Tempico Software

Release 1.2.0

Tausand Electronics

Jun 30, 2025

CONTENTS:

Tempico Software is a program developed by Tausand Electronics, with Joan Amaya and David Guzman as the lead developers. It is implemented in Python, utilizing key libraries such as PyQt5 (integrated with PySide2) for the graphical user interface, HIDAPI for serial communication, and PyGraph for data visualization. The software facilitates the use of Tausand Tempico TP1004 devices, streamlining the creation of histograms and lifetime measurements. For more information, visit tausand.com.

CHAPTER

ONE

LIFETIMEGRAPHICS MODULE

class LifeTimeGraphics.**LifeTimeGraphic**(*comboBoxStartChannel: QComboBox*, *comboBoxStopChannel: QComboBox*, *graphicFrame: QFrame*, *startButton: QPushButton*, *stopButton: QPushButton*, *initialParametersButton: QPushButton*, *clearButton: QPushButton*, *saveDataButton: QPushButton*, *savePlotButton: QPushButton*, *statusLabel: QLabel*, *pointLabel: QLabel*, *binWidthComboBox: QComboBox*, *numberBins: QComboBox*, *functionComboBox: QComboBox*, *numberMeasurementsSpinBox: QSpinBox*, *totalMeasurements: QLabel*, *totalStart: QLabel*, *totalTime: QLabel*, *timeRange: QLabel*, *device*, *applyButton: QPushButton*, *parameterTable: QTableWidget*, *MainWindow*, *timerConnection: QTimer*)

Bases: `object`

Class responsible for the logic and functionality of the LifeTime (Fluorescence Lifetime Measurement) window.

This class manages the creation and updating of graphical plots related to LifeTime, controls user interactions with various buttons, and ensures that measurements are taken from the specified channels. The class handles: - Initialization of the graphical interface elements. - Starting and stopping of measurements. - Saving of data and plots. - Displaying parameters of the measurements, such as the number of measurements taken, and the percentage of completion.

Parameters

- **comboBoxStartChannel** – The combo box for selecting the start channel (QComboBox).
- **comboBoxStopChannel** – The combo box for selecting the stop channel (QComboBox).
- **graphicFrame** – The frame that holds the graphical plot (QFrame).
- **startButton** – The button to start the measurements (QPushButton).
- **stopButton** – The button to stop the measurements (QPushButton).
- **initialParametersButton** – The button to configure initial parameters (QPushButton).
- **clearButton** – The button to clear the graphs and data (QPushButton).
- **saveDataButton** – The button to save the measurement data (QPushButton).
- **savePlotButton** – The button to save the graph plot (QPushButton).
- **statusLabel** – The label showing the current status (QLabel).
- **pointLabel** – The label displaying the current point in the measurement (QLabel).
- **binWidthComboBox** – The combo box for selecting the bin width (QComboBox).
- **numberBins** – The combo box for selecting the number of bins (QComboBox).

- **functionComboBox** – The combo box for selecting a function (QComboBox).
- **numberMeasurementsSpinBox** – The spin box for specifying the number of measurements (QSpinBox).
- **totalMeasurements** – The label showing the total number of measurements (QLabel).
- **totalStart** – The label showing the number of measurements taken from the start channel (QLabel).
- **totalTime** – The label showing the total time of measurements (QLabel).
- **timeRange** – The label showing the time range for the measurements (QLabel).
- **device** – The connected Tempico device used for measurements.
- **applyButton** – The button for applying selected parameters (QPushButton).
- **parameterTable** – The table displaying the parameters of the measurements (QTableWidget).
- **MainWindow** – The main window that holds the application UI elements.
- **timerStatus** – The timer used for periodic updates (QTimer).

Returns
None

applyAction()

Captures the action of the 'apply' button, calculates the selected fit, and stores the resulting parameters.

This function determines which fitting function is selected from *functionComboBox* and attempts to calculate the fit using the measured time and data. If the parameters can be determined, they are stored in *FitParameters*. If not, a dialog informs the user that the parameters could not be determined.

Variables assignment: - The parameters *i0*, *tau0*, and other fit-specific variables are calculated using the corresponding fit function. - If the fit is successful, the rounded parameters are stored in *FitParameters*. - If the fit fails or returns "Undefined", a message box is shown to inform the user of the error.

Returns
None

applyInitialDialog()

Updates the initial parameters based on user input from the parameter dialog.

This function is called when the user clicks the "Apply" button in the parameter dialog. It retrieves the values from the input fields corresponding to the selected fitting function and assigns them to the appropriate instance variables.

