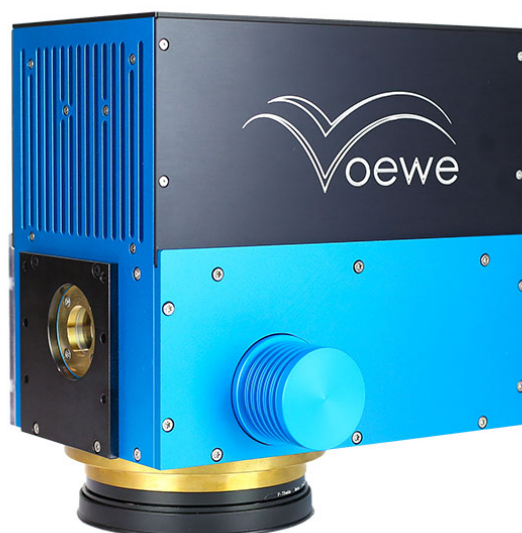


Software Manual for MOEWE Polygon Scanner Library

Johannes Thoms

July 8, 2024



 太平貿易株式会社

光学機器課

03-3270-4826 tokyo@taiheiboeki.co.jp

CONTENTS

1	Introduction	3
2	Usage	3
3	Function References	3
4	Code Examples	20
4.1	C++ Code Example	20

1 INTRODUCTION

Welcome to the Polygon Scanner Software Library Manual. This comprehensive guide is designed to help you harness the full potential of our advanced scanning technology. Whether you're a seasoned developer or a researcher, this manual will provide you with the detailed information and practical insights needed to effectively integrate and utilize our software library in your projects.

2 USAGE

In our Polygon Scanner Software Library, we provide two versions of the Dynamic Link Library (DLL) to cater to different project requirements and development environments. These versions are:

- **Managed Version (MoewePS.dll):** This version of the DLL is designed for use in managed code environments. It is ideal for applications developed using .NET languages such as C# or VB.NET.
- **Unmanaged Version (MoewePS_U.lib):** This version is tailored for unmanaged code environments and can be easily included in e.g. C++ projects. Please note, that this version still needs the MoewePS.dll and its associated files must be located in the output directory of your application.

3 FUNCTION REFERENCES

This section provides a comprehensive reference guide to on each function, including descriptions, parameters, and return values available in the MoewePS_U.dll. The functions in the managed MoewePS.dll are named analogously but are capitalized for clarity and consistency.

- **void bmpDataStream(const char* filename):**
 - **Description:** Starts an asynchronous thread which sends all the bitmaps in a specified folder to the scanner in alphanumeric order. To prevent unprocessed data from being overwritten, the internal read pointer of the ARM is constantly monitored.
 - **Parameter:**
 - * **folder:** Folder which contains the bitmaps.
- **void clearCorrectionFile():**
 - **Description:** Resets the optical x-axis correction.
- **void connect(const char* ip, bool isPollingActive, int timeout):**
 - **Description:** Establishes a TCP connection to a scanner.
 - **Parameters:**
 - * **ip:** IP of the scanner.
 - * **isPollingActive:** If true, all values are constantly synchronized between the PolygonScanner.Configuration class and the scanner. In this case, it's not necessary to call the SendConfig(PolygonScanner.Configuration) and LoadConfig(PolygonScanner.Configuration) routines manually.
 - * **timeout:** Connection watchdog timeout.
- **void disconnect():**
 - **Description:** Interrupts the TCP connection.
- **void enableEngraving(bool enable):**
 - **Description:** Enables engraving of the bitmap currently loaded to the data field.
 - **Parameter:**
 - * **enable:** true activates engraving, false deactivates.

- **void enableMotor(bool enable):**
 - **Description:** Enables the motor of the polygon.
 - **Parameter:**
 - * **enable:** true to enable, false to disable.
- **void enableOpticalXCorrection(bool enable):**
 - **Description:** Enables the usage of optical x-correction.
 - **Parameter:**
 - * **enable:** True activates optical correction.
- **int getAnalogLaserOutput(bool getValue1):**
 - **Description:** Getter for the analog laser output.
 - **Parameter:**
 - * **getValue1:** true for value 1, false for value 2.
 - **Returns:** Analog output value.
- **unsigned char getAxisInput():**
 - **Description:** Getter for the axis input.
 - **Returns:** A byte where the first four bits represent the input signals from the axis. All other bits are unused.
- **int getCopyX():**
 - **Description:** Gets the number of repetitions of the data pattern in the x-direction. Attention: Returns 0, if CopyX is set to an infinite number of repetitions.
 - **Returns:** Number of data pattern repetitions in the x-direction.
- **int getCopyY():**
 - **Description:** Gets the number of repetitions of the data pattern in y-direction. Attention: Returns 0, if CopyY is set to an infinite number of repetitions.
 - **Returns:** No. of data pattern repetitions in y-direction.
- **int getCurrentConfigSlot():**
 - **Description:** Gets the currently active internal configuration slot.
 - **Returns:** config slot [0..3].
- **int getCurrentFacet():**
 - **Description:** Getter for the facet which is currently in use.
 - **Returns:** Facet [1..8].
- **int getCurrentObjectCount():**
 - **Description:** Getter for the number of currently processed objects.
 - **Returns:** Number of currently processed objects.
- **double getCurrentScannerPositionX():**
 - **Description:** Getter for the current x-position of the scanner.
 - **Returns:** Current x-position of the scanner in mm relative to the scanfield.

