Capture

class **ingeniamotion.capture.Capture**(*motion_controller*)

Capture.

create_poller(*registers*, *servo='default'*, *sampling_time=0.125*, *buffer_size=100*, *start=True*)

Returns a Poller instance with target registers.

Parameters:

- **registers** (`List` [`Dict` [`str` , `Union` [`int` , `str`]]]) – list of registers to add to the Poller. Dicts should have the follow format:

```
[
    { # Poller register one
        "name": "CL_POS_FBK_VALUE",  # Register name.
        "axis": 1  # Register axis.
        # If it has no axis field, by default axis 1.
    },
    { # Poller register two
        "name": "CL_VEL_FBK_VALUE",  # Register name.
        "axis": 1  # Register axis.
        # If it has no axis field, by default axis 1.
    }
]
```

- **servo** (`str`) – servo alias to reference it. `default` by default.
- **sampling_time** (`float`) – period of the sampling in seconds. By default `0.125` seconds.
- **buffer_size** (`int`) – number maximum of sample for each data read. `100` by default.
- **start** (`bool`) – if `True` , function starts poller, if `False` poller should be started after. `True` by default.

Return type: `Poller`

Returns: Poller object with chosen registers.

Poller.start()

Poller starts reading the registers.

Poller.stop()

Poller stop reading the registers.

Poller.data

tuple with 3 items: a list of timestamp, list of lists of values (one list of values for each register), and a boolean that indicates if data was lost.

When the poller starts, the lists are filled with the timestamp and the value of the registers reading. The maximum length of the list will be buffer_size value, when this size is reached, the older value will be removed and the newest will be added.

When the property data is read list are reset to a empty list.

Raises:
- IMRegisterNotExist – If register does not exist in dictionary.
- TypeError – If some parameter has a wrong type.

`create_empty_monitoring(servo='default')`

Returns a Monitoring instance not configured.

Parameters: **servo** (`str`) – servo alias to reference it. `default` by default.

Return type: `Monitoring`

Returns: Not configured instance of monitoring.

Raises: NotImplementedError – If an wrong monitoring version is requested.

`create_monitoring(registers, prescaler, sample_time, trigger_delay=0, trigger_mode=<MonitoringSoCType.TRIGGER_EVENT_AUTO: 0>, trigger_config=None, trigger_signal=None, trigger_value=None, servo='default', start=False)`

Returns a Monitoring instance configured with target registers.

Parameters:

- **registers** (`List` [`Dict` [`str` , `Union` [`int` , `str`]]]) – list of registers to add to Monitoring. Dicts should have the follow format:

```
[
    { # Monitoring register one
        "name": "CL_POS_FBK_VALUE",  # Register name.
        "axis": 1  # Register axis.
        # If it has no axis field, by default axis 1.
    },
    { # Monitoring register two
        "name": "CL_VEL_FBK_VALUE",  # Register name.
        "axis": 1  # Register axis.
        # If it has no axis field, by default axis 1.
    }
]
```

- **prescaler** (`int`) – determines monitoring frequency. Frequency will be `Position & velocity loop rate frequency / prescaler` , see `ingeniamotion.configuration.Configuration.get_position_and_velocity_loop_rate()` to know about this frequency. It must be `1` or higher.
- **sample_time** (`float`) – sample time in seconds.
- **trigger_delay** (`float`) – trigger delay in seconds. Value should be between `-sample_time/2` and `sample_time/2` . `0` by default.
- **trigger_mode** (`MonitoringSoCType`) – monitoring start of condition type. `TRIGGER_EVENT_NONE` by default.
- **trigger_config** (`Optional` [`MonitoringSoCConfig`]) – monitoring edge condition. `None` by default.
- **trigger_signal** (`Optional` [`Dict` [`str` , `Union` [`int` , `str`]]]) – dict with name and axis of trigger signal for rising or falling edge trigger.
- **trigger_value** (`Union` [`float` , `int` , `None`]) – value for rising or falling edge trigger.
- **servo** (`str`) – servo alias to reference it. `default` by default.
- **start** (`bool`) – if `True` , function starts monitoring, if `False` monitoring should be started after. `False` by default.

