pyTempico

Release 2.0.0

Tausand Electronics

Oct 18, 2025

CONTENTS:

PyTempico is a Python library developed by Tausand Electronics, with Joan Amaya and David Guzman as the lead developers. It enables seamless interaction with Tempico TP1004 devices, facilitating data communication and control. The library relies on key modules such as hid and pyserial for efficient device connectivity. For more information, visit tausand.com

Release 2.0.0

CHAPTER

ONE

PYTEMPICO PACKAGE

Created on Jan 30 2024

@author: David Guzman at Tausand Electronics
dguzman@tausand.com
https://www.tausand.com

Core class and methods for PyTempico library.

To use with Tausand Electronics' Time-to-Digital Converters (TDCs) of the family *Tausand Tempico*.

1.1 TempicoDevicesSearch class

class pyTempico.core.**TempicoDevicesSearch**

A class for discovering Tempico devices.

This class provides methods to search for Tempico devices in a network or connected system.

findDevices()

Finds and verifies whether a device with the given VID and PID is a Tempico device.

This function takes the Vendor ID (VID) and Product ID (PID) as inputs, converts them to integers, and attempts to open the device using these values. It then checks if the manufacturer and product strings match the expected values for a Tempico device.

Parameters

- **vid_s** (*str*) – The Vendor ID (VID) of the device in string format.
- **pid_s** (*str*) – The Product ID (PID) of the device in string format.

Returns *True* if the device is a Tempico, *False* otherwise.

Return type bool

getVidPid(*vid_pid_information*)

Extracts the Vendor ID (VID) and Product ID (PID) from a string and returns them as a tuple.

This function processes a string that contains the VID and PID information in the format 'VID:PID=xxxx:yyyy'. It splits the string and retrieves the VID and PID values, returning them as a tuple of strings.

Parameters **vid_pid_information** (*str*) – A string containing the VID and PID information.

Returns A tuple containing the VID and PID as strings (vid, pid).

Return type tuple

tryOpenDevices(*vid_s*, *pid_s*)

Finds and verifies whether a device with the given VID and PID is a Tempico device.

This function takes the Vendor ID (VID) and Product ID (PID) as inputs, converts them to integers, and attempts to open the device using these values. It then checks if the manufacturer and product strings match the expected values for a Tempico device.

Parameters

- **vid_s** (*str*) – The Vendor ID (VID) of the device in string format.
- **pid_s** (*str*) – The Product ID (PID) of the device in string format.

Returns *True* if the device is a Tempico, *False* otherwise.

Return type bool

verifyPyTempico(*tuple_vid_pid*)

Verifies whether the connected device is a Tempico device.

This function checks if the device's Vendor ID (VID) and Product ID (PID) match the values corresponding to a Tempico device. It returns *True* if the device is identified as a Tempico, and *False* otherwise.

Parameters **tuple_vid_pid** (*tuple*) – A tuple containing the VID and PID of the device.

Returns *True* if the device is a Tempico, *False* otherwise.

Return type bool

1.2 TempicoDevice class

class pyTempico.core.**TempicoDevice**(*com_port*)

Tausand Tempico TDC device object.

To create an object of the *TempicoDevice()* class, it is required to send as parameter the desired com_port. For example,

```
>>> my_tempico_device_object = pyTempico.TempicoDevice('COM5')
```

To modify or access attributes, **please use methods**. For example,

```
>>> my_tempico_device_object.getIdn()
```

To access attributes of a particular channel, use methods of the *TempicoChannel()* class through attributes ch1, ch2, ch3, ch4 of this class. For example, to get average cycles on channel 2,

```
>>> my_tempico_device_object.ch2.getAverageCycles()
```

or, as an alternative, send the channel number as a parameter

```
>>> my_tempico_device_object.getAverageCycles(2)
```

Changing attributes without using methods, do not change the actual parameters in the device.

Accesing attributes without using methods, returns values registered in local memory, that may not be updated.

To begin a measurement and read its results, use methods *measure()* and *fetch()*.

id_tempico_device
Unique identifier of the *TempicoDevice()* object.

Type int

ch1
Object of the *TempicoChannel()* class linked to TDC in channel 1 (input A).

Type *TempicoChannel*

ch2
Object of the *TempicoChannel()* class linked to TDC in channel 2 (input B).

Type *TempicoChannel*

ch3
Object of the *TempicoChannel()* class linked to TDC in channel 3 (input C).

Type *TempicoChannel*

ch4
Object of the *TempicoChannel()* class linked to TDC in channel 4 (input D).

Type *TempicoChannel*

device
Serial port object.

Type Serial

idn
Identification string.

Type str

number_of_channels
number of stop inputs of the device.

Type int

number_of_runs
Number of measurement runs of the TDCs in *TempicoDevice()*.

Type int

port
Serial port string.

Type str

threshold
Threshold voltage on the rising edge of start and stops inputs of TDCs in the *TempicoDevice()*.

Type float

abort(*validate=True*)
Cancels an ongoing measurement on the *TempicoDevice()*.

This function sends a cancel command to the *TempicoDevice()* to stop any measurement currently in progress. It ensures that all measurement processes are halted and the device is ready for a new operation or safely turned off.

This function requires that a connection is established with the *TempicoDevice()*.

Parameters **validate** (*bool, optional*) – If True, waits and validates if every channel applies the abort sucessfully. Default is True.

`calibrateDelay()`
Calibrates the internal delay`.

This command adjusts the hardware's internal timing to ensure accurate delay measurements. Only supported on devices with hardware version "TP12".

Parameters `None` –

Returns None

`cleanNumberList(number_list)`
Removes non-valid elements of a Tempico dataset, given in number_list, a number 2D-list (read data). Returns a clean number 2D-list.

Validations applied:

- length of row
- type of data (integers, except for start_s register)
- ch in range
- run in range
- sorted stops
- stops in range
- sequential run values
- increasing start

The dataset of a *TempicoDevice()* is in the following format:

[[ch,run,start_s,stop_ps1,...,stop_psN],...,[ch,run,start_time_us,stop_ps1,...,stop_psN]]

where

- 'ch' indicates the TDC channel,
- 'run' goes from 1 to NumberOfRuns,
- 'start_s' is the epoch timestamp of start pulse, in seconds, with microseconds precision; this value overflows (go back to zero) after 2^32 seconds,
- 'stop_ps1' is the measured precision timelapse between start and the 1st stop pulse, in picoseconds,
- 'N' is the NumberOfStops.

Parameters `number_list` (*list(number)*) – a complete dataset message converted.

Returns a cleaned complete dataset message converted.

Return type list(number)

`close()`
Ends (closes) a connection with a *TempicoDevice()*.

It is recommended to close connection at the end of a routine, to free the device's port for future use.

Parameters `(none)` –

`closeTempico()`
Ends (closes) a connection with a *TempicoDevice()*.

Same as method *close()*.

Parameters `(none)` –

convertReadDataToFloatList(*data_string*)

Converts a string with a read dataset message issued by a *TempicoDevice()*, into an float 2D-list.

The dataset of a *TempicoDevice()* is in the following format:

[[ch,run,start_s,stop_ps1,...,stop_psN],...,[ch,run,start_time_us,stop_ps1,...,stop_psN]]

where

- 'ch' indicates the TDC channel,
- 'run' goes from 1 to NumberOfRuns,
- 'start_s' is the epoch timestamp of start pulse, in seconds, with microseconds precision; this value overflows (go back to zero) after 2^32 seconds,
- 'stop_ps1' is the measured precision timelapse between start and the 1st stop pulse, in picoseconds,
- 'N' is the NumberOfStops.

Every value in the dataset is converted to a float.

Parameters **data_string** (*str*) – dataset message to convert.

Returns dataset message converted.

Return type list(float)

convertReadDataToNumberList(*data_string*)

Converts a string with a read dataset message issued by a *TempicoDevice()*, into a number 2D-list (integer or float).

The dataset of a *TempicoDevice()* is in the following format:

```
[[ch,run,start_s,stop_ps1,...,stop_psN],...,[ch,run,start_time_us,stop_ps1,...,
↪stop_psN]]
```

where

- 'ch' (int) indicates the TDC channel,
- 'run' (int) goes from 1 to NumberOfRuns,
- 'start_s' (float) is the epoch timestamp of start pulse, in seconds, with microseconds precision; this value overflows (go back to zero) after 2^32 seconds,
- 'stop_ps1' (int) is the measured precision timelapse between start and the 1st stop pulse, in picoseconds,
- 'N' (int) is the NumberOfStops.

Every value in the dataset is either an integer or a float.

Parameters **data_string** (*str*) – dataset message to convert.

Returns dataset message converted.

Return type list(number)

decreaseGeneratorFrequency()

Decreases the internal generator frequency by one step (only for TP12 devices).

This function lowers the generator frequency by a predefined decrement factor (1, 2, 5, or 10 × 10) depending on the current frequency range.

Parameters **None** –

Returns None

decrementThresholdVoltage()
Reduces the threshold voltage on the rising edge of start and stops inputs of TDCs in the *TempicoDevice()*.

To validate the actual threshold voltage applied, method *getThresholdVoltage()* should be called.

This function requires that a connection is established with the *TempicoDevice()*.

Parameters **(none)** –

disableChannel(*channel*)
Disables the specified *TempicoChannel()*.

By default, channels are enabled.