- **double getCurrentScannerPositionY():**
 - **Description:** Getter for the current y-position of the scanner.
 - **Returns:** Current y-position of the scanner in mm relative to the scanfield.
- **double GetCurrentShift(MoewePS_U::Axis axis):**
 - **Description:** Getter for the current offset due to shifter events.
 - **Parameter:**
 - * axis: Axis.
 - **Returns:** Current shift in μm .
- **double getDatafieldDepth():**
 - **Description:** Getter for the data fields depth.
 - **Returns:** Data fields depth.
- **double getDatafieldHeight():**
 - **Description:** Getter for the data fields height.
 - **Returns:** Data fields height.
- **double getDatafieldWidth():**
 - **Description:** Getter for the data fields width.
 - **Returns:** Data fields width.
- **double getDatafieldX():**
 - **Description:** Getter for the location of the data fields lower left corner on the x-axis.
 - **Returns:** Location of the data fields lower left corner on the x-axis.
- **double getDatafieldY():**
 - **Description:** Getter for the location of the data fields lower left corner on the y-axis.
 - **Returns:** Location of the data fields lower left corner on the y-axis.
- **double getDatafieldZ():**
 - **Description:** Getter for the start position on the z-axis.
 - **Returns:** Start position on the z-axis.
- **unsigned char getDigitalLaserOutput():**
 - **Description:** Getter for the digital laser output.
 - **Returns:** A byte where the first four bits represent D0; D1; D2 and the trigger signal. The other bits are unused.
- **int getDroppedLines():**
 - **Description:** Getter for the number of dropped lines.
 - **Returns:** Dropped lines.
- **unsigned char getEnabledFacets():**
 - **Description:** Gets the enabled state of each facet as a byte.
 - **Returns:** A byte where each bit represents the enabled state e.g. 0x0F, if facets 5-8 are disabled.

- **unsigned char getEncoderInput(Axis axis):**
 - **Description:** Getter for the encoder input values.
 - **Parameter:**
 - * **axis:** Selects the encoder axis.
 - **Returns:** A byte where the first three bits represent the encoders A-, B- and Interrupt signal. All other bits are unused.
- **int getFacetCorrectionPhi(int facet):**
 - **Description:** Gets the facet correction value phi (d) of a given facet.
 - **Parameter:**
 - * **facet:** Number of the facet (1-8).
 - **Returns:** Facet correction value.
- **int getFacetCorrectionX(int facet):**
 - **Description:** Gets the horizontal facet correction value (dX) of a given facet.
 - **Parameter:**
 - * **facet:** Number of the facet (1-8).
 - **Returns:** Facet correction value.
- **int getFacetCorrectionY(int facet):**
 - **Description:** Gets the vertical facet correction value (dY) of a given facet.
 - **Parameter:**
 - * **facet:** Number of the facet (1-8).
 - **Returns:** Facet correction value.
- **double getGalvoDelay():**
 - **Description:** Returns the delay of the galvo.
 - **Returns:** Delay in μ s.
- **double getGalvoHysteresis():**
 - **Description:** Returns the hysteresis of the galvo.
 - **Returns:** Hysteresis in mm.
- **double getGalvoLeadTime():**
 - **Description:** Getter for the lead time of the galvoscanner.
 - **Returns:** Lead time of the galvoscanner in μ s.
- **unsigned int getGlobalXShift():**
 - **Description:** Getter for the index offset of the galvo (x offset).
 - **Returns:** X offset.
- **unsigned int getGlobalYShift():**
 - **Description:** Getter for the index offset of the polygon (y offset).
 - **Returns:** Y offset.
- **double getHatch():**

- **Description:** Getter for the distance between the separate laser tracks in μm .
- **Returns:** Distance between the separate laser tracks in μm .
- **bool getIOPin(int pin, bool isInput):**
 - **Description:** Getter for the status of the GPIO-Pins.
 - **Parameters:**
 - * **pin:** Number of the Pin (0..7).
 - * **isInput:** Set to true for input pins or to false for output.
 - **Returns:** true, if the pin is on high level.
- **unsigned char getIOPins(bool isInput):**
 - **Description:** Getter for the status of the GPIO-Pins.
 - **Parameter:**
 - * **isInput:** Set to true for input pins or to false for output.
 - **Returns:** A byte where each bit represents one of the 8 inputs.
- **double getLaserOnDelay():**
 - **Description:** Getter for the laser on delay.
 - **Returns:** Laser on delay in μs .
- **unsigned char getLEDState():**
 - **Description:** Getter for the status of the ethernet LED.
 - **Returns:** A byte where the first three bits represent the state of the orange, green, and yellow ethernet status LEDs. The other bits are unused.
- **int getLineCount():**
 - **Description:** Gets the number of line repetitions.
 - **Returns:** No. of line repetitions.
- **double getLineFrequency():**
 - **Description:** Getter for the line frequency.
 - **Returns:** Line frequency in Hz.
- **double getMicronsPerBitX():**
 - **Description:** Getter for the optical resolution in x-direction.
 - **Returns:** $\mu\text{m}/\text{Bit}$ in x-direction.
- **double getMicronsPerBitY():**
 - **Description:** Getter for the optical resolution in y-direction.
 - **Returns:** $\mu\text{m}/\text{Bit}$ in y-direction.
- **double getMicronsPerBitZ():**
 - **Description:** Getter for the optical resolution in z-direction.
 - **Returns:** $\mu\text{m}/\text{Bit}$ in z-direction.
- **unsigned char getMotorInput(Axis axis):**
 - **Description:** Getter for the motor IO values.