The following parameters are updated based on the selected fit type: - For Exponential fit: - I0 (initial value) - tau0 (decay constant) - For Kohlrausch fit: - I0 (initial value) - tau0 (decay constant) - beta (exponent) - For Shifted Exponential fit: - I0 (initial value) - tau0 (decay constant) - alpha (shift parameter) - b (baseline value) - For Double Exponential fit: - I0 (initial value) - tau0 (first decay constant) - tau1 (second decay constant) - alpha (mixing coefficient)

Variables assignment: - The updated values are assigned to their respective initial parameter variables. - A boolean flag indicating that initial parameters have changed is set to True for the relevant fit type.

Returns
None

calculateR2(*data*, *fitData*)

Calculates the R^2 value based on the observed and fitted data.

Parameters

- **data** – The observed data points (array-like).
- **fitData** – The fitted data points (array-like).

Returns
The R^2 value indicating the goodness of fit (float).

changeCreatedStatus()

Notifies that the measurement thread has been created.

This function sets the status of thread creation to True.

Returns
None

changeFunction()

Updates the parameter values in the UI based on the selected function in the functionComboBox.

This function checks the current fitting function (e.g., "ExpDecay", "Kohlrausch", "ShiftedExponential", or "DoubleExponential") and updates the parameter labels accordingly. If the parameters have already been calculated, it will display the values along with their uncertainties and units. If the parameters are not available, it sets the values to "Undefined".

Variables assignment: - Parameters like "I0", "tau0", "Beta", "alpha", "b", "R^2", and their respective uncertainties are assigned using *insertParameters()* based on the fitting model. - Units are assigned through the *self.units* attribute for certain parameters (e.g., tau0).

Behavior by function: - ExpDecay: Updates *I0*, *tau0*, and *R^2*. - Kohlrausch: Updates *I0*, *tau0*, *Beta*, and *R^2*. - ShiftedExponential: Updates *I0*, *tau0*, *alpha*, *b*, and *R^2*. - DoubleExponential: Updates *I0*, *tau0*, *tau1*, *alpha*, and *R^2*.

Returns
None

changeStatusColor(*color*)

Changes the color of the point in the status bar. Each color is assigned a specific numeric value.

Parameters
color – The numeric value corresponding to the desired color (int). - 0: Gray - 1: Green - 2: Yellow - 3: Orange

Returns
None

changeStatusLabel(*textValue*)

Changes the text of the status label.

Parameters
textValue – The text to set for the status label (str).

Returns
None

checkDeviceStatus()

Verifies the device's operational status by attempting to read its identification.

This function performs the following actions: - Attempts to read the identification from the device to ensure it is functioning correctly. - If the read operation fails, it stops any running worker thread and disconnects the device from the serial port. - Displays an error message indicating that the connection with the device has been lost.

Returns
None

`clearGraphic`()

Clears the measured data and time lists when a measurement is in progress and also clears the graphic by executing the worker's clear function.

Returns
None

`connectedDevice`(*device*)

Configures the UI options when the Tempico device is connected.

This function enables and disables the appropriate buttons and starts the timer to monitor the device status. - Enables the disconnect button and disables the connect button. - Starts a status timer with a 500 ms interval. - Assigns the connected device to the class attribute and enables the start button.

Parameters
device – The connected Tempico device.

Returns
None

`disconnectedDevice`()

Configures the UI options when the Tempico device is disconnected.

This function disables and enables the appropriate buttons and stops the timer that monitors the device status. - Disables the disconnect button and enables the connect button. - Stops the status timer. - Disables the start button.

Returns
None

`double_Exponential`(*t*, *I0*, *tau0*, *tau1*, *alpha*)

Computes the value of a double exponential decay function.

This function calculates the value of a double exponential decay function using the given parameters: time *t*, initial value *I0*, decay constants *tau0* and *tau1*, and mixing parameter *alpha*.

Variables assignment: - The computed value is returned directly and not assigned to any variable within the function.

Parameters

- **t** – The time at which to evaluate the function (float).
- **I0** – The initial value of the exponential function (float).
- **tau0** – The first decay constant (float).
- **tau1** – The second decay constant (float).
- **alpha** – The mixing parameter that determines the contribution of the first decay (float).

Returns
The value of the double exponential decay function at time *t*.

Return type
float

`enableAfterFinisihThread()`

Executes when a measurement ends, either by pressing the stop button or when it has naturally finished. Re-enables the buttons, combo boxes, and fields necessary for a new measurement or for saving data. Resets the default parameters for the settings.

Returns
None

`exp_decay(t, I0, tau0)`

Computes the value of an exponential decay function.

This function calculates the value of the exponential decay function using the given parameters for time *t*, initial value *I0*, and decay constant *tau0*.

Variables assignment: - The computed value is returned directly and not assigned to any variable within the function.

Parameters

- **t** – The time at which to evaluate the function (float).
- **I0** – The initial value of the exponential function (float).
- **tau0** – The decay constant (float).

