레이저 감쇠기
API

Compact High Contrast Ratio Laser Attenuator
API

User Manual

Target version – v3.0.X

HIL Lab. Inc.
603 포항지식산업센터
경상북도 포항시
전화: 054-261-2901
팩스: 054-261-2902
이메일: official@hillab.co.kr
홈페이지: www.hillab.co.kr

목차

1. 개요

본 라이브러리는 본사의 레이저 감쇠기(이하 디바이스)를 제어할 수 있는 API 를 제공합니다.
RS232 포트를 활용하여 디바이스와 통신합니다.

C++ 지원 플랫폼

C++ 버전	ISO C++ 14 이상
플랫폼	Windows 10 x86, Windows 10 x64 Windows 11 x86, Windows 11 x64
Compiler	Visual Studio 2019 (MSVC v142), Visual Studio 2022 (MSVC v143)

C 지원 플랫폼

C 표준	ISO C99 이상
플랫폼	Windows 10 x86, Windows 10 x64 Windows 11 x86, Windows 11 x64
Compiler	Visual Studio 2019 (MSVC v142), Visual Studio 2022 (MSVC v143) (MinGW, GCC 등은 별도 import library 필요, 제조사 문의 요망)

C# 지원 플랫폼

.NET 버전	.NET Framework 4.8
플랫폼	Windows 10 x64 Windows 11 x64
빌드 환경	Visual Studio 2019, Visual Studio 2022

2. API 특징

디자인

API 의 진입 지점(Entry Point)은 `AttenuatorHandler` 클래스이며, 선언된 하나의 객체는 하나의 디바이스와 연결될 수 있습니다. 디바이스의 연결 및 해제를 포함한 모든 제어는 상위 수준의 인터페이스로 제공됩니다. 그러므로 사용자는 PC 와 디바이스 간의 통신 및 프로토콜에 대한 정보를 알지 않아도 되며, 알 수도 없습니다. 아래 그림 1 은 디바이스 제어 인터페이스의 내부 파이프라인 모습을 보여줍니다.

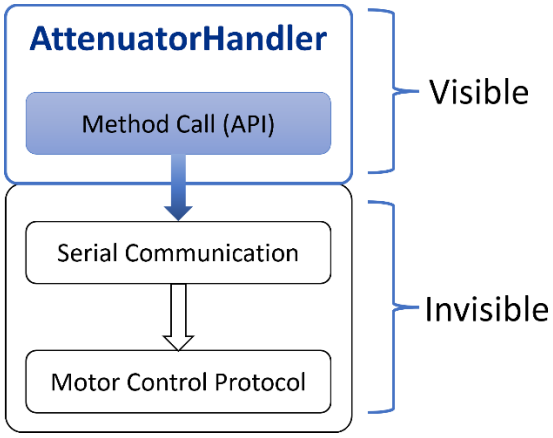

그림 1: 디바이스 제어 파이프라인

오류 추적

본 라이브러리의 디바이스 제어 파이프라인은 예외를 던지지 않도록 설계되었습니다. 대신 모든 메소드는 해당 명령 시행에 대한 피드백을 반환하며, 이 피드백으로 파이프라인의 어느 단계에서 문제가 발생하였는지 파악할 수 있습니다. 피드백은 `AttenuatorStatus` 열거형으로 정의되어 있으며, 파이프라인 단계별 피드백 종류는 아래 그림 2 와 같습니다.