This function requires that a connection is established with the *TempicoDevice()*, and delegates the disabling operation to the selected *TempicoChannel()*. If the channel is invalid, the function does nothing.

To validate the status of the channel, method *isEnabled()* may be used.

This version belongs to the *TempicoDevice()* class and requires specifying the channel number or label ('A'–'D') to access and configure the corresponding *TempicoChannel()*.

Parameters **channel** (*int or str*) – Channel identifier. Accepted values are:

- 1 or 'A' or 'a'
- 2 or 'B' or 'b'
- 3 or 'C' or 'c'
- 4 or 'D' or 'd'

enableChannel(*channel*)
Enables the specified *TempicoChannel()*.

By default, channels are enabled.

This function requires that a connection is established with the *TempicoDevice()*, and delegates the enabling operation to the selected *TempicoChannel()*. If the channel is invalid, the function does nothing.

To validate the status of the channel, method *isEnabled()* may be used.

This version belongs to the *TempicoDevice()* class and requires specifying the channel number or label ('A'–'D') to access and configure the corresponding *TempicoChannel()*.

Parameters **channel** (*int or str*) – Channel identifier. Accepted values are:

- 1 or 'A' or 'a'
- 2 or 'B' or 'b'
- 3 or 'C' or 'c'
- 4 or 'D' or 'd'

fetch(*validate=True*)
Reads the most recent measurement data set form a *TempicoDevice()*.

The dataset of a *TempicoDevice()* is in the following format:

```
[[ch,run,start_s,stop_ps1,...,stop_psN],...,[ch,run,start_time_us,stop_ps1,...,
↪stop_psN]]
```

where

- 'ch' (int) indicates the TDC channel,
- 'run' (int) goes from 1 to NumberOfRuns,

- 'start_s' (float) is the epoch timestamp of start pulse, in seconds with microsecond resolution. This value overflows (go back to zero) after 2^32 seconds,
- 'stop_ps1' (int) is the measured precision timelapse between start and the 1st stop pulse, in picoseconds,
- 'N' (int) is the NumberOfStops.

Every value in the dataset is either an integer or a float number.

If no measurement has been done, the device may respond with an empty dataset. To make a new measurement, method *measure()* must be used.

This function requires that a connection is established with the *TempicoDevice()*.

Parameters **validate** (*bool, optional*) – If True, the measured dataset is validated, and invalid registers get cleaned. Default is True.

Returns measured dataset.

Return type list(number)

getAverageCycles(*channel*)

Returns the average cycles of the specified *TempicoChannel()*.

By default, average cycles = 1 (no multi-cycle averaging).

If the connection is established with the *TempicoDevice()*, this function delegates the query to the selected channel object and retrieves its average cycles value. If the connection is not open, or the channel is invalid, the function returns -1.

This version belongs to the *TempicoDevice()* class and requires specifying the channel number or label ('A'–'D') to access the corresponding *TempicoChannel()*.

Parameters **channel** (*int or str*) – Channel identifier. Accepted values are:

- 1 or 'A' or 'a'
- 2 or 'B' or 'b'
- 3 or 'C' or 'c'
- 4 or 'D' or 'd'

Returns Number of average cycles, or -1 if the channel is invalid.

Return type integer

getBaudRate()

Returns the *TempicoDevice()* baud rate.

Parameters **(none)** –

Returns baud rate.

Return type int

getDateTime(*convert_to_datetime=False*)

Same as getDatetime

getDatetime(*convert_to_datetime=False*)

Returns the number of seconds since the Tempico device was powered on, based on its internal clock. If the device has been synchronized, the seconds count corresponds to the system time of the host PC.

This function sends the *DTIMe?* command to the device, reads the response, and parses it into a float representing the elapsed time in seconds. If the device does not respond correctly, the function returns -1.

The timestamp resolution is in microseconds (µs), and the count starts from power-up by default. Synchronization with the PC time must be previously configured on the device, otherwise the value is relative to the device's uptime.

Parameters **convert_to_datetime** (*bool, optional*) – If True, the value is returned as a datetime object. Default is False.

Returns Elapsed time in seconds since the device was powered on or synchronized with the PC clock. Returns -1 if no valid response is received.

Return type float

getDelay(*channel*)

Retrieves the delay value for the specified channel`.

The function selects the given channel and queries its current delay setting. If the channel is invalid, it returns -1000000.

Parameters **channel** (*int*) – Channel number to read the delay from.

Returns Delay value of the selected channel, or -1000000 if the channel is invalid.

Return type float

getFirmware()

Returns the *TempicoDevice()* firmware version.

Parameters **(none)** –

Returns firmware version.

Return type str

getGeneratorFrequency()

Returns the frequency of the internal pulse generator (only for TP12 devices).

This function queries the device with the 'CONF:GEN:FREQ?' command to obtain the current frequency of the internal pulse generator. If the connection is not open, an informational message is printed. This feature is not available on TP1004 devices.

Parameters **None** –

Returns Current frequency of the internal pulse generator. Returns an empty string if it cannot be retrieved.

Return type float

getIdn()

Returns the *TempicoDevice()* identification string.

If the connection is established with the *TempicoDevice()*, this function request the device for the string. If not, the most recent read string is returned.

Parameters **(none)** –

Returns identification string.

Return type str

getLastDelaySync(*convert_to_datetime=False*)

Returns the timestamp of the last delay calibration performed on the *TempicoDevice()*.

If the device is connected and the hardware version is "TP12", the function requests the last delay calibration time using the 'DTIMe:LDELay?' command. If *convert_to_datetime* is True, the value is returned as a datetime object.

Parameters **convert_to_datetime** (*bool, optional*) – If True, returns the timestamp as a datetime object. Default is False.

Returns Timestamp of the last delay calibration, as a Unix timestamp or datetime object. Returns -1 if it cannot be retrieved.

Return type float or datetime

getLastStart(*convert_to_datetime=False*)

Returns the datetime of the last start event registered by the *TempicoDevice()*.

If the connection is established with the *TempicoDevice()*, this function requests the device for the last start timestamp using the 'DTIMe:LSTart?' command. If not, the value -1 is returned.

The returned value corresponds to the timestamp of the most recent start event. If no start has occurred yet, the device returns 0. If *convert_to_datetime* is set to True, the value is returned as a datetime object instead of a float.

Parameters **convert_to_datetime** (*bool, optional*) – If True, the value is returned as a datetime object. Default is False.

Returns Timestamp of the last start event, either as a float (Unix timestamp) or as a datetime object if convert_to_datetime is True. Returns -1 if the value could not be retrieved.

Return type float or datetime

getLastSync(*convert_to_datetime=False*)

Returns the datetime of the last synchronization performed on the *TempicoDevice()*.

If the connection is established with the *TempicoDevice()*, this function requests the device for the last synchronization timestamp using the 'DTIMe:LSYNc?' command. If not, the value -1 is returned.

The returned value corresponds to the timestamp of the most recent synchronization with the host system. If no synchronization has occurred yet, the device returns 0. If *convert_to_datetime* is set to True, the value is returned as a datetime object instead of a float.

Parameters **convert_to_datetime** (*bool, optional*) – If True, the value is returned as a datetime object. Default is False.

Returns Timestamp of the last synchronization event, either as a float (Unix timestamp) or as a datetime object if convert_to_datetime is True. Returns -1 if the value could not be retrieved.

Return type float or datetime

getMaximumDateTime(*convert_to_datetime=False*)

Same as getMaximumDatetime

getMaximumDatetime(*convert_to_datetime=False*)

Returns the maximum datetime value allowed by the *TempicoDevice()*.

If the connection is established with the *TempicoDevice()*, this function requests the device for the maximum allowed date using the 'DTIMe:MAXimum?' command. If not, the value -1 is returned.

The returned value corresponds to the latest timestamp that can be set on the device without causing an error. If *convert_to_datetime* is set to True, the value is returned as a datetime object instead of a float.

Parameters **convert_to_datetime** (*bool, optional*) – If True, the value is returned as a datetime object. Default is False.

Returns Maximum allowed datetime, either as a float (Unix timestamp) or as a datetime object if convert_to_datetime is True.

Return type float or datetime

getMinimumDateTime(*convert_to_datetime=False*)
Same as getMinimumDatetime

getMinimumDatetime(*convert_to_datetime=False*)
Returns the minimum datetime value allowed by the *TempicoDevice()*.

If the connection is established with the *TempicoDevice()*, this function requests the device for the minimum allowed date using the 'DTIMe:MINimum?' command. If not, the value -1 is returned.

The returned value corresponds to the earliest timestamp that can be set on the device without causing an error. If *convert_to_datetime* is set to True, the value is returned as a datetime object instead of a float.

Parameters **convert_to_datetime** (*bool, optional*) – If True, the value is returned as a datetime object. Default is False.

Returns Minimum allowed datetime, either as a float (Unix timestamp) or as a datetime object if convert_to_datetime is True.

Return type float or datetime

getMode(*channel*)
Returns the measurement mode of the specified *TempicoChannel()*.

By default, mode = 1.

If the connection is established with the *TempicoDevice()*, this function delegates the query to the selected channel object and retrieves its current measurement mode. If the connection is not open, or the channel is invalid, the function returns -1.

This version belongs to the *TempicoDevice()* class and requires specifying the channel number or label ('A'–'D') to access the corresponding *TempicoChannel()*.