Return type: `Monitoring`

Returns: Instance of monitoring configured.

Raises:
- ValueError – If prescaler is less than `1`.
- ValueError – If trigger_delay is not between `-total_time/2` and `total_time/2`.
- IMMonitoringError – If register maps fails in the servo.
- IMMonitoringError – If buffer size is not enough for all the registers and samples.
- IMMonitoringError – If trigger_mode is rising or falling edge trigger and trigger signal is not mapped.
- TypeError – If trigger_mode is rising or falling edge trigger and trigger_signal or trigger_value are None.

`create_disturbance(register, data, freq_divider, servo='default', axis=1, start=False)`

Returns a Disturbance instance configured with target registers.

Parameters:
- **register** (`str`) – target register UID.
- **data** (`Union` [`List` [`Union` [`int` , `float`]], `ndarray` [`Any` , `dtype` [`int32`]], `ndarray` [`Any` , `dtype` [`float64`]]]) – data to write in disturbance.
- **freq_divider** (`int`) – determines disturbance frequency divider. Frequency will be `Position & velocity loop rate frequency / freq_divider` , see `ingeniamotion.configuration.Configuration.get_position_and_velocity_loop_rate()` to know about this frequency. It must be `1` or higher.
- **servo** (`str`) – servo alias to reference it. `default` by default.
- **axis** (`int`) – servo axis. `1` by default.
- **start** (`bool`) – if `True` , function starts disturbance, if `False` disturbance should be started after. `False` by default.

Return type: `Disturbance`

Returns: Instance of disturbance configured.

Raises:
- ValueError – If freq_divider is less than `1`.
- IMDisturbanceError – If buffer size is not enough for all the registers and samples.

`enable_monitoring_disturbance(servo='default')`

Enable monitoring and disturbance.

Parameters: **servo** (`str`) – servo alias to reference it. `default` by default.

Raises: IMMonitoringError – If monitoring can't be enabled.

Return type: `None`

enable_monitoring(*servo='default'*)

Enable monitoring.

Parameters: **servo** (`str`) – servo alias to reference it. `default` by default.

Raises: IMMonitoringError – If monitoring can't be enabled.

Return type: `None`

enable_disturbance(*servo='default'*, *version=None*)

Enable disturbance.

Parameters:
- **servo** (`str`) – servo alias to reference it. `default` by default.
- **version** (`Optional` [`MonitoringVersion`]) – Monitoring/Disturbance version, if `None` reads from drive. `None` by default.

Raises: IMMonitoringError – If disturbance can't be enabled.

Return type: `None`

disable_monitoring_disturbance(*servo='default'*)

Disable monitoring and disturbance.

Parameters: **servo** (`str`) – servo alias to reference it. `default` by default.

Return type: `None`

disable_monitoring(*servo='default'*, *version=None*)

Disable monitoring.

Parameters:
- **servo** (`str`) – servo alias to reference it. `default` by default.
- **version** (`Optional` [`MonitoringVersion`]) – Monitoring/Disturbance version, if `None` reads from drive. `None` by default.

Return type: `None`

disable_disturbance(*servo='default'*, *version=None*)

Disable disturbance.

Parameters:
- **servo** (`str`) – servo alias to reference it. `default` by default.
- **version** (`Optional` [`MonitoringVersion`]) – Monitoring/Disturbance version, if `None` reads from drive. `None` by default.

Return type: `None`

`get_monitoring_disturbance_status(servo='default')`

Get Monitoring Status.

Parameters: **servo** (`str`) – servo alias to reference it. `default` by default.

Return type: `int`

Returns: Monitoring/Disturbance Status.

Raises:
- IMRegisterNotExist – If the register doesn't exist.
- TypeError – If some read value has a wrong type.

`get_monitoring_status(servo='default')`

Get Monitoring Status.

Parameters: **servo** (`str`) – servo alias to reference it. `default` by default.

Return type: `int`

Returns: Monitoring Status.

Raises:
- IMRegisterNotExist – If the register doesn't exist.
- TypeError – If some read value has a wrong type.

`get_disturbance_status(servo='default', version=None)`

Get Disturbance Status.