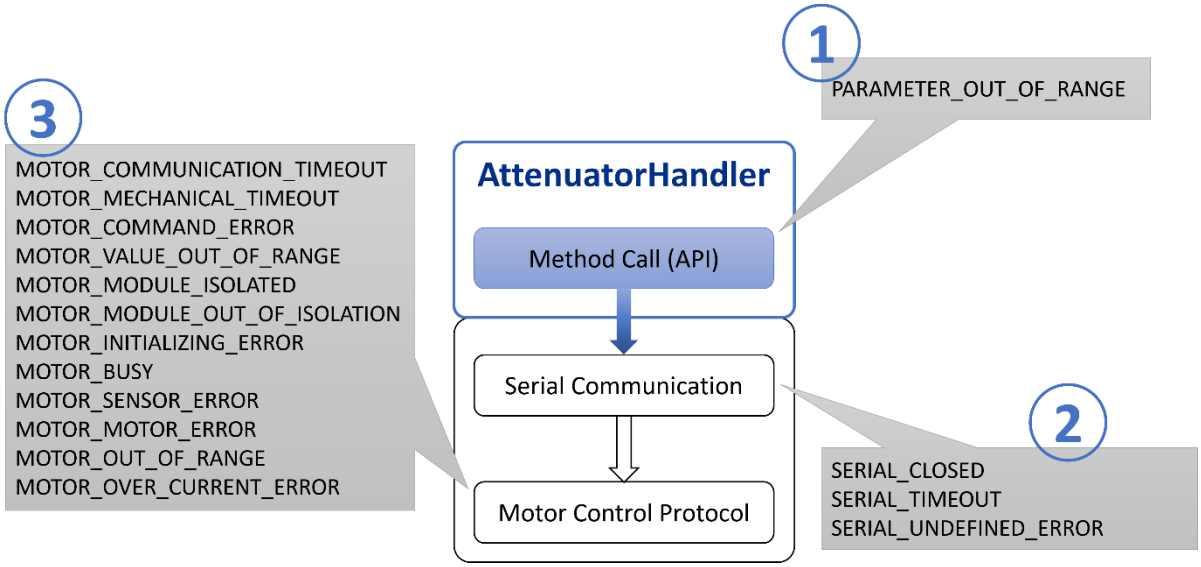

그림 2: 디바이스 제어 파이프라인 별 피드백 종류

비동기 제어

디바이스 제어 파이프라인에서, '시리얼 통신(Serial Communication) → 모터 제어 프로토콜(Motor Control Protocol)' 단계는 동기적으로 실행되나, 모터 제어 프로토콜이 완료된 후 실제 모터의 물리적 움직임은 비동기적으로 진행됩니다. 다시 말해, 시리얼 통신은 호출 스레드를 지연시킬 수 있으나, 시리얼 통신에 성공하면 스레드는 모터 동작의 완료를 기다리지 않습니다. 시리얼 통신의 지연은 `serial_timeout` 인자로 사전에 제어할 수 있으며, 모터 동작 가능 여부는 피드백(`AttenuatorStatus`)으로 확인할 수 있습니다.

한 편, 모든 메소드의 호출 내부에는 본 객체에 대한 상호 배제(mutex)가 적용되어 thread-safety를 제공합니다. 이는 하나의 디바이스에 대한 다중 명령 실행에서 순차적인 명령어 수행을 보장합니다. 가령, 아래 예제코드 1 에서 `t1` 의 작업이 먼저 실행된다면 파이프라인 진행 순서는 아래 그림 3 과 같습니다.

예제코드 1: 비동기 제어

```cpp
std::thread t1([&]() {
  handler.StartOptimization();
});

std::thread t2([&]() {
  handler.SetTransmittance(0.3, 0.0);
});
```