Parameters **channel** (*int or str*) – Channel identifier. Accepted values are:

- 1 or 'A' or 'a'
- 2 or 'B' or 'b'
- 3 or 'C' or 'c'
- 4 or 'D' or 'd'

Returns

Mode. Possible values are:

- 1: Short measurement range. Start-stop times from 12ns to 500ns.
- 2: Large measurement range. Start-stop times from 125ns to 4ms.

Returns -1 if the channel is invalid.

Return type integer

getModelIdn()
Returns the identifier of the model associated with the device.

Returns The internal model identifier (*model_idn*).

Return type str

getNumberOfRuns()
Returns the number of measurement runs of the TDCs in *TempicoDevice()*.

By default, number of runs = 1 (single measurement).

If the connection is established with the *TempicoDevice()*, this function request the device for the value. If not, the most recent value is returned.

Parameters **(none)** –

Returns Number of number of runs.

Return type integer

getNumberOfStops(*channel*)

Returns the expected number of stop pulses for the specified *TempicoChannel()*.

By default, number of stops = 1 (single start → single stop).

The TDC must receive all the expected number of stops to register them as a valid measurement; otherwise, the measurements are discarded.

If the connection is established with the *TempicoDevice()*, this function delegates the query to the selected channel object and retrieves its number of stops. If the connection is not open, or the channel is invalid, the function returns -1.

This version belongs to the *TempicoDevice()* class and requires specifying the channel number or label ('A'–'D') to access the corresponding *TempicoChannel()*.

Parameters **channel** (*int or str*) – Channel identifier. Accepted values are:

- 1 or 'A' or 'a'
- 2 or 'B' or 'b'
- 3 or 'C' or 'c'
- 4 or 'D' or 'd'

Returns Number of stops, or -1 if the channel is invalid.

Return type integer

getOverflowParameter()

Retrieves the overflow parameter.

The returned value depends on the device hardware version. For TP12 devices, the function returns -1000000; otherwise, it returns -1.

Parameters **None** –

Returns Overflow parameter value based on the hardware version.

Return type int

getSerialNumber()

Returns the serial number of the connected *TempicoDevice()*.

This function searches for the serial number associated with the current COM port used by the device. If the connection is established, it queries the list of available serial ports to match the active one and extract its serial number.

If the connection is not open, a message is printed and an empty string is returned.

Parameters **(none)** –

Returns Serial number of the device, or empty string if not found or if the connection is not open.
e.g: "TP1004-220500"

Return type string

getSettings()

Reads the current settings form a *TempicoDevice()*.

The response for settings query on a *TempicoDevice()* is in the following format:

```
CH1:ACYC 1;CH1:ENAB 1;CH1:NST 1;...;CH4:STOP:MASK 0;NRUN 1;THR 1.00
```

This function requires that a connection is established with the *TempicoDevice()*.

Parameters **(none)** –

Returns device settings.

Return type str

getStartEdge(*channel*)

Returns the edge type used on start pulses of the specified *TempicoChannel()*.

By default, start edge = 'RISE'.

If the connection is established with the *TempicoDevice()*, this function delegates the query to the selected channel object and retrieves its configured start edge type. If the connection is not open, or the channel is invalid, the function returns -1.

This version belongs to the *TempicoDevice()* class and requires specifying the channel number or label ('A'–'D') to access the corresponding *TempicoChannel()*.

Parameters **channel** (*int or str*) – Channel identifier. Accepted values are:

- 1 or 'A' or 'a'
- 2 or 'B' or 'b'
- 3 or 'C' or 'c'
- 4 or 'D' or 'd'

Returns

Start edge type. Possible values are:

- 'RISE': TDC timing starts on a rising edge of the start pulse.
- 'FALL': TDC timing starts on a falling edge of the start pulse.

Returns -1 if the channel is invalid.

Return type string or int

getStartSource(*channel*)

Returns the start signal source for the specified channel.

This function checks whether the start signal of the selected channel comes from the internal generator or from an external input connected to the device.

Parameters **channel** (*int*) – Channel number to read the start source from.

Returns Start source identifier. Returns -1 if the channel is invalid.

Return type int

getState(*channel*)

Returns the state of the specified *TempicoChannel()*.

This function is used to validate if a reset or an abort command has been successfully applied.

Some possible states are:

- 0: disabled.
- 1: idle, enabled.
- 10: processing a reset.
- 11: processing an abort.

other states are related with the measurement process.

This version belongs to the *TempicoDevice()* class and requires specifying the channel number or label ('A'–'D') to access the corresponding *TempicoChannel()*.

Parameters **channel** (*int or str*) – Channel identifier. Accepted values are:

- 1 or 'A' or 'a'
- 2 or 'B' or 'b'
- 3 or 'C' or 'c'
- 4 or 'D' or 'd'

Returns

state.

Returns -1 if the channel is invalid.

Return type int

getStates()

Returns a list with the state of each *TempicoChannel()* within *TempicoDevice()*.

This function is used to validate if a reset or an abort command has been successfully applied.

Some possible states are:

- 0: disabled.
- 1: idle, enabled.
- 10: processing a reset.
- 11: processing an abort.

other states are related with the measurement process.

Parameters **(none)** –

Returns states per channel.

Return type list(integer)

getStatus(*channel*)

Returns the internal status of the specified *TempicoChannel()*.

This function is used to obtain the state of a channel, used to validate if a reset or an abort command has been successfully applied.

This version belongs to the *TempicoDevice()* class and requires specifying the channel number or label ('A'–'D') to access the corresponding *TempicoChannel()*.

Parameters **channel** (*int or str*) – Channel identifier. Accepted values are:

- 1 or 'A' or 'a'
- 2 or 'B' or 'b'
- 3 or 'C' or 'c'

- 4 or ‘D’ or ‘d’

Returns

status fields and values.

Returns -1 if the channel is invalid.

Return type dict

getStopEdge(*channel*)

Returns the edge type used on stop pulses of the specified *TempicoChannel()*.

By default, stop edge = ‘RISE’.

If the connection is established with the *TempicoDevice()*, this function delegates the query to the selected channel object and retrieves its configured stop edge type. If the connection is not open, or the channel is invalid, the function returns -1.

This version belongs to the *TempicoDevice()* class and requires specifying the channel number or label (‘A’–‘D’) to access the corresponding *TempicoChannel()*.

Parameters **channel** (*int or str*) – Channel identifier. Accepted values are:

- 1 or ‘A’ or ‘a’
- 2 or ‘B’ or ‘b’
- 3 or ‘C’ or ‘c’
- 4 or ‘D’ or ‘d’

Returns

Stop edge type. Possible values are:

- ’RISE’: TDC timing ends on a rising edge of the stop pulse.
- ’FALL’: TDC timing ends on a falling edge of the stop pulse.

Returns -1 if the channel is invalid.

Return type string or int

getStopMask(*channel*)

Returns the stop mask time (in microseconds) of the specified *TempicoChannel()*.

By default, stop mask = 0 (no masking).

The stop mask defines the period after receiving a start pulse during which stop pulses are ignored. This helps eliminate unwanted noise or early pulses.

If the connection is established with the *TempicoDevice()*, this function delegates the query to the selected channel object and retrieves its current stop mask time. If the connection is not open, or the channel is invalid, the function returns -1.

This version belongs to the *TempicoDevice()* class and requires specifying the channel number or label (‘A’–‘D’) to access the corresponding *TempicoChannel()*.

Parameters **channel** (*int or str*) – Channel identifier. Accepted values are:

- 1 or ‘A’ or ‘a’
- 2 or ‘B’ or ‘b’
- 3 or ‘C’ or ‘c’
- 4 or ‘D’ or ‘d’

Returns Stop mask time in microseconds, or -1 if the channel is invalid.

Return type integer

`getStopSource`(*channel*)
Returns the stop signal source for the specified channel.

This function checks whether the stop signal of the selected channel comes from the internal generator or from an external input connected to the device.

Parameters `channel` (*int*) – Channel number to read the stop source from.

Returns Stop source identifier. Returns -1 if the channel is invalid.

Return type int

`getTempicoChannel`(*channel*)
Returns the *TempicoChannel()* object corresponding to the specified channel.

This function allows selecting a channel by its number (1–4) or label ('A'–'D'). If the input is valid, it returns the corresponding *TempicoChannel()* instance. If the input is invalid, an error message is printed and -1 is returned.

Parameters `channel` (*int or str*) – Channel identifier. Accepted values are:

- 1 or 'A' or 'a'
- 2 or 'B' or 'b'
- 3 or 'C' or 'c'
- 4 or 'D' or 'd'

Returns The corresponding *TempicoChannel()* object if the input is valid, or -1 if the channel is invalid.

Return type *TempicoChannel* or int

`getThresholdVoltage`()
Returns the threshold voltage on the rising edge of start and stops inputs of TDCs in the *TempicoDevice()*.

Start and stop inputs are coupled to 50 ohms.

By default, threshold voltage = 1.00V (recommended for TTL>2.5V).

All inputs are 5V tolerant.

Gate input. This parameter does not have effect on the gate input. Gate input accepts 3.3V TTL and 5V TTL signals.

- When gate is disconnected, system is enabled.
- When gate is connected to 0V, system is disabled.
- When gate is connected to 3.3V/5V, system is enabled.

If the connection is established with the *TempicoDevice()*, this function request the device for the value. If not, the most recent value is returned.

Parameters `(none)` –

Returns start and stop inputs threshold voltage.