Parameters:
- **servo** (`str`) – servo alias to reference it. `default` by default.
- **version** (`Optional` [`MonitoringVersion`]) – Monitoring/Disturbance version, if `None` reads from drive. `None` by default.

Return type: `int`

Returns: Disturbance Status.

Raises:
- IMRegisterNotExist – If the register doesn't exist.
- TypeError – If some read value has a wrong type.

is_monitoring_enabled(*servo='default'*)

Check if monitoring is enabled.

Parameters: **servo** (`str`) – servo alias to reference it. `default` by default.

Return type: `bool`

Returns: True if monitoring is enabled, else False.

Raises: IMRegisterNotExist – If the register doesn't exist.

is_disturbance_enabled(*servo='default'*, *version=None*)

Check if disturbance is enabled.

Parameters:
- **servo** (`str`) – servo alias to reference it. `default` by default.
- **version** (`Optional` [`MonitoringVersion`]) – Monitoring/Disturbance version, if `None` reads from drive. `None` by default.

Return type: `bool`

Returns: True if disturbance is enabled, else False.

Raises: IMRegisterNotExist – If the register doesn't exist.

get_monitoring_process_stage(*servo='default'*, *version=None*)

Return monitoring process stage.

Parameters:
- **servo** (`str`) – servo alias to reference it. `default` by default.
- **version** (`Optional` [`MonitoringVersion`]) – Monitoring/Disturbance version, if `None` reads from drive. `None` by default.

Return type: `MonitoringProcessStage`

Returns: Current monitoring process stage.

Raises: IMRegisterNotExist – If the register doesn't exist.

is_frame_available(*servo='default'*, *version=None*)

Check if monitoring has an available frame.

Parameters:
- **servo** (`str`) – servo alias to reference it. `default` by default.
- **version** (`Optional` [`MonitoringVersion`]) – Monitoring/Disturbance version, if `None` reads from drive. `None` by default.

Return type: `bool`

Returns: True if monitoring has an available frame, else False.

Raises: IMRegisterNotExist – If the register doesn't exist.

`clean_monitoring(servo='default', version=None)`

Disable monitoring/disturbance and remove monitoring mapped registers.

Parameters:
- **servo** (`str`) – servo alias to reference it. `default` by default.
- **version** (`Optional` [`MonitoringVersion`]) – Monitoring/Disturbance version, if None reads from drive. `None` by default.

Return type: `None`

`clean_disturbance(servo='default', version=None)`

Disable monitoring/disturbance and remove disturbance mapped registers.

Parameters:
- **servo** (`str`) – servo alias to reference it. `default` by default.
- **version** (`Optional` [`MonitoringVersion`]) – Monitoring/Disturbance version, if None reads from drive. `None` by default.

Return type: `None`

`clean_monitoring_disturbance(servo='default')`

Disable monitoring/disturbance, remove disturbance and monitoring mapped registers.

Parameters: **servo** (`str`) – servo alias to reference it. `default` by default.

Return type: `None`

`mcb_synchronization(servo='default')`

Synchronize MCB, necessary to monitoring and disturbance. Motor must be disabled.

Parameters: **servo** (`str`) – servo alias to reference it. `default` by default.

Raises: IMStatusWordError – If motor is enabled.

Return type: `None`

`disturbance_max_sample_size(servo='default')`

Return disturbance max size, in bytes.

Parameters: **servo** (`str`) – servo alias to reference it. `default` by default.

Return type: `int`

Returns: Max buffer size in bytes.

Raises: TypeError – If some read value has a wrong type.

monitoring_max_sample_size(*servo='default'*)

Return monitoring max size, in bytes.

Parameters: **servo** (`str`) – servo alias to reference it. `default` by default.

Return type: `int`

Returns: Max buffer size in bytes.

Raises: TypeError – If some read value has a wrong type.

get_frequency(*servo='default'*, *axis=1*)

Returns the monitoring frequency.

Parameters:
- **servo** (`str`) – servo alias to reference it. `default` by default.
- **axis** (`int`) – servo axis. `1` by default.

Return type: `float`

Returns: Sampling rate in Hz.

Raises: TypeError – If some read value has a wrong type.