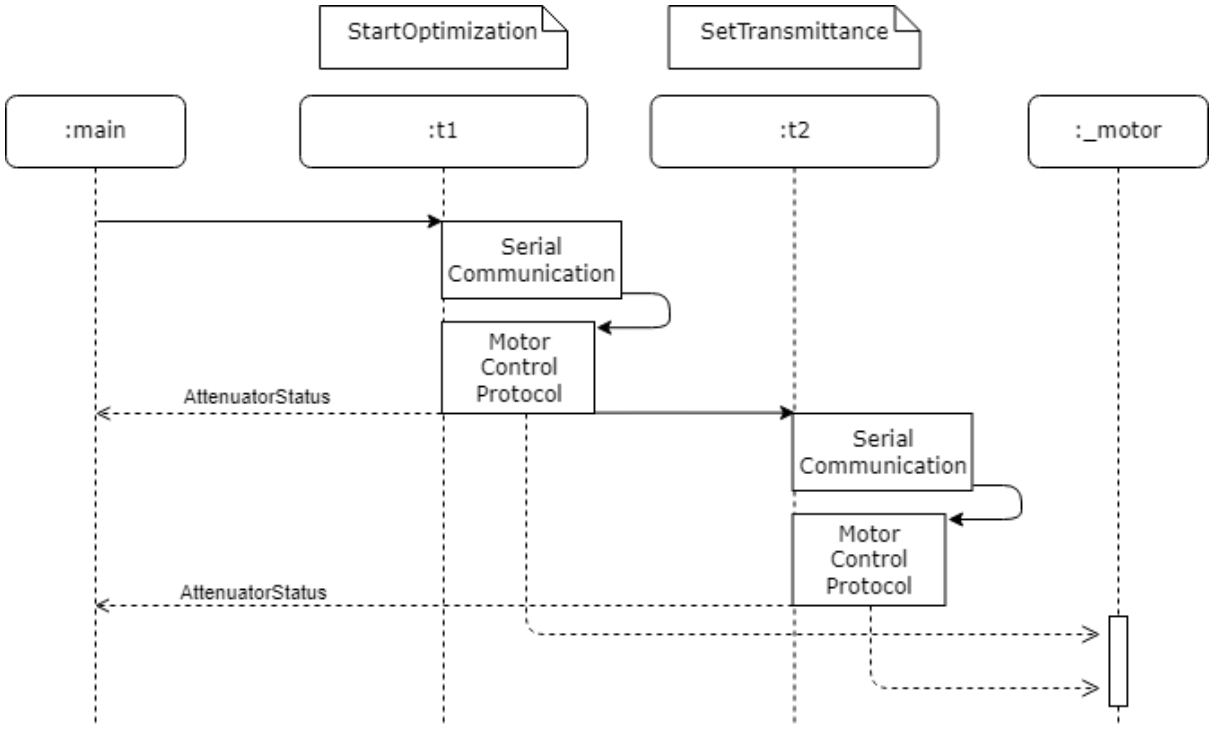

그림 3: 메소드 비동기 호출 예시

각 문맥 간의 '시리얼 통신 → 모터 제어 프로토콜' 은 실선 화살표 순서대로 실행됩니다. 즉 `t2` 스레드는 `t1` 의 모터 제어 프로토콜이 `AttenuatorStatus` 를 반환할 때까지 실행이 지연됩니다.

3. API 사용

3.1. 디바이스 연결 및 해제

하나의 AttenuatorHandler 객체는 하나의 디바이스와 연결될 수 있습니다. Connect 및 Disconnect 메소드를 통해 PC 와 디바이스 간의 연결을 관리합니다. PC 에서 식별할 수 있는 물리적 입/출력 단자의 이름으로 디바이스를 구분합니다(일반적으로 "COM").
만일 디바이스에 전원을 인가한 뒤 최초 연결이라면 MechanicalCalibration 을 호출하여 모터를 초기화하는 작업이 필요합니다. 이를 진행하지 않을 경우, 디바이스 제어가 정상적으로 이루어지지 않을 수 있습니다.

예제코드 2: 디바이스 연결 및 해제

```
HILLAB::AttenuatorHandler handler;
if (handler.Connect("COM8")) {
  handler.Connect("COM9"); // return false
  // Still "COM8" connected
}
handler.MechanicalCalibration();
handler.Disconnect();
```

3.2. 임의 각도 제어

디바이스의 회전모터는 모터 스텝 수(GetMotorSteps, SetMotorSteps) 혹은 각도 (GetMotorDegree, SetMotorDegree)를 통해 임의의 위치로 회전될 수 있습니다. 스텝 수와 각도는 모두 모터의 절대적인 위치를 의미합니다. 그러므로 만일 모터를 +15° 만큼 회전시키고 싶다면 현재 모터의 각도에 +15° 더한 값을 인자로 SetMotorDegree 를 호출해야 합니다.
모터의 각도와 스텝 수는 아래 표의 범위 내에서 비례 대응합니다. 가령, 현재 모터의 스텝 수가 71680 이라면, 이는 현재 모터의 회전각도가 180°임을 의미합니다.