Return type float

`increaseGeneratorFrequency`()
Increases the internal generator frequency by one step (only for TP12 devices).

This function raises the generator frequency by a predefined increment factor (1, 2, 5, or 10 × 10) depending on the current frequency range.

Parameters `None` –

Returns None

`incrementThresholdVoltage()`
Increases the threshold voltage on the rising edge of start and stops inputs of TDCs in the *TempicoDevice()*.

To validate the actual threshold voltage applied, method *getThresholdVoltage()* should be called.

This function requires that a connection is established with the *TempicoDevice()*.

Parameters `(none)` –

`isEnabled`(*channel*)
Returns whether the specified *TempicoChannel()* is enabled.

By default, channels are enabled.

If the connection is established with the *TempicoDevice()*, this function delegates the query to the selected channel object and retrieves its current enable status. If the connection is not open, or the channel is invalid, the function returns -1.

This version belongs to the *TempicoDevice()* class and requires specifying the channel number or label ('A'–'D') to access and query the corresponding *TempicoChannel()*.

Parameters `channel` (*int or str*) – Channel identifier. Accepted values are:

- 1 or 'A' or 'a'
- 2 or 'B' or 'b'
- 3 or 'C' or 'c'
- 4 or 'D' or 'd'

Returns True if the channel is enabled, False if disabled, or -1 if the channel is invalid.

Return type bool or int

`isIdle()`
Finds if every channel in the *TempicoDevice()* is in the idle state.

This method is used to validate if a reset command has been successfully applied.

Returns True if every channel is idle.

Return type bool

`isIdleOrDisabled()`
Finds if every channel in the *TempicoDevice()* is in the idle state or in the disabled state.

This method is used to validate if an abort command has been successfully applied.

Returns True if every channel is either idle or disabled.

Return type bool

`isOpen()`
Returns if a TDC *TempicoDevice()* connection is established (open).

Parameters `(none)` –

Returns True when *TempicoDevice()* connection is open.

Return type bool

`isPendingReadMessage()`
Determines if a pending message is available to be read in a *TempicoDevice()* serial port.

Parameters **(none)** –

Returns True, when a pending message is found.

Return type bool

`isValidConsecutiveNumberListRows(row, next_row)`
Pair of consecutive rows validation of a number 2D-list (read data).

Validations applied:

- length of next_row
- sequential run values
- increasing start

A single row of the dataset of a *TempicoDevice()* is in the following format:

[ch,run,start_s,stop_ps1,...,stop_psN]

where

- 'ch' indicates the TDC channel,
- 'run' goes from 1 to NumberOfRuns,
- 'start_s' is the epoch timestamp of start pulse, in seconds, with microseconds precision; this value overflows (go back to zero) after 2^32 seconds,
- 'stop_ps1' is the measured precision timelapse between start and the 1st stop pulse, in picoseconds,
- 'N' is the NumberOfStops.

Parameters

- **row** (*list(number)*) – single line of dataset message converted.
- **next_row** (*list(number)*) – next single line of dataset message converted.

Returns True when pair of rows passes all validations.

Return type bool

`isValidNumberListRow(row)`
Single row validation of a number 2D-list (read data).

Validations applied:

- length of row
- type of data (integers, except for start_s register)
- ch in range
- run in range
- sorted stops
- stops in range

A single row of the dataset of a *TempicoDevice()* is in the following format:

[ch,run,start_s,stop_ps1,...,stop_psN]

where

- 'ch' indicates the TDC channel,
- 'run' goes from 1 to NumberOfRuns,
- 'start_s' is the epoch timestamp of start pulse, in seconds, with microseconds precision; this value overflows (go back to zero) after 2^32 seconds,
- 'stop_ps1' is the measured precision timelapse between start and the 1st stop pulse, in picoseconds,
- 'N' is the NumberOfStops.

Parameters **row** (*list(number)*) – single line of dataset message converted.

Returns True when single row passes all validations.

Return type bool

measure(*validate=True*)

Begins a measurement sequence and reads its dataset from a *TempicoDevice()*.

The dataset of a *TempicoDevice()* is in the following format:

```
[[ch,run,start_s,stop_ps1,...,stop_psN],...,[ch,run,start_time_us,stop_ps1,...,
↪stop_psN]]
```

where

- 'ch' (int) indicates the TDC channel,
- 'run' (int) goes from 1 to NumberOfRuns,
- 'start_s' (float) is the epoch timestamp of start pulse, in seconds with microsecond resolution. This value overflows (go back to zero) after 2^32 seconds,
- 'stop_ps1' (int) is the measured precision timelapse between start and the 1st stop pulse, in picoseconds,
- 'N' (int) is the NumberOfStops.

Every value in the dataset is either an integer or a float number.

If measurement cannot be completed within timeout, the device may respond with an incomplete or empty dataset. In this case, to obtain a complete dataset, the method *fetch()* may be called later.

This function requires that a connection is established with the *TempicoDevice()*.

Parameters **validate** (*bool, optional*) – If True, the measured dataset is validated, and invalid registers get cleaned. Default is True.

Returns measured dataset.

Return type list(number)

open()

Establishes (opens) a connection with a *TempicoDevice()*.

It is mandatory to establish a connection with this method before to be able to send/to receive data to/from the device.

Parameters **(none)** –

openTempico()
Establishes (opens) a connection with a *TempicoDevice()*.

Same as method *open()*.

Parameters **(none)** –

readIdnFromDevice()
Returns the *TempicoDevice()* identification string, by requesting it to the device.

This function requires that a connection is established with the *TempicoDevice()*. As an alternative, method *getIdn()* may be used.

Parameters **(none)** –

Returns identification string.

Return type str

readMessage()
Reads pending messages sent by a *TempicoDevice()* from its serial port.

If no message is received, it waits the port timeout, typically 1s.

This function requires that a connection is established with the *TempicoDevice()*.

Parameters **(none)** –

Returns read message.

Return type str

reset(*validate=True*)
Sends a reset command to the *TempicoDevice()*.

Applying a reset clears all the settings of the *TempicoDevice()* and its TempicoChannels to their default values.

This function requires that a connection is established with the *TempicoDevice()*.

Parameters **validate** (*bool, optional*) – If True, waits and validates if every channel applies the reset sucessfully. Default is True.

selfTest()
Performs a self-test on the *TempicoDevice()* hardware.

This function initiates a self-diagnostic test on the *TempicoDevice()* to verify its hardware integrity. If the self-test is successful, it prints the message "Self test passed. Device is working properly." If the self-test fails, it prints the message "Self test failed. Device may have a problem.", indicating a potential issue with the hardware that may require further investigation or support.

This function requires that a connection is established with the *TempicoDevice()*.

Parameters **(none)** –

Returns None

setAverageCycles(*channel*, *averageCycles*)
Modifies the average cycles of the specified *TempicoChannel()*.

By default, average cycles = 1 (no multi-cycle averaging).

This function requires that a connection is established with the *TempicoDevice()*, and delegates the configuration to the selected *TempicoChannel()*. If the channel is invalid, the function does nothing.

This version belongs to the *TempicoDevice()* class and requires specifying the channel number or label ('A'–'D') to access and configure the corresponding *TempicoChannel()*.

Parameters

- **channel** (*int or str*) – Channel identifier. Accepted values are:
  - 1 or 'A' or 'a'
  - 2 or 'B' or 'b'
  - 3 or 'C' or 'c'
  - 4 or 'D' or 'd'
- **averageCycles** (*int*) – Desired average cycles for the TDC. Valid values are: 1, 2, 4, 8, 16, 32, 64, 128.

setDateTime(*timeStampDateTime=None*)
Same as setDatetime

setDatetime(*timeStampDateTime=None*)
Sets the internal clock of the Tempico device to a specified timestamp or to the current system time.

Sends a *DTIMe <timestamp>* command using either the provided *timeStampDateTime* or the current time in Unix format with microsecond precision. If no response is received from the device, the function verifies the update by comparing timestamps.

Parameters **timeStampDateTime** (*float, optional*) – Unix timestamp (in seconds) to set on the device. If not provided, the current system time is used.

Returns None

setGeneratorFrequency(*desired_frequency*)
Sets the frequency of the internal pulse generator (only for TP12 devices).

This function changes the frequency of the internal generator using the 'CONF:GEN:FREQ' command. The value must be between 10 Hz and 10 000 000 Hz. The command is verified by reading back the applied frequency and checking that the relative error is below 3%. If the connection is not open or the hardware version is not supported, an informational message is printed.

Parameters **desired_frequency** (*float*) – Desired generator frequency in hertz (10 Hz – 10 000 000 Hz).

Returns None

setMaximumDateTime(*maximum_datetime*)
Same as setMaximumDatetime

setMaximumDatetime(*maximum_datetime*)
Sets the maximum datetime value allowed by the *TempicoDevice()*.

If the connection is established with the *TempicoDevice()*, this function sends the 'DTIMe:MAXimum <timestamp>' command to update the maximum allowed timestamp. If the device accepts the value, it should return a confirmation response. If no response is received, the value is validated by re-reading it from the device.

Parameters **maximum_datetime** (*float*) – New maximum timestamp to configure on the device.

Returns None

setMinimumDateTime(*minimum_datetime*)
Same as setMinimumDatetime

setMinimumDatetime(*minimum_datetime*)
Sets the minimum datetime value allowed by the *TempicoDevice()*.