- **Parameters:**
 - * **axis:** Selects the encoder axis.
- **Returns:** A byte where the first four bits represent the motors enable-, direction-, step- and limit switch-signal. All other bits are unused.
- **unsigned char getNegatedIOPins(bool isInput):**
 - **Description:** Getter for the negated status of the GPIO-Pins.
 - **Parameters:**
 - * **isInput:** Set to true for input pins or to false for output.
 - **Returns:** A byte where a bit is active if the corresponding input is negated.
- **int getObjectCount():**
 - **Description:** Gets the number of object repetitions.
 - **Returns:** No. of object repetitions.
- **double getObjectDepth():**
 - **Description:** Getter for the objects depth.
 - **Returns:** Objects depth.
- **double getObjectHeight():**
 - **Description:** Getter for the objects height.
 - **Returns:** Objects height.
- **double getObjectWidth():**
 - **Description:** Getter for the objects width.
 - **Returns:** Objects width.
- **double getObjectX():**
 - **Description:** Getter for the location of the objects lower left corner on the x-axis.
 - **Returns:** Location of the objects lower left corner on the x-axis.
- **double getObjectY():**
 - **Description:** Getter for the location of the objects lower left corner on the y-axis.
 - **Returns:** Location of the objects lower left corner on the y-axis.
- **double getObjectZ():**
 - **Description:** Gets the start position on the z axis for the objects.
 - **Returns:** Start position of the objects on the z-axis.
- **OutputMode getOutputMode():**
 - **Description:** Getter for the number of the output Mode.
 - **Returns:** Offset by which the lines are shifted after each object repetition.
- **int getProcessedLines():**
 - **Description:** Getter for the number of totally processed lines.
 - **Returns:** Totally processed lines.

- **double getRoundsPerMinute():**
 - **Description:** Getter for the rounds per minute.
 - **Returns:** Rounds per minute.
- **double getScanfieldDepth():**
 - **Description:** Getter for the depth of the scanfield.
 - **Returns:** Scanfield depth.
- **double getScanfieldHeight():**
 - **Description:** Getter for the scanfields height.
 - **Returns:** Scanfields height.
- **double getScanfieldWidth():**
 - **Description:** Getter for the scanfields width.
 - **Returns:** Scanfields width.
- **double getScanfieldX():**
 - **Description:** Getter for the location of the scanfields lower left corner on the x-axis.
 - **Returns:** Location of the scanfields lower left corner on the x-axis.
- **double getScanfieldY():**
 - **Description:** Getter for the location of the scanfields lower left corner on the y-axis.
 - **Returns:** Location of the scanfields lower left corner on the y-axis.
- **double getScanfieldZ():**
 - **Description:** Getter for the location of the scanfields start on the z-axis.
 - **Returns:** Scanfield start on the z-axis.
- **double getScanSpeed():**
 - **Description:** Getter for the scan speed.
 - **Returns:** Scan speed in m/s.
- **unsigned char getShifterOptions(Axis axis, ShiftType shiftType):**
 - **Description:** Gets the shifter options as a byte for a certain axis and shift type.
 - **Parameters:**
 - * **axis:** Axis enum.
 - * **shiftType:** Type (Set, Reset, Direction, Out).
 - **Returns:** Shifter options as a byte.
- **double getShiftValue(Axis axis, ShiftValue shiftValue):**
 - **Description:** Returns one of the two shift values or the reset value for a given axis.
 - **Parameters:**
 - * **axis:** Axis enum.
 - * **shiftValue:** Enum to select the specific value that should be set.
 - **Returns:** Shift value in:
 - * μm (Value1 and Value2)

