Capture Examples

Polling Example

```python
import argparse

import matplotlib.pyplot as plt
import matplotlib.animation as animation

from ingeniamotion import MotionController


def main(args):
    mc = MotionController()
    mc.communication.connect_servo_eoe(args.ip, args.dictionary_path)

    # List of registers
    registers = [
        {
            "name": "CL_POS_FBK_VALUE",  # Register name
            "axis": 1  # Register axis
        },
    ]
    # Create Poller with our registers
    poller = mc.capture.create_poller(registers)

    # PLOT DATA
    fig, ax = plt.subplots()
    global x_values, y_values
    x_values = []
    y_values = []
    line, = ax.plot(x_values, y_values)

    def animate(i):
        timestamp, registers_values, _ = poller.data  # Read Poller data
        global x_values, y_values
        x_values += timestamp
        y_values += registers_values[0]
        line.set_data(x_values, y_values)
        ax.relim()
        ax.autoscale_view()
        return line,

    ani = animation.FuncAnimation(
        fig, animate, interval=100)
    plt.autoscale()
    if args.close:
        plt.show(block=False)
        plt.pause(5)
        plt.close()
    else:
        plt.show()
    poller.stop()
    mc.communication.disconnect()


def setup_command():
    parser = argparse.ArgumentParser(description='Disturbance example')
    parser.add_argument('--dictionary_path', help='Path to drive dictionary', required=True)
    parser.add_argument('--ip', help='Drive IP address', required=True)
    parser.add_argument('--close', help='Close plot after 5 seconds', action='store_true')
    return parser.parse_args()
```

```
if __name__ == '__main__':
    args = setup_command()
    main(args)
```

Monitoring Example

```
import argparse
import logging
import numpy as np
import matplotlib.pyplot as plt

from ingeniamotion import MotionController
from ingeniamotion.enums import MonitoringSoCType, MonitoringSoCConfig

logging.basicConfig(level=logging.DEBUG)
logging.getLogger('matplotlib.font_manager').disabled = True


def main(args):
    mc = MotionController()
    mc.communication.connect_servo_eoe(args.ip, args.dictionary_path)

    # Monitoring registers
    registers = [{"axis": 1, "name": "CL_POS_FBK_VALUE"},
                 {"axis": 1, "name": "CL_VEL_FBK_VALUE"}]

    # Servo frequency divisor to set monitoring frequency
    monitoring_prescaler = 25

    total_time_s = 1  # Total sample time in seconds
    trigger_delay_s = 0.0  # Trigger delay time in seconds

    trigger_mode = MonitoringSoCType.TRIGGER_EVENT_AUTO
    # trigger_mode = MonitoringSoCType.TRIGGER_EVENT_EDGE
    trigger_config = None
    # trigger_config = MonitoringSoCConfig.TRIGGER_CONFIG_RISING
    # trigger_config = MonitoringSoCConfig.TRIGGER_CONFIG_FALLING

    # Trigger signal register if trigger_mode is TRIGGER_CYCLIC_RISING_EDGE or
TRIGGER_CYCLIC_FALLING_EDGE
    # else, it does nothing
    trigger_signal = {"axis": 1, "name": "CL_POS_FBK_VALUE"}
    # Trigger value if trigger_mode is TRIGGER_CYCLIC_RISING_EDGE or TRIGGER_CYCLIC_FALLING_EDGE
    # else, it does nothing
    trigger_value = 0

    monitoring = mc.capture.create_monitoring(registers,
                                              monitoring_prescaler,
                                              total_time_s,
                                              trigger_delay=trigger_delay_s,
                                              trigger_mode=trigger_mode,
                                              trigger_config=trigger_config,
                                              trigger_signal=trigger_signal,
                                              trigger_value=trigger_value)
    # Enable Monitoring
    mc.capture.enable_monitoring()
    print("Waiting for trigger")
    # Blocking function to read monitoring values
    data = monitoring.read_monitoring_data()
    print("Triggered and data read!")

    # Calculate abscissa values with total_time_s and trigger_delay_s
    x_start = -total_time_s/2 + trigger_delay_s
    x_end = total_time_s/2 + trigger_delay_s
    x_values = np.linspace(x_start, x_end, len(data[0]))
```

```
    # Plot result
    fig, axs = plt.subplots(2)
    for index in range(len(axs)):
        ax = axs[index]
        ax.plot(x_values, data[index])
        ax.set_title(registers[index]["name"])

    plt.autoscale()
    if args.close:
        plt.show(block=False)
        plt.pause(5)
        plt.close()
    else:
        plt.show()
    plt.show()

    mc.communication.disconnect()


def setup_command():
    parser = argparse.ArgumentParser(description='Disturbance example')
    parser.add_argument('--dictionary_path', help='Path to drive dictionary', required=True)
    parser.add_argument('--ip', help='Drive IP address', required=True)
    parser.add_argument('--close', help='Close plot after 5 seconds', action='store_true')
    return parser.parse_args()


if __name__ == '__main__':
    args = setup_command()
    main(args)
```