단위	범위
절대각도	0° ~ 360°
스텝 수	0 ~ 143,360

예제코드 3: 임의 각도 제어

```
float degree;
i4t steps;

handler.GetMotorDegree(&degree); // degree ~ 0
handler.GetMotorSteps(&steps); // steps ~ 0

handler.SetMotorDegree(45.0);

handler.GetMotorDegree(&degree); // degree ~ 45
handler.GetMotorSteps(&steps); // steps ~ 17918
```

3.3. 빔의 목표 투과율을 이용한 각도 제어

디바이스의 회전모터는 목표로 하는 빔의 투과율(GetTransmittance, SetTransmittance)에 대응되는 위치로 회전될 수 있습니다. 투과율의 전체 범위는 회전각도 45° 에 대응되며, 회전각도에 따른 빔의 투과율은 아래 그림 4 와 같습니다.

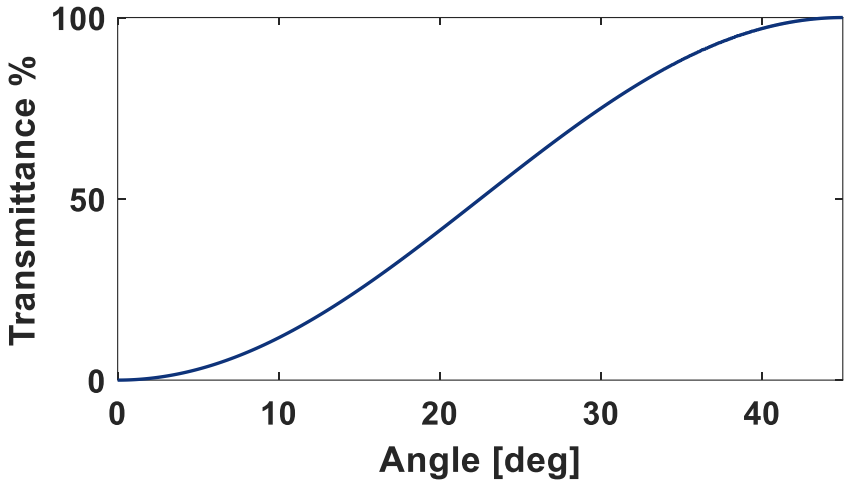

그림 4: 빔의 투과율과 회전모터의 범위

모터의 각도와 빔 투과율은 아래 표의 범위 내에서 비례 대응합니다. 이 때 투과율이 0%가 되는 기준각도는 x로 표기되었습니다.

단위	범위
절대각도	$(x+0)° \sim (x+45)°\ (<360°)$
빔 투과율	$0\% \sim 100\%$

가령, 기준각도 75°에서 50%의 투과율을 명령하면 디바이스는 절대각도 97.5°로 회전합니다.

기준 각도를 찾기 위한 사용자 캘리브레이션

빔의 목표 투과율을 이용한 각도 제어를 하기에 앞서, 우선 사용자는 모터를 임의의 각도로 회전시키며 빔의 투과율이 0%인 위치(기준각도)를 찾아야 합니다.

투과율에 대응하는 회전각도의 범위

빔 투과율에 대응하는 회전각도의 범위(45°) 또한 사용자가 직접 설정할 수 있으나, 본 디바이스는 기준각도로부터 45° 회전했을 때 투과율이 100%가 되도록 설계되었습니다.