- * mm (ResetValue)
- * steps (SetEach and ResetEach)
- * μ s (SetTimer and ResetTimer)
- **double getSpaceX():**
 - **Description:** Getter for the space between the data patterns in x-direction in case repetition is on.
 - **Returns:** Horizontal space between the data patterns.
- **double getSpaceY():**
 - **Description:** Getter for the space between the data patterns in y-direction in case repetition is on.
 - **Returns:** Vertical space between the data patterns.
- **double getSubhatch():**
 - **Description:** Getter for the offset by which the lines are shifted after each object repetition in μ m.
 - **Returns:** Offset by which the lines are shifted after each object repetition.
- **int getThreshold():**
 - **Description:** Getter for the threshold for various modifiers e.g. pulse width modulation.
 - **Returns:** Threshold value.
- **double getVelocity():**
 - **Description:** Getter for the velocity of the fast axis.
 - **Returns:** The velocity of the fast axis in m/s.
- **double getVelocityRPM():**
 - **Description:** Getter for the velocity of the fast axis in rounds per minute.
 - **Returns:** The velocity of the fast axis in rpm.
- **int getXCorrectionScaling():**
 - **Description:** Returns the currently active scaling used for the optical x-correction.
 - **Returns:** Scaling value (1, 2, 4, or 8).
- **void invertBmp(bool invert):**
 - **Description:** Inverts the uploaded bitmap.
 - **Parameters:**
 - * invert: True to invert, false for normal processing.
- **void invertX(bool invert):**
 - **Description:** Inverts the count direction of the galvanometer for the x-axis.
 - **Parameters:**
 - * invert: If true, the galvanometer counts in the positive direction.
- **void invertY(bool invert):**
 - **Description:** Inverts the count direction of the polygon motor for the y-axis.

- **Parameters:**
 - * **invert:** If true, the polygon motor counts in the positive direction.
- **bool isBitmapInverted():**
 - **Description:** Determines if bitmap is inverted.
 - **Returns:** True if the bitmap is inverted.
- **bool isConnected():**
 - **Description:** Determines if the scanner is currently connected.
 - **Returns:** True if the scanner is connected.
- **bool isDataLoaded():**
 - **Description:** Getter for data-loaded-flag.
 - **Returns:** True if the data is loaded.
- **bool isValidData():**
 - **Description:** Getter for data-valid-flag.
 - **Returns:** True if the data is valid.
- **bool isEncoderOk():**
 - **Description:** Getter for encoder-ok-flag.
 - **Returns:** True if the encoder works.
- **bool isEngravingEnabled():**
 - **Description:** Determines if engraving of the bitmap is active.
 - **Returns:** True if the bitmap is engraved.
- **bool isGalvoInPosition():**
 - **Description:** Getter for galvo-in-position-flag.
 - **Returns:** True if the galvo is in position.
- **bool isGalvoXOk():**
 - **Description:** Getter for galvo-x-flag.
 - **Returns:** True if the galvo for the x-axis is okay.
- **bool isGalvoXPowerOn():**
 - **Description:** Getter for y-axis galvo power on and enabled flag.
 - **Returns:** True if the y-axis galvo's power is on.
- **bool isGalvoYOk():**
 - **Description:** Getter for galvo-y-flag.
 - **Returns:** True if the galvo for the y-axis is okay.
- **bool isGalvoYPowerOn():**
 - **Description:** Getter for y-axis galvo power on and enabled flag.
 - **Returns:** True if the y-axis galvo's power is on.
- **bool isIOPinNegated(int pin, bool isInput):**

- **Description:** Determines if a certain IO-Pin is negated.
- **Parameters:**
 - * **pin:** Number of the IO-pin.
 - * **isInput:** True for input; false for output.
- **Returns:** True, if pin is negated.
- **bool isMotorEnabled():**
 - **Description:** Determines if the polygon motor is enabled.
 - **Returns:** True, if motor movement is enabled.
- **bool isOpticalXCorrectionEnabled():**
 - **Description:** Determines if the correction file is used to adjust x movement.
 - **Returns:** True, if optical correction is on.
- **bool isPaused():**
 - **Description:** Getter for the process paused flag.
 - **Returns:** True, if the process is halted.
- **bool isPositionValid():**
 - **Description:** Getter for position-valid-flag.
 - **Returns:** True, if the position is valid.
- **bool isProcessRunning():**
 - **Description:** Getter for process-running-flag.
 - **Returns:** True, if the process is running.
- **bool isRotationSynchronized():**
 - **Description:** Getter for rotation-synchronized-flag.
 - **Returns:** True, if the rotation is synchronized.
- **bool isSecondaryScannerSynchronized():**
 - **Description:** Getter for secondary-scanner-synchronized-flag.
 - **Returns:** True, if the secondary scanner is synchronized.
- **bool isSimulationModeActive():**
 - **Description:** Determines if the simulation mode is active.
 - **Returns:** True, if motor movement is simulated.
- **bool isXInverted():**
 - **Description:** Determines if the galvanometer direction is inverted.
 - **Returns:** True, if the galvanometer counts in positive direction.
- **bool isYInverted():**
 - **Description:** Determines if the polygon motor direction is inverted.
 - **Returns:** True, if the polygon motor counts in positive direction.
- **void loadConfig():**