If the connection is established with the *TempicoDevice()*, this function sends the 'DTIMe:MINimum <timestamp>' command to update the minimum allowed timestamp. If the device accepts the value, it should return a confirmation response. If no response is received, the value is validated by re-reading it from the device.

Parameters **minimum_datetime** (*float*) – New minimum timestamp to configure on the device.

Returns None

setMode(*channel*, *mode*)

Modifies the measurement mode of the specified *TempicoChannel()*.

By default, mode = 1. Possible values are:

- 1: Short measurement range. Start-stop times from 12ns to 500ns.
- 2: Large measurement range. Start-stop times from 125ns to 4ms.

This function requires that a connection is established with the *TempicoDevice()*, and delegates the configuration to the selected *TempicoChannel()*. If the channel is invalid, the function does nothing.

This version belongs to the *TempicoDevice()* class and requires specifying the channel number or label ('A'–'D') to access and configure the corresponding *TempicoChannel()*.

Parameters

- **channel** (*int or str*) – Channel identifier. Accepted values are:
  - 1 or 'A' or 'a'
  - 2 or 'B' or 'b'
  - 3 or 'C' or 'c'
  - 4 or 'D' or 'd'
- **mode** (*int*) – Desired measurement mode for the TDC. Valid values are 1 or 2.

setNumberOfRuns(*number*)

Modifies the number of measurement runs of the TDCs in *TempicoDevice()*.

By default, number of runs = 1 (single measurement).

This function requires that a connection is established with the *TempicoDevice()*.

Parameters **number** (*int*) – desired number of runs for every TDC. Valid values are from 1 to 1000.

setNumberOfStops(*channel*, *numberOfStops*)

Modifies the expected number of stop pulses of the specified *TempicoChannel()*.

By default, number of stops = 1 (single start → single stop).

The TDC must receive all the expected number of stops to register them as a valid measurement; otherwise, the measurements are discarded. For extending the valid time range, consider using measurement mode 2.

This function requires that a connection is established with the *TempicoDevice()*, and delegates the configuration to the selected *TempicoChannel()*. If the channel is invalid, the function does nothing.

This version belongs to the *TempicoDevice()* class and requires specifying the channel number or label ('A'–'D') to access and configure the corresponding *TempicoChannel()*.

Parameters

- **channel** (*int or str*) – Channel identifier. Accepted values are:

  - 1 or 'A' or 'a'
  - 2 or 'B' or 'b'
  - 3 or 'C' or 'c'
  - 4 or 'D' or 'd'
- **numberOfStops** (*int*) – Desired number of stops for the TDC. Valid values are from 1 to 5.

setStartEdge(*channel*, *edge_type*)

Sets the edge type used on start pulses of the specified *TempicoChannel()*.

By default, start edge = 'RISE'. Possible values are:

- 'RISE': TDC timing starts on a rising edge of the start pulse.
- 'FALL': TDC timing starts on a falling edge of the start pulse.

This function requires that a connection is established with the *TempicoDevice()*, and delegates the configuration to the selected *TempicoChannel()*. If the channel is invalid, the function does nothing.

This version belongs to the *TempicoDevice()* class and requires specifying the channel number or label ('A'–'D') to access and configure the corresponding *TempicoChannel()*.

Parameters

- **channel** (*int or str*) – Channel identifier. Accepted values are:
  - 1 or 'A' or 'a'
  - 2 or 'B' or 'b'
  - 3 or 'C' or 'c'
  - 4 or 'D' or 'd'
- **edge_type** (*str or int*) – Desired start edge type for the TDC. Accepted values are 'RISE', 1, 'RIS' or 'FALL', 0, 'FAL'.

setStartExternalSource(*channel*)

Sets the start signal source to external for the specified channel.

This function configures the selected channel so that its start signal comes from an external input instead of the internal generator.

Parameters **channel** (*int*) – Channel number to configure.

Returns None

setStartInternalSource(*channel*)

Sets the start signal source to the internal generator for the specified channel.

This function configures the selected channel so that its start signal is provided by the internal pulse generator instead of an external input.

Parameters **channel** (*int*) – Channel number to configure.

Returns None

setStopEdge(*channel*, *edge_type*)

Sets the edge type used on stop pulses of the specified *TempicoChannel()*.

By default, stop edge = 'RISE'. Possible values are:

- 'RISE': TDC timing ends on a rising edge of the stop pulse.

- 'FALL': TDC timing ends on a falling edge of the stop pulse.

This function requires that a connection is established with the *TempicoDevice()*, and delegates the configuration to the selected *TempicoChannel()*. If the channel is invalid, the function does nothing.

This version belongs to the *TempicoDevice()* class and requires specifying the channel number or label ('A'–'D') to access and configure the corresponding *TempicoChannel()*.

Parameters

- **channel** (*int or str*) – Channel identifier. Accepted values are:
  - 1 or 'A' or 'a'
  - 2 or 'B' or 'b'
  - 3 or 'C' or 'c'
  - 4 or 'D' or 'd'
- **edge_type** (*str or int*) – Desired stop edge type for the TDC. Accepted values are 'RISE', 1, 'RIS' or 'FALL', 0, 'FAL'.

setStopExternalSource(*channel*)

Sets the stop signal source to external for the specified channel.

This function configures the selected channel so that its stop signal comes from an external input instead of the internal generator.

Parameters **channel** (*int*) – Channel number to configure.

Returns None

setStopInternalSource(*channel*)

Sets the stop signal source to the internal generator for the specified channel.

This function configures the selected channel so that its stop signal is provided by the internal pulse generator instead of an external input.

Parameters **channel** (*int*) – Channel number to configure.

Returns None

setStopMask(*channel*, *stop_mask_in_us*)

Modifies the stop mask time of the specified *TempicoChannel()*.

By default, stop mask = 0 (no masking).

The stop mask defines the period (in microseconds) after receiving a start pulse during which stop pulses are ignored. This can help suppress unwanted noise or early signals.

This function requires that a connection is established with the *TempicoDevice()*, and delegates the configuration to the selected *TempicoChannel()*. If the channel is invalid, the function does nothing.

This version belongs to the *TempicoDevice()* class and requires specifying the channel number or label ('A'–'D') to access and configure the corresponding *TempicoChannel()*.

Parameters

- **channel** (*int or str*) – Channel identifier. Accepted values are:
  - 1 or 'A' or 'a'
  - 2 or 'B' or 'b'
  - 3 or 'C' or 'c'
  - 4 or 'D' or 'd'

- **stop_mask_in_us** (*int*) – Desired stop mask time in microseconds. Valid values are from 0 to 4000.

setThresholdVoltage(*desired_voltage*)

Changes the threshold voltage on the rising edge of start and stops inputs of TDCs in the *TempicoDevice()*.

Start and stop inputs are coupled to 50 ohms.

By default, threshold voltage = 1.00V (recommended for TTL>2.5V).

All inputs are 5V tolerant.

Gate input. This parameter does not have effect on the gate input. Gate input accepts 3.3V TTL and 5V TTL signals.

- When gate is disconnected, system is enabled.
- When gate is connected to 0V, system is disabled.
- When gate is connected to 3.3V/5V, system is enabled.

To validate the actual threshold voltage applied, method *getThresholdVoltage()* should be called.

This function requires that a connection is established with the *TempicoDevice()*.

Parameters **desired_voltage** (*float*) – desired start and stop inputs threshold voltage. Valid parameters are MINimum|MAXimum|DOWN|UP or a number from 0.90 to 1.60.

setThresholdVoltageToMaximum()

Sets to the maximum valid value the threshold voltage on the rising edge of start and stops inputs of TDCs in the *TempicoDevice()*.

To validate the actual threshold voltage applied, method *getThresholdVoltage()* should be called.

This function requires that a connection is established with the *TempicoDevice()*.

Parameters **(none)** –

setThresholdVoltageToMinimum()

Sets to the minimum valid value the threshold voltage on the rising edge of start and stops inputs of TDCs in the *TempicoDevice()*.

To validate the actual threshold voltage applied, method *getThresholdVoltage()* should be called.

This function requires that a connection is established with the *TempicoDevice()*.

Parameters **(none)** –

waitAndReadMessage(*wait_time_ms=1*)

Waits the specified time, and then reads pending messages sent by a *TempicoDevice()* from its serial port, if any.

If no message is received, it does not wait for a port timeout.

This function requires that a connection is established with the *TempicoDevice()*.

Parameters **wait_time_ms** (*int, optional*) – Waiting time, in miliseconds. Defaults to 1.

Returns read message.

Return type str

writeMessage(*message*)

Writes a message to a *TempicoDevice()* in its serial port.

If a response is expected after writing a message, the *readMessage()* method should be called afterwards to obtain the response.

This function requires that a connection is established with the *TempicoDevice()*.

Parameters **message** (*str*) – message to be sent.

1.3 TempicoChannel class

class pyTempico.core.**TempicoChannel**(*id_device*, *ch_num*)

Single channel on a Tempico Device.

To modify or access attributes, **please use methods**. For example, to get average cycles on channel 2,

```
>>> my_tempico_device_object.ch2.getAverageCycles()
```

or, as an alternative, send the channel number as a parameter

```
>>> my_tempico_device_object.getAverageCycles(2)
```

Changing attributes without using methods, do not change the actual parameters in the device.

Accesing attributes without using methods, returns values registered in local memory that may not be updated.

id_tempico_channel

Unique identifier for a *TempicoChannel()* object.