예제코드 4: 빔의 목표 투과율을 이용한 각도 제어

```
float transmittance, degree;
double offset = 0.0; // User-tuned value

handler.SetTransmittance(0.0, offset);
handler.GetTransmittance(&transmittance, offset); // transmittance ~ 0%
```

```
handler.GetMotorDegree(&degree); // degree ~ 0

handler.SetTransmittance(0.5, offset);
handler.GetTransmittance(&transmittance, offset); // transmittance ~ 50%
handler.GetMotorDegree(&degree); // degree ~ 22.5

handler.SetTransmittance(1.0, offset);
handler.GetTransmittance(&transmittance, offset); // transmittance ~ 100%
handler.GetMotorDegree(&degree); // degree ~ 45
```

3.4. 디바이스 상태 모니터링

디바이스를 제어하는 모든 메소드는 디바이스의 상태를 나타내는 `AttenuatorStatus` 열거형 타입을 반환합니다. 디바이스 제어 파이프라인 수행 중 아무런 문제도 발생하지 않으면 `OK` 피드백을 반환하며, 문제가 발생하면 발생한 지점에서 수행을 중단하고 해당 오류를 즉시 피드백으로 반환합니다. 예를 들어, 아래 코드는 `handler` 객체를 디바이스에 연결하지 않았음에도 불구하고 그 사실을 `SetTransmittance` 메소드의 첫 번째 호출에서 알 수 없습니다. 인자의 범위가 허용 가능한 범위를 벗어났기에 호출 단계(Method Call)에서 파이프라인의 실행이 중단되기 때문입니다. (`SetTransmittance` 메소드의 인자 `transmittance` 의 범위는 0.0 ~ 1.0)

예제코드 5: 디바이스 상태 모니터링

```
AttenuatorStatus err;
handler.Disconnect();

err = handler.SetTransmittance(1.5, 0.0);
// err = AttenuatorStatus::PARAMETER_OUT_OF_RANGE

err = handler.SetTransmittance(0.5, 0.0);
// err = AttenuatorStatus::SERIAL_CLOSED
```

본 API 를 정상적으로 사용할 때 마주할 수 있는 피드백과 그 의미는 다음과 같습니다. 아래 외의 상태가 반복 발생한다면 기술 지원 문의 바랍니다.

AttenuatorStatus	Description
`PARAMETER_OUT_OF_RANGE`	사용자가 입력한 값이 가능한 범위를 벗어났습니다.
`SERIAL_CLOSED`	입출력 장치에 데이터를 쓸 수 없습니다.
`SERIAL_TIMEOUT`	입출력 장치로부터 데이터를 읽을 수 없습니다.
`MOTOR_BUSY`	디바이스가 다른 작업을 수행중입니다.
`OK`	아무런 오류도 발견되지 않았습니다.

타임아웃 한계

디바이스를 제어하는 모든 메소드는 시리얼 통신에 대한 타임아웃 인자를 제공하며, 이 시간동안 시리얼 통신이 응답하지 않을 경우 `SERIAL_TIMEOUT` 피드백을 반환합니다. 만약 이 타임아웃 시간이 모터 제어 프로토콜(Motor Control Protocol)의 수행시간보다 짧다면, 시리얼 통신에 문제가 없음에도 `SERIAL_TIMEOUT` 피드백이 반환될 가능성이 있습니다. 이를 방지하기 위해 모든 메소드의 기본 타임아웃 인자는 3500ms 로 설정되어 있습니다.

4. API References

4.1. HILLAB::AttenuatorHandler Class

4.1.1. AttenuatorHandler()

Default constructor.

<Possible Error Code>

- None

4.1.2. ~AttenuatorHandler()

Default destructor. It calls Disconnect() when this invoked.

<Possible Error Code>

- None

4.1.3. Connect()

Connects device via RS232 interface. USB adapter acts just as electrical adapter. If the device is already connected, this call returns false and keeps the old device connected.

	Type	Name	Description
in	`const std::string&`	device_port_name	Serial port. (e.g. "COM1")
return	`bool`		Result of connection.

<Possible Error Code>

- None

4.1.4. Disconnect()

Disconnects device.

<Possible Error Code>

- None

4.1.5. GetCurrentStatus()

Queries device status.

	Type	Name	Description
in	`const int`	serial_timeout	Serial communication timeout in milliseconds (default=3500).

