or servo mode.

◆ GetEefEff ()

```
std::vector< double > airbot::AirbotSdk::GetEefEff ( ) const
```

Retrieves the current force/effort of the end-effector (EEF).

Returns

A vector containing the force/effort of the end-effector.

Retrieves the current force or effort applied by the end-effector.

◆ GetEefPos ()

```
std::vector< double > airbot::AirbotSdk::GetEefPos ( ) const
```

Retrieves the current position of the end-effector (EEF).

Returns

A vector containing the position of the end-effector.

Retrieves the current position of the end-effector.

◆ GetEefVel ()

```
std::vector< double > airbot::AirbotSdk::GetEefVel ( ) const
```

Retrieves the current velocity of the end-effector (EEF).

Returns

A vector containing the velocity of the end-effector.

Retrieves the current velocity of the end-effector.

◆ GetEndPose ()

```
std::pair< std::array< double, 3 >, std::array< double, 4 > > airbot::AirbotSdk::GetEndPose ( ) const
```

Retrieves the end-effector pose (position and orientation) of the robotic arm.

Returns

A pair containing the position (3D vector) and orientation (4D quaternion) of the end-effector.

Retrieves the current pose of the end-effector, including its position and orientation.

◆ GetJointEff ()

```
std::array< double, 6 > airbot::AirbotSdk::GetJointEff ( ) const
```

Retrieves the current joint efforts (torques) of the robotic arm.

Returns

An array containing the joint efforts (torques) of the robotic arm.

Retrieves the current efforts (torques) of all joints in the robotic arm.

◆ GetJointPos ()

```
std::array< double, 6 > airbot::AirbotSdk::GetJointPos ( ) const
```

Retrieves the current joint positions of the robotic arm.

Returns

An array containing the joint positions of the robotic arm.

Retrieves the current positions of all joints in the robotic arm.

◆ GetJointVel ()

```
std::array< double, 6 > airbot::AirbotSdk::GetJointVel ( ) const
```

Retrieves the current joint velocities of the robotic arm.

Returns

An array containing the joint velocities of the robotic arm.

Retrieves the current velocities of all joints in the robotic arm.

◆ GetLogger ()

```
const std::shared_ptr< spdlog::logger > airbot::AirbotSdk::GetLogger ( ) const
```

Retrieves the logger used by the SDK.

Returns

A shared pointer to the logger instance.

The logger can be used to log messages and events from the SDK.

◆ GetParams ()

```
std::vector< std::pair< std::string, std::variant< int32_t, double, std::string,
bool > > > airbot::AirbotSdk::GetParams ( std::vector< std::string > names ) const
```

Retrieves the modes available for each robot component.

Returns

A pair containing two vectors: one with component names and one with their corresponding modes.

Retrieves the available control modes for each component of the robotic arm.

Retrieves the parameters of the robotic arm specified by the provided names.

Parameters

names A vector of parameter names to retrieve.

Returns

A vector of pairs containing parameter names and their corresponding values.

Retrieves the values of the specified parameters from the robotic arm.

◆ GetProductInfo()

```
std::vector< std::pair< std::string, std::vector< std::string > > > airbot::AirbotSdk::GetProductInfo ( ) const
```

Retrieves product information (e.g., name, version) from the robotic arm.

Returns

A vector of key-value pairs containing product information.

Retrieves information about the robotic arm, such as its name, version, and other relevant details.

◆ GetState()

```
State airbot::AirbotSdk::GetState ( ) const
```

Retrieves the current state of the robotic arm.

Returns

The current state of the robotic arm.

Retrieves the current state of the robotic arm, such as whether it is idle, running, or in an error state.

◆ IsRunning()

```
bool airbot::AirbotSdk::IsRunning ( ) const
```

Checks if the SDK is running and connected to the robotic arm.

Returns

True if the SDK is running, false otherwise.

This method checks whether the SDK is currently connected to the robotic arm and is in a running state.

◆ LoadApp()

```
bool
airbot::AirbotSdk::LoadApp ( const std::string &                               name = "record_replay_app/record_r
                           const std::vector< std::pair< std::string, std::string > > & params = {} ) const
```

Loads an application onto the robotic arm.

Parameters

- name** The name of the application to load (default: "record_replay_app/record_replay_app::RecordReplayApp").
- params** The parameters to pass to the application.

Returns

True if the application is successfully loaded, false otherwise.

Loads an application onto the robotic arm. The application name must be specified, and any required parameters can be passed as a vector of pairs. The function returns true if the application is successfully loaded, otherwise false.

◆ MitJointIntegratedControl()

```
void airbot::AirbotSdk::MitJointIntegratedControl ( const std::array< double, 6 > & joint_pos,
                                                    const std::array< double, 6 > & joint_vel,
                                                    const std::array< double, 6 > & joint_eff,
                                                    const std::array< double, 6 > & joint_kp,
                                                    const std::array< double, 6 > & joint_kd ) const
```

Sends integrated MIT control commands to the robotic arm's joints.