- **Description:** Loads the current configuration of the active configuration slot from the connected scanner to the PolygonScanner.Configuration (only necessary if no auto polling is active).
- **void loadConfig(const char* fileName):**
 - **Description:** Loads the configuration from an XML or binary file.
 - **Parameters:**
 - * **fileName:** Name of the file.
- **void negateIOPin(int pin, bool isInput, bool negate):**
 - **Description:** Negates or affirms a certain pin.
 - **Parameters:**
 - * **pin:** Number of the IO-pin.
 - * **isInput:** True for input; false for output.
 - * **negate:** True to negate, false to affirm.
- **void pause(bool pause):**
 - **Description:** Pauses or continues the scanning process. If auto polling is deactivated, an additional sendConfig is necessary.
 - **Parameters:**
 - * **pause:** True to pause, false to continue.
- **void readBitmapData(const char* filename, PixelFormat format, int stride, bool saveMode = false):**
 - **Description:** Reads the currently stored image from the scanner and saves it at the given location.
 - **Parameters:**
 - * **filename:** Name of the file which should store the image.
 - * **format:** Pixel format.
 - * **stride:** Bytes per row.
 - * **saveMode:** Optional parameter, false by default.
- **void saveConfig(const char* fileName):**
 - **Description:** Saves the currently loaded configuration to an XML or binary file.
 - **Parameters:**
 - * **fileName:** Name of the file.
- **void saveConfig(unsigned char slot):**
 - **Description:** Saves the currently active configuration to a given slot in the internal storage.
 - **Parameters:**
 - * **slot:** Internal storage slot (0-3).
- **int sendBitmapData(const char* fileName, int timeout = 60000):**
 - **Description:** Sends a bitmap to the scanner.
 - **Parameters:**
 - * **fileName:** Filename of the bitmap.
 - * **timeout:** Timeout in milliseconds, default is 60000 ms.
 - **Returns:**

- * 0 - Success.
- * 1 - General error (e.g., wrong file format).
- * 100 - No answer from scanner.
- * 200 - Timeout (60 seconds).
- **void sendConfig():**
 - **Description:** Sends the currently loaded configuration to the scanner. This is not necessary if auto polling is activated.
- **void sendConfig(const char* fileName):**
 - **Description:** Sends the configuration from a specified file to the scanner.
 - **Parameters:**
 - * **fileName:** Path of the configuration file.
- **void setActiveConfigSlot(unsigned char slot):**
 - **Description:** Switches to one of the 4 internal configuration slots and loads the stored values.
 - **Parameters:**
 - * **slot:** Internal slot (0-3).
- **void setCopyX(int copyX):**
 - **Description:** Sets the repetitions of the data pattern in x-direction.
 - **Parameters:**
 - * **copyX:** Number of data pattern repetitions.
- **void setCopyY(int copyY):**
 - **Description:** Sets the repetitions of the data pattern in y-direction.
 - **Parameters:**
 - * **copyY:** Number of data pattern repetitions.
- **void setDatafieldDimensions(double x, double y, double width, double height):**
 - **Description:** Sets the size and position of the data field.
 - **Parameters:**
 - * **x:** Location of the data field's lower left corner on the x-axis.
 - * **y:** Location of the data field's lower left corner on the y-axis.
 - * **width:** Width of the data field.
 - * **height:** Height of the data field.
- **void setDatafieldDimensions(double x, double y, double z, double width, double height, double depth):**
 - **Description:** Sets the size and position of the data field in 3D.
 - **Parameters:**
 - * **x:** Location of the data field's lower left corner on the x-axis.
 - * **y:** Location of the data field's lower left corner on the y-axis.
 - * **z:** Start of the data field on the z-axis.
 - * **width:** Width of the data field.
 - * **height:** Height of the data field.
 - * **depth:** Depth of the data field.

- **void setEnabledFacets(unsigned char enabledFacets):**
 - **Description:** Sets the enabled state of each facet.
 - **Parameters:**
 - * **enabledFacets:** A byte where each bit represents the enabled state (e.g., 0x0F to disable facets 5-8).
- **void setFacetCorrectionPhi(int facet, int correctionValue):**
 - **Description:** Sets the facet correction value phi (d) of a given facet.
 - **Parameters:**
 - * **facet:** Facet number (1-8).
 - * **correctionValue:** Correction value range [-128 to 127].
 - **Returns:** True on success, false if parameter is out of range or connection is not established.
- **void setFacetCorrectionX(int facet, int correctionValue):**
 - **Description:** Sets the horizontal facet correction value (dX) of a given facet.
 - **Parameters:**
 - * **facet:** Facet number (1-8).
 - * **correctionValue:** Correction value range [-128 to 127].
 - **Returns:** True on success, false if parameter is out of range or connection is not established.
- **void setFacetCorrectionY(int facet, int correctionValue):**
 - **Description:** Sets the vertical facet correction value (dY) of a given facet.
 - **Parameters:**
 - * **facet:** Facet number (1-8).
 - * **correctionValue:** Correction value range [-512 to 511].
 - **Returns:** True on success, false if parameter is out of range or connection is not established.
- **void setGalvoDelay(double delay):**
 - **Description:** Sets the delay of the galvo.
 - **Parameters:**
 - * **delay:** Delay in microseconds (μs).
- **void setGalvoHysteresis(double hysteresis):**
 - **Description:** Sets the hysteresis of the galvo.
 - **Parameters:**
 - * **hysteresis:** Hysteresis in millimeters (mm).
- **void setGalvoLeadTime(double galvoLeadTime):**
 - **Description:** Sets the lead time of the galvo scanner.
 - **Parameters:**
 - * **galvoLeadTime:** Lead time in microseconds (μs).
- **void setGlobalXShift(unsigned int xShift):**
 - **Description:** Sets the offset in the x-direction.
 - **Parameters:**