<Possible Error Code>

- `AttenuatorStatus::OK`
- `AttenuatorStatus::SERIAL_XX`
- `AttenuatorStatus::MOTOR_XX`

4.1.6. GetMotorInfo()

Queries motor information.

	Type	Name	Description
out	`MotorInfo*`	out_info	Return value.
in	`const int`	serial_timeout	Serial communication timeout in milliseconds (default=7000).

<Possible Error Code>

- `AttenuatorStatus::OK`
- `AttenuatorStatus::SERIAL_XX`
- `AttenuatorStatus::MOTOR_XX`

4.1.7. MechanicalCalibration()

Start mechanical calibration. It should be called once after power is applied to the product.

	Type	Name	Description
in	`const int`	serial_timeout	Serial communication timeout in milliseconds (default=3500).

<Possible Error Code>

- `AttenuatorStatus::OK`
- `AttenuatorStatus::SERIAL_XX`
- `AttenuatorStatus::MOTOR_XX`

4.1.8. SetTransmittance()

Sets target transmittance value to run motor at the corresponding angle. The offset degree cannot be less than -360, and the sum of the offset degree and range degree cannot be more than 360.

	Type	Name	Description
in	`const float`	transmittance	Target transmittance value, range must be [0.0, 1.0].
in	`const double`	offset_degree	Motor angle in degree when the transmittance is 0.

in	`const double`	range_degree	Angle in degree converted to transmittance range of 0-100% (default=45.0). That is, when the angle of the motor is offset_degree + range_degree, the transmittance will be maximum.
in	`const int`	serial_timeout	Serial communication timeout in milliseconds (default=3500).

<Possible Error Code>

- `AttenuatorStatus::OK`
- `AttenuatorStatus::PARAMETER_OUT_OF_RANGE`
- `AttenuatorStatus::SERIAL_XX`
- `AttenuatorStatus::MOTOR_XX`

4.1.9. GetTransmittance()

Queries current transmittance value. The offset degree cannot be less than -360, and the sum of the offset degree and range cannot be more than 360.

	Type	Name	Description
out	`float*`	out_transmittance	Return value, range might be [0.0, 1.0].
in	`const double`	offset_degree	Motor angle in degree when the transmittance is 0.
in	`const double`	range_degree	Angle in degree converted to transmittance range of 0-100% (default=45.0). That is, when the angle of the motor is offset_degree + range_degree, the transmittance will be maximum.
in	`const int`	serial_timeout	Serial communication timeout in milliseconds (default=3500).

<Possible Error Code>

- `AttenuatorStatus::OK`
- `AttenuatorStatus::SERIAL_XX`
- `AttenuatorStatus::MOTOR_XX`

4.2.0. SetMotorSteps()

Sets target steps to run motor at the corresponding angle.

	Type	Name	Description

in	`const int`	steps	Target steps, range must be [-143360, 143360].
in	`const int`	serial_timeout	Serial communication timeout in milliseconds (default=3500).

<Possible Error Code>

- `AttenuatorStatus::OK`
- `AttenuatorStatus::PARAMETER_OUT_OF_RANGE`
- `AttenuatorStatus::SERIAL_XX`
- `AttenuatorStatus::MOTOR_XX`

4.2.1. GetMotorSteps()

Queries current steps.

	Type	Name	Description
out	`int*`	out_steps	Return value, range might be [-143360, 143360].
in	`const int`	serial_timeout	Serial communication timeout in milliseconds (default=3500).

<Possible Error Code>

- `AttenuatorStatus::OK`
- `AttenuatorStatus::SERIAL_XX`
- `AttenuatorStatus::MOTOR_XX`

4.2.2. SetMotorDegree()

Sets target degree to run motor at the corresponding angle.

	Type	Name	Description
in	`const float`	degree	Target degree, range must be (-360.0, 360.0).
in	`const int`	serial_timeout	Serial communication timeout in milliseconds (default=3500).

<Possible Error Code>

- `AttenuatorStatus::OK`
- `AttenuatorStatus::PARAMETER_OUT_OF_RANGE`
- `AttenuatorStatus::SERIAL_XX`
- `AttenuatorStatus::MOTOR_XX`

4.2.3. GetMotorDegree()

Queries current rotary angle of the motor.