Parameters

- joint_pos** An array of length 6 containing the target joint positions.
- joint_vel** An array of length 6 containing the target joint velocities.
- joint_eff** An array of length 6 containing the target joint efforts (torques).
- joint_kp** An array of length 6 containing the target joint position gains (Kp).
- joint_kd** An array of length 6 containing the target joint velocity gains (Kd).

This function sends integrated MIT control commands to the robotic arm, allowing direct control over each joint's position, velocity, effort, and control gains (Kp, Kd). All input arrays must have a length of 6, corresponding to the 6 joints of the robotic arm. It is typically used in advanced control modes where fine-grained, real-time control of the robot's actuators is required.

Note

Recommended call frequency: 250Hz.

◆ MoveEefPos() [1/2]

```
bool airbot::AirbotSdk::MoveEefPos ( const double & eef_pos ) const
```

Moves the robotic arm's end-effector to the specified position (single value).

Parameters

- eef_pos** The target position value for the end-effector.

Returns

True if the move operation is successful, false otherwise.

Moves the robotic arm's end-effector to the specified position. The target position must be provided as a single double value. The function returns true if the move operation is successful, otherwise false.

◆ MoveEefPos() [2/2]

```
bool airbot::AirbotSdk::MoveEefPos ( const std::vector< double > & eef_pos ) const
```

Moves the robotic arm's end-effector to the specified position.

Parameters

eef_pos A vector containing the target end-effector position.

Returns

True if the move operation is successful, false otherwise.

Moves the robotic arm's end-effector to the specified position. The target position must be provided as a vector of doubles. The function returns true if the move operation is successful, otherwise false.

Moves the robotic arm's end-effector to the specified position.

Parameters

eef_pos A vector containing the target end-effector position.

Returns

True if the move operation is successful, false otherwise.

Moves the robotic arm's end-effector to the specified position. The target position must be provided as a vector of doubles. The function returns true if the move operation is successful, otherwise false.

◆ MoveToCartPos()

```
bool  
airbot::AirbotSdk::MoveToCartPos ( const std::pair< std::array< double, 3 >, std::array< double, 4 > > & cart_pos ) const
```

Moves the robotic arm to the specified Cartesian position and orientation.

Parameters

cart_pos A pair containing the position (3D vector) and orientation (4D quaternion).

Returns

True if the move operation is successful, false otherwise.

Moves the robotic arm to the specified Cartesian position and orientation. The position and orientation must be provided as a pair of arrays. The function returns true if the move operation is successful, otherwise false.

◆ MoveToJointPos()

```
bool airbot::AirbotSdk::MoveToJointPos ( const std::array< double, 6 > & joint_pos ) const
```

Moves the robotic arm to the specified joint positions.

Parameters

joint_pos An array containing the joint positions to move to.

Returns

True if the move operation is successful, false otherwise.

Moves the robotic arm to the specified joint positions. The joint positions must be provided as an array of doubles. The function returns true if the move operation is successful, otherwise false.

◆ MoveWithCartWaypoints()

```
bool  
airbot::AirbotSdk::MoveWithCartWaypoints ( const std::vector< std::pair< std::array< double, 3 >, std::array< double, 4 > > >
```

Moves the robotic arm along specified Cartesian waypoints.

Parameters

cart_waypoints A vector of waypoints represented as pairs of position (3D vector) and orientation (4D quaternion).

Returns

True if the move operation is successful, false otherwise.

Moves the robotic arm along a series of Cartesian waypoints. Each waypoint is represented by a pair of arrays: one for the position (x, y, z) and one for the orientation (quaternion: x, y, z, w). The function returns true if the move operation is successful, otherwise false.

◆ MoveWithJointWaypoints()

```
bool airbot::AirbotSdk::MoveWithJointWaypoints ( const std::vector< std::array< double, 6 > > & joint_waypoints ) const
```

Moves the robotic arm along specified joint waypoints.

Parameters

joint_waypoints A vector of joint positions to move through.

Returns

True if the move operation is successful, false otherwise.

Moves the robotic arm along a series of joint waypoints. Each waypoint is represented by an array of joint positions. The function returns true if the move operation is successful, otherwise false.

◆ ServoCartPos()

```
void  
airbot::AirbotSdk::ServoCartPos ( const std::pair< std::array< double, 3 >, std::array< double, 4 > > & cart_pos ) const
```

Directly controls the robotic arm's end-effector position.

Parameters

cart_pos A pair containing the target position (3D vector) and orientation (4D quaternion).

Directly controls the robotic arm's end-effector position. The target position and orientation must be provided as a pair of arrays. This function is used in servo mode.

◆ ServoCartTwist()

```
void airbot::AirbotSdk::ServoCartTwist ( const std::pair< std::array< double, 3 >, std::array< double, 3 > > & twist ) const
```

Directly controls the robotic arm's twist (linear and angular velocity) in Cartesian space.

Parameters

twist A pair containing the linear velocity (3D vector) and angular velocity (3D vector).

Directly controls the robotic arm's twist (linear and angular velocity) in Cartesian space. The target linear and angular velocities must be provided as a pair of arrays. This function is used in servo mode.