Type int

id_tempico_device

Identifier of the *TempicoDevice()* linked to this *TempicoChannel()* object.

Type int

average_cycles

Average cycles.

Type int

channel_number

Number of the channel in the device (1=A, 2=B,...).

Type int

enable

True when the channel is enabled.

Type bool

mode

Measurement mode. 1|2.

- 1: Short measurement range. Start-stop times from 12ns to 500ns.
- 2: Large measurement range. Start-stop times from 125ns to 4ms.

Type int

number_of_stops

Number of stop pulses expected after a start pulse arrives. 1..5.

Type int

parent_tempico_device
Pointer reference to parent object of *TempicoDevice()* class.

Type *TempicoDevice*

start_edge
Edge type on the start pulse used to begin timing. RISE|FALL.

Type str

stop_edge
Edge type on the stop pulses used to end timing. RISE|FALL.

Type str

stop_mask
Time that stop pulses are ignored after receiving a start pulse on the TDC. Value in microseconds. 0..4000.

Type int

disableChannel()
Disables a TDC *TempicoChannel()*.

By default, channels are enabled.

This function requires that a connection is established with the *TempicoDevice()* of the *TempicoChannel()*.

To validate the status of the channel, method *isEnabled()* may be used.

Parameters **(none)** –

enableChannel()
Enables a TDC *TempicoChannel()*.

By default, channels are enabled.

This function requires that a connection is established with the *TempicoDevice()* of the *TempicoChannel()*.

To validate the status of the channel, method *isEnabled()* may be used.

Parameters **(none)** –

getAverageCycles()
Returns the average cycles of the TDC *TempicoChannel()*.

By default, average cycles = 1 (no multi-cycle averaging).

If the connection is established with the *TempicoDevice()*, this function request the device for the value. If not, the most recent value is returned.

Parameters **(none)** –

Returns Number of average cycles.

Return type integer

getDelay()
Returns the delay value for this channel (only for TP12 devices).

This function queries the device using the 'CONFigure:CHx:DELay?' command to obtain the delay assigned to the current channel. If the connection is not open or the hardware version is not supported, an informational message is printed.

Parameters **None** –

Returns Delay value of the channel. Returns -1000000 if it cannot be retrieved.

Return type float

`getMode()`
Returns the measurement mode of the TDC *TempicoChannel()*.

By default, mode = 1.

If the connection is established with the *TempicoDevice()*, this function request the device for the value. If not, the most recent value is returned.

Parameters `(none)` –

Returns

Mode. Possible values are,

- 1: Short measurement range. Start-stop times from 12ns to 500ns.
- 2: Large measurement range. Start-stop times from 125ns to 4ms.

Return type integer

`getNumberOfStops()`
Returns the expected number of stop pulses of the TDC *TempicoChannel()*.

By default, number of stops = 1 (single start -> single stop).

The TDC must receive all the expected number of stops to register them as a valid measurement; otherwise, the measurements are discarded.

If the connection is established with the *TempicoDevice()*, this function request the device for the value. If not, the most recent value is returned.

Parameters `(none)` –

Returns Number of stops.

Return type integer

`getStartEdge()`
Returns the edge type used on start pulses of the TDC *TempicoChannel()*.

By default, start edge = 'RISE'.

If the connection is established with the *TempicoDevice()*, this function request the device for the value. If not, the most recent value is returned.

Parameters `(none)` –

Returns

start edge type. Possible values are,

- 'RISE': TDC timing starts on a rising edge of the start pulse.
- 'FALL': TDC timing starts on a falling edge of the start pulse.

Return type string

`getStartSource()`
Returns the start signal source for this channel (only for TP12 devices).

This function determines whether the start signal of the current channel comes from the internal generator or from an external input.

Parameters `None` –

Returns Signal source identifier (“INTernal” or “EXTernal”).

Return type str

`getState()`
Returns the state of a *TempicoChannel()*.

This function is used to validate if a reset or an abort command has been successfully applied.

Some possible states are:

- 0: disabled.
- 1: idle, enabled.
- 10: processing a reset.
- 11: processing an abort.

other states are related with the measurement process.

Parameters **(none)** –

Returns state.

Return type int

`getStatus()`
Returns the internal status of a *TempicoChannel()*.

This function is used to obtain the state of a channel, used to validate if a reset or an abort command has been successfully applied.

Parameters **(none)** –

Returns status fields and values.

Return type dict

`getStopEdge()`
Returns the edge type used on stop pulses of the TDC *TempicoChannel()*.

By default, stop edge = ‘RISE’.

If the connection is established with the *TempicoDevice()*, this function request the device for the value. If not, the most recent value is returned.

Parameters **(none)** –

Returns

stop edge type. Possible values are,

- ’RISE’: TDC timing ends on a rising edge of the stop pulse.
- ’FALL’: TDC timing ends on a falling edge of the stop pulse.

Return type string

`getStopMask()`
Returns the time that stop pulses are ignored after receiving a start pulse on the TDC *TempicoChannel()*. In microseconds.

By default, stop mask = 0 (no masking).

If the connection is established with the *TempicoDevice()*, this function request the device for the value. If not, the most recent value is returned.

Parameters **(none)** –

Returns stop mask time, in microseconds.

Return type integer

`getStopSource()`
Returns the stop signal source for this channel (only for TP12 devices).

This function determines whether the stop signal of the current channel comes from the internal generator or from an external input.

Parameters `None` –

Returns Signal source identifier ("INTernal" or "EXTernal").

Return type str

`isEnabled()`
Returns if a TDC *TempicoChannel()* is enabled.

By default, channels are enabled.

If the connection is established with the *TempicoDevice()*, this function request the device for the value. If not, the most recent value is returned.

Parameters `(none)` –

Returns True, when TDC channel is enabled.

Return type bool

`setAverageCycles(number)`
Modifies the average cycles of the TDC *TempicoChannel()*.

By default, average cycles = 1 (no multi-cycle averaging).

This function requires that a connection is established with the *TempicoDevice()* of the *TempicoChannel()*.

Parameters **number** (*int*) – desired average cycles for the TDC. Valid values are 1|2|4|8|16|32|64|128.

`setMode(number)`
Modifies the measurement mode of the TDC *TempicoChannel()*.

By default, mode = 1. Possible values are,

- 1: Short measurement range. Start-stop times from 12ns to 500ns.
- 2: Large measurement range. Start-stop times from 125ns to 4ms.

This function requires that a connection is established with the *TempicoDevice()* of the *TempicoChannel()*.

Parameters **number** (*int*) – desired measurement mode for the TDC. Valid values are 1 or 2.

`setNumberOfStops(number)`
Modifies the expected number of stop pulses of the TDC *TempicoChannel()*.

By default, number of stops = 1 (single start -> single stop).

The TDC must receive all the expected number of stops to register them as a valid measurement; otherwise, the measurements are discarded. For extending the valid time range, consider using measurement mode 2.

This function requires that a connection is established with the *TempicoDevice()* of the *TempicoChannel()*.

Parameters **number** (*int*) – desired number of stops for the TDC. Valid values are from 1 to 5.

setStartEdge(*edge_type*)
Sets the edge type used on start pulses of the TDC *TempicoChannel()*.

By default, start edge = 'RISE'. Possible values are,

- 'RISE': TDC timing starts on a rising edge of the start pulse.
- 'FALL': TDC timing starts on a falling edge of the start pulse.

This function requires that a connection is established with the *TempicoDevice()* of the *TempicoChannel()*.

Parameters **edge_type** (*str*) – desired start edge type for the TDC. Valid values are 'RISE', 1, 'FALL', 0.

setStartExternalSource()
Sets the start signal source to external (only for TP12 devices).

This function configures the start input of the current channel so that its signal comes from an external source instead of the internal generator.

Parameters **None** –

Returns None

setStartInternalSource()
Sets the start signal source to the internal generator (only for TP12 devices).

This function configures the start input of the current channel so that its signal is provided by the internal pulse generator instead of an external source.

Parameters **None** –

Returns None

setStopEdge(*edge_type*)
Sets the edge type used on stop pulses of the TDC *TempicoChannel()*.

By default, stop edge = 'RISE'. Possible values are,

- 'RISE': TDC timing ends on a rising edge of the stop pulse.
- 'FALL': TDC timing ends on a falling edge of the stop pulse.

This function requires that a connection is established with the *TempicoDevice()* of the *TempicoChannel()*.

Parameters **edge_type** (*str*) – desired stop edge type for the TDC. Valid values are 'RISE', 1, 'FALL', 0.

setStopExternalSource()
Sets the stop signal source to external (only for TP12 devices).

This function configures the stop input of the current channel so that its signal comes from an external source instead of the internal generator.

Parameters **None** –

Returns None

setStopInternalSource()
Sets the stop signal source to the internal generator (only for TP12 devices).

This function configures the stop input of the current channel so that its signal is provided by the internal pulse generator instead of an external source.

Parameters **None** –

Returns None

setStopMask(*stop_mask_in_us*)

Modifies the time that stop pulses are ignored after receiving a start pulse on the TDC *TempicoChannel()*.

By default, stop mask = 0 (no masking).

This function requires that a connection is established with the *TempicoDevice()* of the *TempicoChannel()*.

Parameters **stop_mask_in_us** (*int*) – desired stop mask for the TDC, in microseconds. Valid values are from 0 to 4000.