- * **xShift**: Galvo index offset.
- **void setGlobalYShift(unsigned int yShift):**
 - **Description:** Sets the index offset in the y-direction.
 - **Parameters:**
 - * **yShift**: Index offset for the polygon.
- **void setHatch(double hatch):**
 - **Description:** Sets the distance between separate laser tracks in micrometers (μm).
 - **Parameters:**
 - * **hatch**: The hatch distance.
- **void setInfiniteCopyX(bool isPositive):**
 - **Description:** Sets infinite repetitions of the data pattern in the x-direction.
 - **Parameters:**
 - * **isPositive**: Sets the direction positive (true) or negative (false).
- **void setInfiniteCopyY(bool isPositive):**
 - **Description:** Sets infinite repetitions of the data pattern in the y-direction.
 - **Parameters:**
 - * **isPositive**: Sets the direction positive (true) or negative (false).
- **void setLaserOnDelay(double laserOnDelay):**
 - **Description:** Sets the laser on delay.
 - **Parameters:**
 - * **laserOnDelay**: Laser on delay in microseconds (μs).
- **void setLineCount(int lineCount):**
 - **Description:** Sets the number of line repetitions.
 - **Parameters:**
 - * **lineCount**: Number of line repetitions.
- **void setLogLevel(LogLevel logLevel):**
 - **Description:** Activates logging to the console based on the log level.
 - **Parameters:**
 - * **logLevel**: Log level used to filter information.
 - Nothing = 0
 - Error = 1
 - Warning = 2
 - Information = 3
 - Everything = 4
- **void setObjectCount(int objectCount):**
 - **Description:** Sets the number of object repetitions.
 - **Parameters:**
 - * **objectCount**: Number of object repetitions.

- **void setObjectDimensions(double x, double y, double width, double height):**
 - **Description:** Sets the size and position of the object.
 - **Parameters:**
 - * **x:** Location of the object's lower left corner on the x-axis.
 - * **y:** Location of the object's lower left corner on the y-axis.
 - * **width:** Width of the object.
 - * **height:** Height of the object.
- **void setObjectDimensions(double x, double y, double z, double width, double height, double depth):**
 - **Description:** Sets the size and position of the object in 3D.
 - **Parameters:**
 - * **x:** Location of the object's lower left corner on the x-axis.
 - * **y:** Location of the object's lower left corner on the y-axis.
 - * **z:** Start of the object on the z-axis.
 - * **width:** Width of the object.
 - * **height:** Height of the object.
 - * **depth:** Depth of the object.
- **void setOffsetX(unsigned int offsetX):**
 - **Description:** Sets the offset in the x-direction.
 - **Parameters:**
 - * **offsetX:** Galvo index offset.
- **void setOffsetY(unsigned int offsetY):**
 - **Description:** Sets the index offset in the y-direction.
 - **Parameters:**
 - * **offsetY:** Index offset for the polygon.
- **void setOutputMode(MoewePS_U::OutputMode outputMode):**
 - **Description:** Sets the output mode to one of the predefined modes.
 - **Parameters:**
 - * **outputMode:** Mode to set (e.g., greater than threshold, pulse width modulation).
- **void setMicronsPerBitX(double micronsPerBit):**
 - **Description:** Sets the optical resolution in the x-direction.
 - **Parameters:**
 - * **micronsPerBit:** Micrometers per bit in the x-direction.
- **void setMicronsPerBitY(double micronsPerBit):**
 - **Description:** Sets the optical resolution in the y-direction.
 - **Parameters:**
 - * **micronsPerBit:** Micrometers per bit in the y-direction.
- **void setMicronsPerBitZ(double micronsPerBit):**
 - **Description:** Sets the optical resolution in the z-direction.