Disturbance Example

```python
import math
import time
import argparse

from ingeniamotion import MotionController
from ingeniamotion.enums import OperationMode


def main(args):
    mc = MotionController()
    mc.communication.connect_servo_eoe(args.ip, args.dictionary_path)

    # Disturbance register
    target_register = "CL_POS_SET_POINT_VALUE"
    # Frequency divider to set disturbance frequency
    divider = 25
    # Calculate time between disturbance samples
    sample_period = divider / mc.configuration.get_position_and_velocity_loop_rate()
    # The disturbance signal will be a simple harmonic motion (SHM) with frequency 0.5Hz and 2000
counts of amplitude
    signal_frequency = 0.5
    signal_amplitude = 2000
    # Calculate number of samples to load a complete oscillation
    n_samples = int(1 / (signal_frequency * sample_period))
    # Generate a SHM with the formula x(t)=A*sin(t*w) where:
    # A = signal_amplitude (Amplitude)
    # t = sample_period*i (time)
    # w = signal_frequency*2*math.pi (angular frequency)
    data = [int(signal_amplitude * math.sin(sample_period * i * signal_frequency * 2 * math.pi))
            for i in range(n_samples)]

    # Call function create_disturbance to configure a disturbance
    dist = mc.capture.create_disturbance(target_register, data, divider)

    # Set profile position operation mode and enable motor to enable motor move
    mc.motion.set_operation_mode(OperationMode.PROFILE_POSITION)
    # Enable disturbance
    mc.capture.enable_disturbance()
    # Enable motor
    mc.motion.motor_enable()
    # Wait 10 seconds
    time.sleep(10)
    # Disable motor
    mc.motion.motor_disable()
    # Disable disturbance
    mc.capture.clean_disturbance()
    mc.communication.disconnect()


def setup_command():
    parser = argparse.ArgumentParser(description='Disturbance example')
    parser.add_argument('--dictionary_path', help='Path to drive dictionary', required=True)
    parser.add_argument('--ip', help='Drive IP address', required=True)
    return parser.parse_args()


if __name__ == '__main__':
    args = setup_command()
    main(args)
```