CHAPTER

TWO

EXAMPLES

2.1 readSettingsExample.py

```
# -*- coding: utf-8 -*-
"""readSettingsExample

    Created on Tue May  7 09:49 2024

    Connects to a Tausand Tempico device and reads its settings.

    Instructions:
        * before running this example, pyTempico package must be installed.
        * change 'my_port' to your corresponding port.
        * run.

    | @author: David Guzman at Tausand Electronics
    | dguzman@tausand.com
    | https://www.tausand.com
"""

import pyTempico

my_port = 'COM17' #change this port to your Tempico device's port

my_device = pyTempico.TempicoDevice(my_port)    #create object

print('opening connection with device in port',my_port)
my_device.open()             #open connection with device
if my_device.isOpen():
    print('connection with my_device is open')
else:
    print('connection with my_device is close')

print('\n1) reading general getSettings')
my_settings = my_device.getSettings()
print('settings:')
print(my_settings)

print('\n2) reading specific settings')
print('NumberOfRuns:', my_device.getNumberOfRuns())
```

(continues on next page)

(continued from previous page)

```
print('ThresholdVoltage:', my_device.getThresholdVoltage())

print('\nreading channel specific settings')
print('Ch1 AverageCycles:', my_device.ch1.getAverageCycles())
print('Ch2 AverageCycles:', my_device.ch2.getAverageCycles())
print('Ch3 AverageCycles:', my_device.ch3.getAverageCycles())
print('Ch4 AverageCycles:', my_device.ch4.getAverageCycles())
print('Ch1 Mode:', my_device.ch1.getMode())
print('Ch1 NumberOfStops:', my_device.ch1.getNumberOfStops())
print('Ch1 isEnabled:', my_device.ch1.isEnabled())

print('\nclosing connection with device in port',my_port)
my_device.close()            #close connection with device
if my_device.isOpen():
    print('connection with my_device is open')
else:
    print('connection with my_device is close')
```

2.2 writeSettings.py

```
# -*- coding: utf-8 -*-
"""writeSettings

    Created on Tue Jul  14 16:42 2025

    This example connects to a Tausand Tempico device and demonstrates how to
    read and modify its basic settings, including average cycles, number of stops,
    and stop mask - both using direct channel access and the main device interface.

    Instructions:
        - Make sure the pyTempico package is installed.
        - Replace 'COM5' with the appropriate serial port for your Tempico device.
        - Run the script.

    Author: Joan Amaya, Tausand Electronics
    Email: jamaya@tausand.com
    Website: https://www.tausand.com
"""

import pyTempico

# Set your device port here
my_port = 'COM5'
my_device = pyTempico.TempicoDevice(my_port)

print(f"\nOpening connection on port {my_port}...")
my_device.open()

if my_device.isOpen():
    print("Device connection is open.")
```

(continues on next page)

(continued from previous page)

```
else:
    print("Failed to open device connection.")

# Reset to default settings before making changes
print("\nResetting device to default settings...")
my_device.reset()

# Read initial settings
print("\n[1] Reading default settings with my_device.getSettings():")
settings = my_device.getSettings()
print(settings)

# Modify average cycles (Channel 1)
print("\n[2] Setting average cycles on Channel 1 using direct channel method (Channel 1␣
↪→ 4 cycles):")
my_device.ch1.setAverageCycles(4)
print(f"New value: {my_device.ch1.getAverageCycles()}")

# Reset and apply the same setting using device-level method
print("\nResetting...")
my_device.reset()
print("\n[3] Setting average cycles via device method (Channel 1 → 8 cycles):")
my_device.setAverageCycles(1, 8)
print(f"Verified: {my_device.getAverageCycles(1)}")

# Modify number of stops
print("\nResetting...")
my_device.reset()
print("\n[4] Setting number of stops on Channel 1 using direct method (Channel 1 → 5␣
↪stops):")
my_device.ch1.setNumberOfStops(5)
print(f"New value: {my_device.ch1.getNumberOfStops()}")

# Reset and set using device method
print("\nResetting...")
my_device.reset()
print("\n[5] Setting number of stops via device method (Channel 1 → 4 stops):")
my_device.setNumberOfStops(1, 4)
print(f"Verified: {my_device.getNumberOfStops(1)}")

# Modify stop mask
print("\nResetting...")
my_device.reset()
print("\n[6] Setting stop mask on 1000 us in Channel 1 using direct method (Channel 1 →␣
↪1000):")
my_device.ch1.setStopMask(1000)
print(f"New value: {my_device.ch1.getStopMask()}")

# Reset and set using device method
print("\nResetting...")
my_device.reset()
print("\n[7] Setting stop mask on 2000 us via device method (Channel 1 → 2000):")
```

(continues on next page)

(continued from previous page)

```
my_device.setStopMask(1, 2000)
print(f"Verified: {my_device.getStopMask(1)}")

# Close connection
print(f"\nClosing connection on port {my_port}...")
my_device.close()

if my_device.isOpen():
    print("Connection is still open.")
else:
    print("Connection closed successfully.")
```

2.3 dateTimeExample.py

```
# -*- coding: utf-8 -*-
"""
dateTimeExample

This script demonstrates how to connect to a Tausand Tempico device and
perform several operations related to time management and measurement flow.

Steps:
- Open connection to the device.
- Reset and configure channels.
- Read and synchronize time.
- Modify allowed datetime range.
- Perform a measurement.

You do **not** need a connected signal to run this example.

Instructions:
1. Ensure `pyTempico` is installed.
2. Replace `'COM5'` with your Tempico port.
3. Run the script.

Author: Joan Amaya, Tausand Electronics
    Email: jamaya@tausand.com
    Website: https://www.tausand.com
"""
import pyTempico

my_port = 'COM5' #change this port to your Tempico device's port

my_device = pyTempico.TempicoDevice(my_port)    #create object

print('opening connection with device in port',my_port)
my_device.open()             #open connection with device
if my_device.isOpen():
    print('connection with my_device is open')
else:
```

(continues on next page)

(continued from previous page)

```
    print('connection with my_device is close')

print('\nreseting device. This clears previous measurements, and changes settings to␣
↪default values.')
my_device.reset()

#Enable channel 1, disable channels 2-4
print('\ndisabling channels 2-4.')
my_device.ch1.enableChannel()   #optional, since enabled by default
my_device.ch2.disableChannel()
my_device.ch3.disableChannel()
my_device.ch4.disableChannel()

print("\nWe want to see the current time in seconds with microsecond resolution. By␣
↪default, this time is taken when the device is powered on.\n")
print(f"{my_device.getDateTime()}")

print("\nThis time can also be printed in a formatted date style, which will become more␣
↪useful later on.\n")
print(f"{my_device.getDateTime(True)}")

print("\nThis time can be synchronized by the user, who provides a timestamp in seconds␣
↪with microsecond resolution.\n")

print("\nHowever, this timestamp must fall within the minimum and maximum limits␣
↪accepted by the device.\n")
print(f"Maximum value with my_device.getMaximumDatetime(): {my_device.
↪getMaximumDatetime()}")
print(f"Formatted (with True): {my_device.getMaximumDatetime(True)}")
print(f"Minimum value with my_device.getMinimumDatetime(): {my_device.
↪getMinimumDatetime()}")
print(f"Formatted (with True): {my_device.getMinimumDatetime(True)}")

print("\nNow that we know the allowed range, we'll configure a custom time value.\n")
print("Setting the time using: my_device.setDateTime(3102462800.0)")
my_device.setDateTime(3102462800.0)

print("\nWe can now verify that the time was correctly synchronized.\n")
print(f"Device time via my_device.getDateTime(): {my_device.getDateTime()}")

print("\nAlso we can synchronize the time with our pc, using the same function without␣
↪parameter.\n")
print("Setting the time using: my_device.setDateTime()")
my_device.setDateTime()

print("\nWe can now verify that the time was correctly synchronized.\n")
print(f"Device time via my_device.getDateTime(): {my_device.getDateTime()}")
print(f"With a date format now my_device.getDateTime(True): {my_device.getDateTime(True)}
↪")


print("\nIf we want to use a timestamp outside the current limits, we can update the␣
↪allowed range.\n")
```

(continues on next page)

(continued from previous page)

```
print(f"Setting new maximum with my_device.setMaximumDatetime(4103462800.0).")
my_device.setMaximumDatetime(4103462800.0)
print(f"Setting new minimum with my_device.setMinimumDatetime(1576854800.0).")
my_device.setMinimumDatetime(1576854800.0)

print("\nNow we can retrieve the new minimum and maximum values.\n")
print(f"New maximum: my_device.getMaximumDatetime(): {my_device.getMaximumDatetime()}")
print(f"New minimum: my_device.getMinimumDatetime(): {my_device.getMinimumDatetime()}")

print("\nThese values are useful to determine when a start signal was received. We will␣
↪now perform a measurement - a connected start signal is required.\n")

print("Disabling channels 2-4.")
my_device.ch1.enableChannel()
my_device.ch2.disableChannel()
my_device.ch3.disableChannel()
my_device.ch4.disableChannel()

print("\nStarting a measurement.\n")
my_device.measure()

print("my_device.fetch():")
data = my_device.fetch()
print(data)

print("\nNote that the start time aligns with the synchronized time. Additionally, we␣
↪can retrieve the time of the last start and the last synchronization (i.e., from␣
↪setDateTime()).\n")
print(f"Last start via my_device.getLastStart(): {my_device.getLastStart()}")
print(f"Last start (formatted): {my_device.getLastStart(True)}")
print(f"Last synchronization via my_device.getLastSync(): {my_device.getLastSync()}")
print(f"Last synchronization (formatted): {my_device.getLastSync(True)}")

my_device.close()
if my_device.isOpen():
    print('connection with my_device is open')
else:
    print('connection with my_device is close')
```