	Type	Name	Description
out	`float*`	out_degree	Return value, range might be [-143360, 143360].
in	`const int`	serial_timeout	Serial communication timeout in milliseconds (default=3500).

<Possible Error Code>

- `AttenuatorStatus::OK`
- `AttenuatorStatus::SERIAL_XX`
- `AttenuatorStatus::MOTOR_XX`

4.2.4. StartOptimization()

Optimizes the operating frequencies for backward and forward movement by searching frequencies and monitoring mechanical performances. Allowing 20 minutes cool down period is highly recommended after optimization. This operation may take several minutes, during that time the device and the associated communication bus are blocked. In this case, AttenuatorStatus::MOTOR_BUSY is thrown when another method is called, except Stop().

	Type	Name	Description
in	`const int`	serial_timeout	Serial communication timeout in milliseconds (default=3500).

<Possible Error Code>

- `AttenuatorStatus::SERIAL_XX`
- `AttenuatorStatus::MOTOR_XX`

4.2.5. StartCleaningMechanics()

Starts cleaning cycle, during which the device will move backwards and forwards over its full range of travel for a few munutes, in order to remove any dust or debris from the bearing rails and motor contacts.Allowing 20 minutes cool down period is highly recommended after optimization. This operation may take several minutes, during that time the device and the associated communication bus are blocked. In this case, AttenuatorStatus::MOTOR_BUSY is thrown when another method is called, except Stop().

	Type	Name	Description
in	`const int`	serial_timeout	Serial communication timeout in milliseconds (default=3500).

<Possible Error Code>

- `AttenuatorStatus::SERIAL_XX`
- `AttenuatorStatus::MOTOR_XX`

4.2.6. Stop()

Aborts the optimization or cleaning process initiated by StartOpeimization() or StartCleaningMechanics().

	Type	Name	Description
in	`const int`	serial_timeout	Serial communication timeout in milliseconds (default=3500).

<Possible Error Code>

- `AttenuatorStatus::OK`
- `AttenuatorStatus::SERIAL_XX`
- `AttenuatorStatus::MOTOR_XX`

4.2. HILLAB::AttenuatorStatus Enum

Value	Description
PARAMETER_OUT_OF_RANGE	User input error
SERIAL_CLOSED	Unable to write bytes to serial port
SERIAL_TIMEOUT	Unable to receive bytes from serial port
SERIAL_UNDEFINED_ERROR	Unexpected serial error
OK	No problem
MOTOR_COMMUNICATION_TIMEOUT	Device internal error
MOTOR_MECHANICAL_TIMEOUT	Device internal error
MOTOR_COMMAND_ERROR	Device internal error
MOTOR_VALUE_OUT_OF_RANGE	Device internal error
MOTOR_MODULE_ISOLATED	Device internal error
MOTOR_MODULE_OUT_OF_ISOLATION	Device internal error
MOTOR_INITIALIZING_ERROR	Device internal error
MOTOR_THERMAL_ERROR	Device internal error
MOTOR_BUSY	Device internal error
MOTOR_SENSOR_ERROR	Device internal error
MOTOR_MOTOR_ERROR	Device internal error
MOTOR_OUT_OF_RANGE	Device internal error
MOTOR_OVER_CURRENT_ERROR	Device internal error

4.3. HILLAB::MotorInfo Struct

Type	Name	Description
`std::string`	serial_number	Motor serial number
`std::string`	year_of_manufacturing	Motor year of manufacturing
`std::string`	travel_degree	Motor travel degree
`std::string`	pulse_per_degree	Motor pulse per degree
`std::string`	firmware_release	Motor firmware release information
`std::string`	hardware_release	Motor hardware release information
`std::string`	mechanical_offset	Motor factory offset

전화: 054-261-2901
팩스: 054-261-2902
official@hillab.co.kr

603 포항지식산업센터
포항, 대한민국

www.hillab.co.kr