◆ ServoEefPos() [1/2]

```
void airbot::AirbotSdk::ServoEefPos ( const double & eef_pos ) const
```

Directly controls the robotic arm's end-effector position (single value).

Parameters

eef_pos The target position value for the end-effector.

Directly controls the robotic arm's end-effector position. The target position must be provided as a single double value. This function is used in servo mode.

◆ ServoEefPos() [2/2]

```
void airbot::AirbotSdk::ServoEefPos ( const std::vector< double > & eef_pos ) const
```

Directly controls the robotic arm's end-effector position.

Parameters

eef_pos A vector containing the target position of the end-effector.

Directly controls the robotic arm's end-effector position. The target position must be provided as a vector of doubles. This function is used in servo mode.

◆ ServoJointPos()

```
void airbot::AirbotSdk::ServoJointPos ( const std::array< double, 6 > & joint_pos ) const
```

Directly controls the robotic arm's joints to move to the specified positions.

Parameters

joint_pos An array containing the target joint positions.

Directly controls the robotic arm's joints to move to the specified positions. The target joint positions must be provided as an array of doubles. This function is used in servo mode.

◆ ServoJointVel()

```
void airbot::AirbotSdk::ServoJointVel ( const std::array< double, 6 > & joint_vel ) const
```

Directly controls the robotic arm's joints to move at the specified velocities.

Parameters

joint_vel An array containing the target joint velocities.

Directly controls the robotic arm's joints to move at the specified velocities. The target joint velocities must be provided as an array of doubles. This function is used in servo mode.

◆ SetParams()

```
bool  
airbot::AirbotSdk::SetParams ( const std::vector< std::pair< std::string, std::variant< int32_t, double, std::string, bool >
```

Sets the parameters of the robotic arm.

Parameters

params A vector of pairs containing parameter names and their new values.

Returns

True if the parameters are successfully set, false otherwise.

Sets the values of the specified parameters on the robotic arm. The parameters must be provided as a vector of pairs, where each pair contains the parameter name and its new value. The function will return true if all parameters are successfully set, otherwise false.

◆ SetSpeedProfile()

```
bool airbot::AirbotSdk::SetSpeedProfile ( const SpeedProfile & profile ) const
```

Sets the speed profile for the robotic arm.

Parameters

profile The speed profile to set.

Returns

True if the speed profile is successfully set, false otherwise.

Sets the speed profile of the robotic arm, which controls its movement speed and acceleration. Three speed profiles are available:

- `SpeedProfile.DEFAULT` : Default speed profile.
- `SpeedProfile.SLOW` : Slow speed profile.
- `SpeedProfile.FAST` : Fast speed profile. **Warning:** \ This speed profile is experimental and may cause the robot to move too fast. \ It is dangerous not to make sure the robot is in a safe environment. \ Use at your own risk.

◆ SwitchMode() [1/2]

```
bool airbot::AirbotSdk::SwitchMode ( const RobotMode & mode ) const
```

Switches the control mode of the robotic arm.

Parameters

mode The control mode to switch to.

Returns

True if the mode is successfully switched, false otherwise.

Switches the control mode of the robotic arm to the specified mode. The mode must be one of the following:

- `RobotMode.PLANNING_POS` : Planning mode. A single target is sent to the driver server, \ then a path is planned and executed.
- `RobotMode.SERVO_CART_TWIST` : Servo mode in cartesian space. The robot follows the cartesian \ twist commands.
- `RobotMode.SERVO_CART_POSE` : Servo mode in cartesian space. The robot follows the cartesian \ pose commands.
- `RobotMode.SERVO_JOINT_POS` : Servo mode in joint space. The robot follows the joint position \ commands.
- `RobotMode.SERVO_JOINT_VEL` : Servo mode in joint space. The robot follows the joint velocity \ commands.
- `RobotMode.GRAVITY_COMP` : Gravity compensation mode. The robot is in a free-drive state with \ gravity compensated.

◆ SwitchMode() [2/2]

```
bool airbot::AirbotSdk::SwitchMode ( const std::vector< std::string > & component_names,  
                                     const std::vector< RobotMode > & modes ) const
```

Switches the control mode for multiple robot components.

Parameters

component_names A vector of component names to switch mode for.

modes A vector of control modes corresponding to each component.

Returns

True if the modes are successfully switched, false otherwise.

Switches the control mode for multiple components of the robotic arm. Each component's mode is specified by the corresponding entry in the `modes` vector. The function returns true if all modes are successfully switched, otherwise false.

◆ UnloadApp()

```
bool airbot::AirbotSdk::UnloadApp ( ) const
```

Unloads the currently loaded application from the robotic arm.

Returns

True if the application is successfully unloaded, false otherwise.

Unloads the currently loaded application from the robotic arm. The function returns true if the application is successfully unloaded, otherwise false.

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

- [/builds/airbot/shipping/build/src/airbot/sdk_server/client/C++/include/airbot_sdk.hpp](#)