- **Parameters:**
 - * **micronsPerBit:** Micrometers per bit in the z-direction.
- **void setScanfieldDimensions(double x, double y, double width, double height):**
 - **Description:** Sets the size and position of the scan field.
 - **Parameters:**
 - * **x:** Location of the scan field's lower left corner on the x-axis.
 - * **y:** Location of the scan field's lower left corner on the y-axis.
 - * **width:** Width of the scan field.
 - * **height:** Height of the scan field.
- **void setScanfieldDimensions(double x, double y, double z, double width, double height, double depth):**
 - **Description:** Sets the size and position of the scan field in 3D.
 - **Parameters:**
 - * **x:** Location of the scan field's lower left corner on the x-axis.
 - * **y:** Location of the scan field's lower left corner on the y-axis.
 - * **z:** Start of the scan field on the z-axis.
 - * **width:** Width of the scan field.
 - * **height:** Height of the scan field.
 - * **depth:** Depth of the scan field.
- **void setShifterOptions(unsigned char options, MoewePS_U::Axis axis, MoewePS_U::ShiftType shiftType):**
 - **Description:** Sets the shifter options for a certain axis and shift type.
 - **Parameters:**
 - * **options:** Shift options (refer to scanner manual).
 - * **axis:** Axis to set options for.
 - * **shiftType:** Type of shift (Set, Reset, Direction, Out).
- **void setShiftValue(double value, MoewePS_U::Axis axis, MoewePS_U::ShiftValue shiftValue):**
 - **Description:** Sets one of the two shift values or the reset value for a given axis.
 - **Parameters:**
 - * **value:** Value to set (in micrometers for Value1 and Value2, millimeters for ResetValue, steps for SetEach and ResetEach, microseconds for SetTimer and ResetTimer).
 - * **axis:** Axis to set the value for.
 - * **shiftValue:** Shift value type.
- **void setSimulationMode(bool isSimulationModeOn):**
 - **Description:** Commands the scanner to simulate motor movement.
 - **Parameters:**
 - * **isSimulationModeOn:** Determines whether the simulation is on (true) or off (false).
- **void setSpaceX(double spaceX):**
 - **Description:** Sets the space between the data patterns in the x-direction.
 - **Parameters:**

- * spaceX: Space between the data patterns in millimeters (mm).
- **void setSpaceY(double spaceY):**
 - **Description:** Sets the space between the data patterns in the y-direction.
 - **Parameters:**
 - * spaceY: Space between the data patterns in millimeters (mm).
- **void setSubhatch(double subhatch):**
 - **Description:** Sets the offset by which the lines are shifted after each object repetition.
 - **Parameters:**
 - * subhatch: Subhatch in micrometers (μm).
- **void setThreshold(int threshold):**
 - **Description:** Sets the threshold for various modifiers (e.g., pulse width modulation).
 - **Parameters:**
 - * threshold: Threshold value.
- **void setVelocity(double velocity):**
 - **Description:** Sets the velocity of the fast axis in meters per second (m/s).
 - **Parameters:**
 - * velocity: Velocity of the laser track. Set to zero to disable the motor.
- **void setVelocityRPM(double velocity):**
 - **Description:** Sets the velocity of the fast axis in rounds per minute (rpm).
 - **Parameters:**
 - * velocity: Velocity of the laser track in rpm. Set to zero to disable the motor.
- **void setXCorrectionScaling(int scaling):**
 - **Description:** Sets the scaling used for optical x-correction.
 - **Parameters:**
 - * scaling: Scaling factor. Allowed values are 1 (1:1), 2 (1:2), 4 (1:4), or 8 (1:8).
- **void start():**
 - **Description:** Starts the scan process.
- **void stop():**
 - **Description:** Stops the scan process.
- **void stopBmpDataStream():**
 - **Description:** Stops bitmap streaming.
- **bool uploadCorrectionFile(const char* fileName):**
 - **Description:** Sends a correction file for optical adjustment to the scanner.
 - **Parameters:**
 - * fileName: File name with the extension .cnf, .txt, or .csv.
 - **Returns:** true on success, false otherwise.

4 CODE EXAMPLES

4.1 C++ CODE EXAMPLE

Below you find a short code example for a C++ console project.

```
// !!! EDIT REQUIRED !!!
#include "C:\PATH_TO_HEADER\MoewePS_U.h"
#include <iostream>
#include <chrono>
#include <thread>
#include <windows.h>

// !!! EDIT REQUIRED !!!
// ip of the scanner
const char* ip = "192.168.222.10";

int main()
{
    try
    {
        // Initialization
        SetConsoleOutputCP(CP_UTF8); // to display infinity symbol in console
        MoewePS_U* scanner = new MoewePS_U();
        scanner->setLogLevel(MoewePS_U::LogLevel::Information);

        // Connect
        scanner->connect(ip, true, 3000);
        while (!scanner->isConnected())
        {
            std::this_thread::sleep_for(std::chrono::milliseconds(10));
            printf("*");
        }
        printf("\n");

        // Stop motor and Process
        scanner->enableMotor(false);
        scanner->stop();

        // Use internal config slot 4 and
        // save current data to filesystem
        // for later restore
        int initialConfigSlot = scanner->getCurrentConfigSlot();
        scanner->setActiveConfigSlot(3); // (sic!) zerobased
        const char* configFileName = "MoewePS_Config_Slot4.xml";
        scanner->saveConfig(configFileName);

        // Set some values
        printf("\n");
        printf("Configuring scanner\n");
        double velocity = 800; // rounds per minute
        scanner->setVelocityRPM(velocity);

        MoewePS_U::OutputMode outputMode = MoewePS_U::OutputMode::PulseWidthModulation;
```

```

scanner->setOutputMode(outputMode);

double objectX = -70; // mm
double objectY = -100; // mm
double objectWidth = 140; // mm
double objectHeight = 200; // mm
scanner->setObjectDimensions(
    objectX,
    objectY,
    objectWidth,
    objectHeight);

double dataX = 0; // mm
double dataY = 0; // mm
double dataWidth = 60; // mm
double dataHeight = 60; // mm
scanner->setDatafieldDimensions(
    dataX,
    dataY,
    dataWidth,
    dataHeight);

int objectCount = 2;
int lineCount = 8;
scanner->setObjectCount(objectCount);
scanner->setLineCount(lineCount);

double hatch = 100; //μm
double subHatch = 0; //μm
scanner->setHatch(hatch);
scanner->setSubhatch(subHatch);

int copyX = 1;
int copyY = 2;
double spaceX = 20; //mm
double spaceY = 10; //mm
scanner->setCopyX(copyX);
scanner->setCopyY(copyY);
scanner->setSpaceX(spaceX);
scanner->setSpaceY(spaceY);

// Read some values
printf("\n");
printf("Reading scanner properties\n");
velocity = scanner->getVelocityRPM();
printf("current velocity: %f rpm\n", velocity);
velocity = scanner->getVelocity();
printf("current velocity: %f m/s\n", velocity);

outputMode = scanner->getOutputMode();
std::string outputModeString = "Output mode was set to: ";
switch (outputMode)
{