2.4 singleMeasureExample.py

```
# -*- coding: utf-8 -*-
"""singleMeasureExample

    Created on Tue May  7 09:23 2024

    Connects to a Tausand Tempico device, starts a measurement and reads the
    results.

    Instructions:
```

(continues on next page)

(continued from previous page)

```
        * before running this example, pyTempico package must be installed.
        * change 'my_port' to your corresponding port.
        * connect signals to your Tempico Device. If no signals are measured,
        this example will return an empty data array.
        * run.

    | @author: David Guzman at Tausand Electronics
    | dguzman@tausand.com
    | https://www.tausand.com
"""

import pyTempico

my_port = 'COM16' #change this port to your Tempico device's port

my_device = pyTempico.TempicoDevice(my_port)    #create object

print('opening connection with device in port',my_port)
my_device.open()            #open connection with device
if my_device.isOpen():
    print('connection with my_device is open')
else:
    print('connection with my_device is close')

print('sending a measure request to device')
data = my_device.measure()   #starts a measurement, and saves response in 'data'
print('measured data, in ps:',data)

my_device.close()           #close connection with device
if my_device.isOpen():
    print('connection with my_device is open')
else:
    print('connection with my_device is close')
```

2.5 modeTwoMeasureExample.py

```
# -*- coding: utf-8 -*-
"""modeTwoMeasureExample

    Created on Tue May  7 11:42 2024

    Connects to a Tausand Tempico device, changes settings to measure only in
    channel 1 in mode 2 (125ns-4ms mode). Then, starts a measurement and reads
    the results.

    Instructions:
        * before running this example, pyTempico package must be installed.
        * change 'my_port' to your corresponding port.
        * connect signals to your Tempico Device. If no signals are measured,
        this example will return an empty data array.
```

(continues on next page)

(continued from previous page)

```
        * run.

    | @author: David Guzman at Tausand Electronics
    | dguzman@tausand.com
    | https://www.tausand.com
"""

import pyTempico

my_port = 'COM16' #change this port to your Tempico device's port

my_device = pyTempico.TempicoDevice(my_port)    #create object

print('opening connection with device in port',my_port)
my_device.open()             #open connection with device
if my_device.isOpen():
    print('connection with my_device is open')
else:
    print('connection with my_device is close')

print('\nreseting device. This clears previous measurements, and changes settings to␣
↪default values.')
my_device.reset()

#Enable channel 1, disable channels 2-4
print('\ndisabling channels 2-4.')
my_device.ch1.enableChannel()   #optional, since enabled by default
my_device.ch2.disableChannel()
my_device.ch3.disableChannel()
my_device.ch4.disableChannel()

print('\nchanging measurement mode in channel 1 to mode 2')
my_device.ch1.setMode(2)    #default mode is 1, changing to mode 2
#verify
print('my_device.ch1.getMode():',my_device.ch1.getMode())

print('\nsending a measure request to device')
data = my_device.measure()   #starts a measurement, and saves response in 'data'
print('measured data, in ps:',data)

print('fetch:',my_device.fetch()) #fetch most recent data

my_device.close()            #close connection with device
if my_device.isOpen():
    print('connection with my_device is open')
else:
    print('connection with my_device is close')
```

2.6 numberOfRunsMeasureExample.py

```
# -*- coding: utf-8 -*-
"""numberOfRunsMeasureExample

    Created on Tue Jul  14 16:42 2025

    Example script for measuring with a Tausand Tempico device using extended
    number of runs (1000), and using Mode 2 (125 ns - 4 ms range) on Channel 1.

    This example connects to the Tempico device, configures it to only enable
    Channel 1, sets the measurement mode to Mode 2, and adjusts the number of
    runs (measurements) to 1000. It then starts a single measurement operation
    and reads the resulting data.

    The key feature demonstrated here is the use of `setNumberOfRuns(1000)`,
    which tells the device to internally perform 1000 measurements in response
    to a single call to `measure()`. This is ideal for bulk acquisition workflows
    where multiple time intervals are collected automatically.

    Instructions:
        - Make sure the `pyTempico` package is installed.
        - Replace `'COM5'` with the serial port corresponding to your device.
        - Connect appropriate start/stop signals to the Tempico device.
        - Run the script. If no signals are present, the measurement will return
          an empty array.

    Author: Joan Amaya, Tausand Electronics
    Email: jamaya@tausand.com
    Website: https://www.tausand.com
"""
import pyTempico

my_port = 'COM5' #change this port to your Tempico device's port

my_device = pyTempico.TempicoDevice(my_port)    #create object

print('opening connection with device in port',my_port)
my_device.open()             #open connection with device
if my_device.isOpen():
    print('connection with my_device is open')
else:
    print('connection with my_device is close')

print('\nreseting device. This clears previous measurements, and changes settings to␣
↪default values.')
my_device.reset()

#Enable channel 1, disable channels 2-4
print('\ndisabling channels 2-4.')
my_device.ch1.enableChannel()   #optional, since enabled by default
my_device.ch2.disableChannel()
```

(continues on next page)

(continued from previous page)

```
my_device.ch3.disableChannel()
my_device.ch4.disableChannel()

print('\nchanging measurement mode in channel 1 to mode 2')
my_device.ch1.setMode(2)    #default mode is 1, changing to mode 2
#verify
print('my_device.ch1.getMode():',my_device.ch1.getMode())

print('\nchanging number of runs to 1000, this setting is applied to every channel in␣
↪the device')
my_device.setNumberOfRuns(1000)    #default number of run is 1, changing to number of␣
↪run 1000
#verify
print('my_device.getNumberOfRuns():',my_device.getNumberOfRuns())

print('\nsending a measure request to device')
data = my_device.measure()   #starts a measurement, and saves response in 'data'
print('measured data, in ps:',data)

print('fetch:',my_device.fetch()) #fetch most recent data
print('\nIf a measurement was succesful, you will see 1000 different measurements with a␣
↪single call of the measurement function.')

my_device.close()
if my_device.isOpen():
    print('connection with my_device is open')
else:
    print('connection with my_device is close')
```

2.7 numberOfStopsMeasureExample.py

```
# -*- coding: utf-8 -*-
"""numberOfStopsMeasureExample

    Created on Tue Jul  14 16:42 2025

    Example script for measuring with a Tausand Tempico device using Mode 2
    (125 ns - 4 ms range) on Channel 1, with multiple stops per measurement.

    This example connects to the Tempico device, enables only Channel 1, sets
    the measurement mode to Mode 2, and changes the number of stops to 5. It
    then starts a measurement and prints the result.

    The key feature demonstrated here is the use of `setNumberOfStops(5)`,
    which configures the device to collect **5 stop events** after a single
    start signal during each measurement cycle. This is useful for analyzing
    multiple photon arrivals or signal repetitions after a common trigger.

    Instructions:
        - Make sure the `pyTempico` package is installed.
```

(continues on next page)

(continued from previous page)

```
        - Replace `'COM5'` with the serial port corresponding to your device.
        - Connect a periodic signal to the **start input**, and the signal to
          be measured to the stop input of Channel 1.
        - Run the script. If no signals are received, the measurement will
          return an empty array.

    Author: Joan Amaya, Tausand Electronics
    Email: jamaya@tausand.com
    Website: https://www.tausand.com
"""
import pyTempico

my_port = 'COM5' #change this port to your Tempico device's port

my_device = pyTempico.TempicoDevice(my_port)    #create object

print('opening connection with device in port',my_port)
my_device.open()             #open connection with device
if my_device.isOpen():
    print('connection with my_device is open')
else:
    print('connection with my_device is close')

print('\nreseting device. This clears previous measurements, and changes settings to␣
↪default values.')
my_device.reset()

#Enable channel 1, disable channels 2-4
print('\ndisabling channels 2-4.')
my_device.ch1.enableChannel()   #optional, since enabled by default
my_device.ch2.disableChannel()
my_device.ch3.disableChannel()
my_device.ch4.disableChannel()

print('\nchanging measurement mode in channel 1 to mode 2')
my_device.ch1.setMode(2)    #default mode is 1, changing to mode 2
#verify
print('my_device.ch1.getMode():',my_device.ch1.getMode())

print('\nchanging number of stops in channel 1 to 5')
my_device.ch1.setNumberOfStops(5)    #default number of stops is 1, changing number of␣
↪stops to 5
#verify
print('my_device.ch1.getNumberOfStops():',my_device.ch1.getNumberOfStops())

print('\nsending a measure request to device')
data = my_device.measure()   #starts a measurement, and saves response in 'data'
print('measured data, in ps:',data)

print('fetch:',my_device.fetch()) #fetch most recent data
print('\nYou will see 5 stop data per measurement, these stops are captured after the␣
↪same start signal.')
```

(continues on next page)

(continued from previous page)

```
my_device.close()
if my_device.isOpen():
    print('connection with my_device is open')
else:
    print('connection with my_device is close')
```

CHAPTER

THREE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

p

INDEX

I

M

N

O

P

R

S

T

V

W