Polling using PDOs Example

```python
import time
from typing import Dict, List, Union

from ingeniamotion.enums import SensorType
from ingeniamotion.motion_controller import MotionController


def set_up_pdo_poller(mc: MotionController) -> None:
    """Set-up a PDO poller.

    Read the Actual Position and the Actual Velocity registers using the PDO poller.

    Args:
        mc: The controller where there are all functions to perform a PDO poller.
    """
    registers: List[Dict[str, Union[int, str]]] = [
        {
            "name": "CL_POS_FBK_VALUE",
            "axis": 1,
        },
        {
            "name": "CL_VEL_FBK_VALUE",
            "axis": 1,
        },
    ]

    poller = mc.capture.pdo.create_poller(registers)
    # Waiting time for generating new samples
    time.sleep(1)
    time_stamps, data = poller.data
    poller.stop()

    print(f"Time: {time_stamps}")
    print(f"Actual Position values: {data[0]}")
    print(f"Actual Velocity values: {data[1]}")


def main() -> None:
    mc = MotionController()
    # Modify these parameters to connect a drive
    interface_ip = "192.168.2.1"
    slave_id = 1
    dictionary_path = "parent_directory/dictionary_file.xdf"
    mc.communication.connect_servo_ethercat_interface_ip(
        interface_ip, slave_id, dictionary_path
    )
    set_up_pdo_poller(mc)
    mc.communication.disconnect()
    print("The drive has been disconnected.")


if __name__ == "__main__":
    main()
```

Process Data object Example

```
import time
from functools import partial

from ingenialink.pdo import RPDOMapItem, TPDOMapItem

from ingeniamotion.motion_controller import MotionController


def notify_actual_value(actual_position: TPDOMapItem) -> None:
    """Callback that is subscribed to get the actual position for cycle.

    Args:
        actual_position: TPDO mapped to the Actual position register.

    """
    print(f"Actual Position: {actual_position.value}")


def update_position_set_point(position_set_point: RPDOMapItem) -> None:
    """Callback to update the position set point value for each cycle.

    Args:
        position_set_point: RPDO mapped to the Position set-point register.
    """
    position_set_point.value += 100
    print(f"Position set-point: {position_set_point.value}")


def update_position_value_using_pdo(mc: MotionController) -> None:
    """Updates the position of a motor using PDOs.

    Args:
        mc : Controller with all the functions needed to perform a PDO exchange.
    """
    waiting_time_for_pdo_exchange = 5
    # Create a RPDO map item
    initial_position_value = mc.motion.get_actual_position()
    position_set_point = mc.capture.pdo.create_pdo_item(
        "CL_POS_SET_POINT_VALUE", value=initial_position_value
    )
    # Create a TPDO map item
    actual_position = mc.capture.pdo.create_pdo_item("CL_POS_FBK_VALUE")
    # Create the RPDO and TPDO maps
    rpdo_map, tpdo_map = mc.capture.pdo.create_pdo_maps([position_set_point], [actual_position])
    # Callbacks subscriptions for TPDO and RPDO map items
    mc.capture.pdo.subscribe_to_receive_process_data(partial(notify_actual_value,
actual_position))
    mc.capture.pdo.subscribe_to_send_process_data(
        partial(update_position_set_point, position_set_point)
    )
    # Map the PDO maps to the slave
    mc.capture.pdo.set_pdo_maps_to_slave(rpdo_map, tpdo_map)
    # Start the PDO exchange
    # Make sure to set an appropriate refresh rate considering the execution time of the send and
    # receive process data callbacks.
    mc.capture.pdo.start_pdos(refresh_rate=0.1)
    time.sleep(waiting_time_for_pdo_exchange)
    # Stop the PDO exchange
    mc.capture.pdo.stop_pdos()
```

```python
def main() -> None:
    # Before running this example, the drive has to be configured for a your motor.
    mc = MotionController()

    # Modify these parameters to connect a drive
    interface_ip = "192.168.2.1"
    slave_id = 1
    dictionary_path = "parent_directory/dictionary_file.xdf"
    mc.communication.connect_servo_ethercat_interface_ip(interface_ip, slave_id, dictionary_path)
    print("Drive is connected.")
    mc.motion.motor_enable()
    print("Motor is enabled.")
    update_position_value_using_pdo(mc)
    mc.motion.motor_disable()
    print("Motor is disabled.")
    mc.communication.disconnect()
    print("The drive has been disconnected.")


if __name__ == "__main__":
    main()
```