```

```

case MoewePS_U::OutputMode::GreaterThanThreshold:
    outputModeString += "Greater than threshold";
    break;
case MoewePS_U::OutputMode::EqualToThreshold:
    outputModeString += "Equal to threshold";
    break;
case MoewePS_U::OutputMode::GreaterThanThresholdAndIncrement:
    outputModeString += "Greater than threshold and increment";
    break;
case MoewePS_U::OutputMode::EqualToThresholdAndIncrement:
    outputModeString += "Equal to threshold and increment";
    break;
case MoewePS_U::OutputMode::PulseWidthModulation:
    outputModeString += "Pulse width modulation";
    break;
case MoewePS_U::OutputMode::MultibitImage:
    outputModeString += "Multibit image";
    break;
case MoewePS_U::OutputMode::LessThanThreshold:
    outputModeString += "Less than threshold";
    break;
case MoewePS_U::OutputMode::LessThanThresholdAndIncrement:
    outputModeString += "Less than threshold and increment";
    break;
case MoewePS_U::OutputMode::Real:
    outputModeString += "Real";
    break;
default:
    outputModeString += "How did we even got here";
    break;
}
outputModeString += "\n";
printf(outputModeString.c_str());

objectX = scanner->getObjectX();
objectY = scanner->getObjectY();
objectWidth = scanner->getObjectWidth();
objectHeight = scanner->getObjectHeight();
printf("\n");
printf("current object dimensions: \n");
printf("      x = %.2f mm\n", objectX);
printf("      y = %.2f mm\n", objectY);
printf("    width = %.2f mm\n", objectWidth);
printf("    height = %.2f mm\n", objectHeight);

dataX = scanner->getDatafieldX();
dataY = scanner->getDatafieldY();
dataWidth = scanner->getDatafieldWidth();
dataHeight = scanner->getDatafieldHeight();
printf("\n");
printf("current datafield dimensions: \n");
printf("      y = %.2f mm\n", dataY);
printf("    width = %.2f mm\n", dataWidth);
printf("    height = %.2f mm\n", dataHeight);

```

```

printf("          x = %.2f mm\n", dataX);

objectCount = scanner->getObjectCount();
lineCount = scanner->getLineCount();
printf("\n");
printf("objects are processed %d times \n", objectCount);
printf("each line is processed %d times \n", lineCount);

hatch = scanner->getHatch();
subHatch = scanner->getSubhatch();
printf("\n");
printf("each line will be shifted %.2f micrometers \n", hatch);
printf("each line will be shifted additional %.2f micrometers afer each object repetiti

copyX = scanner->getCopyX();
copyY = scanner->getCopyY();
spaceX = scanner->getSpaceX();
spaceY = scanner->getSpaceY();

printf("\n");
printf("In x-direction the datafield will be copied %d times (processed %d times in tota
printf("In y-direction the datafield will be copied %d times (processed %d times in tota
printf("The distance between the copies is set to %.2f mm in x-direction \n", spaceX);
printf("and %.2f mm in y-direction \n", spaceY);

// optional run the process
printf("Start process? (y/n)\n");
char c = getchar();
while (getchar() != '\n');
if(c == 'y' || c == 'Y')
{
    scanner->enableMotor(true);
    scanner->start();
    printf("Process started. Press ENTER to stop\n");
    c = getchar();
    scanner->enableMotor(false);
    scanner->stop();
}

//restore the saved config from filesystem
scanner->loadConfig(configFileName);
scanner->saveConfig(3);
scanner->setActiveConfigSlot(initialConfigSlot);

// disconnect
scanner->disconnect();
}
catch (const char* msg)
{
    printf("An error has occured: %s", msg);
}
}

```