Returns
The value of the exponential decay function at time *t*.

Return type
float

`finishedThreadMeasurement()`

Executes when the measurement thread finishes.

This function enables the UI components related to new measurements and resets the settings to default.

Returns
None

`fitDoubleExponential(xData, yData)`

Fits a double exponential decay model to the provided data.

This function fits a double exponential decay model to the given *xData* and *yData* using the initial guesses for the parameters (I0, tau0, tau1, alpha). If the fit is successful, it updates the corresponding values for each parameter and plots the fitted curve. If the fit fails, a warning message is displayed to the user.

Variables assignment: - *self.FitCov*: Stores the covariance values for I0, tau0, tau1, and alpha, or "nan" if not calculable. - *self.xDataFitCopy*: Stores a copy of the *xData* for the fitted curve. - *self.yDataFitCopy*: Stores the computed y-values for the fitted curve. - *self.R2*: Stores the calculated R^2 value for the fit. - *self.curveFit*: Updates the plot with the fitted curve data.

Parameters

- **xData** – The x-axis data points (array-like).
- **yData** – The y-axis data points (array-like).

Returns
Tuple (I0_opt, tau0_opt, tau1_opt, alpha_opt) containing the fitted values of I0, tau0, tau1, and alpha, or ("Undefined", "Undefined", "Undefined", "Undefined") if fitting fails.

Return type
tuple(float, float, float, float) or tuple(str, str, str, str)

`fitExpDecay`(*xData*, *yData*)

Calculates the exponential decay fit for the given data.

This function attempts to fit an exponential decay model to the provided *xData* and *yData*. If successful, the optimal parameters (I0, tau0) are used to update the respective labels in the parameters table along with their uncertainties (covariance), and the fitted curve is plotted. If the fit cannot be performed, it alerts the user via a message box.

Variables assignment: - *self.FitCov*: Contains the covariance values for I0 and tau0, or "nan" if not calculable. - *self.xDataFitCopy*: Stores a copy of the *xData* for the fitted curve. - *self.yDataFitCopy*: Stores the computed y-values for the fitted curve. - *self.R2*: Stores the calculated R^2 value for the fit. - *self.curveFit*: Updates the plot with the fitted curve data.

Parameters

- **xData** – The x-axis data points (array-like).
- **yData** – The y-axis data points (array-like).

Returns

Tuple (I0_opt, tau0_opt) containing the fitted values of I0 and tau0, or ("Undefined", "Undefined") if fitting fails.

Return type

tuple(float, float) or tuple(str, str)

`fitKohlrauschFit`(*xData*, *yData*)

Calculates the Kohlrausch fit for the given data.

This function attempts to fit a Kohlrausch (stretched exponential) model to the provided *xData* and *yData*. If successful, the optimal parameters (I0, tau0, Beta) are used to update the respective labels in the parameters table along with their uncertainties (covariance), and the fitted curve is plotted. If the fit cannot be performed, it alerts the user via a message box.

Variables assignment: - *self.FitCov*: Contains the covariance values for I0, tau0, and Beta, or "nan" if not calculable. - *self.xDataFitCopy*: Stores a copy of the *xData* for the fitted curve. - *self.yDataFitCopy*: Stores the computed y-values for the fitted curve. - *self.R2*: Stores the calculated R^2 value for the fit. - *self.curveFit*: Updates the plot with the fitted curve data.

Parameters

- **xData** – The x-axis data points (array-like).
- **yData** – The y-axis data points (array-like).

Returns

Tuple (I0_opt, tau0_opt, beta_opt) containing the fitted values of I0, tau0, and Beta, or ("Undefined", "Undefined", "Undefined") if fitting fails.

Return type

tuple(float, float, float) or tuple(str, str, str)

`fitShiiftedExponential`(*xData*, *yData*)

Fits a shifted exponential decay model to the provided data.

This function fits a shifted exponential decay model to the given *xData* and *yData* using the initial guesses for the parameters (I0, tau0, alpha, b). If the fit is successful, it updates the corresponding values for each parameter and plots the fitted curve. If the fit fails, a warning message is displayed to the user.

Variables assignment: - *self.FitCov*: Stores the covariance values for I0, tau0, alpha, and b, or "nan" if not calculable. - *self.xDataFitCopy*: Stores a copy of the *xData* for the fitted curve. - *self.yDataFitCopy*:

Stores the computed y-values for the fitted curve. - *self.R2*: Stores the calculated R^2 value for the fit. - *self.curveFit*: Updates the plot with the fitted curve data.

Parameters

- **xData** – The x-axis data points (array-like).
- **yData** – The y-axis data points (array-like).

Returns

Tuple (I0_opt, tau0_opt, alpha_opt, b_opt) containing the fitted values of I0, tau0, alpha, and b, or ("Undefined", "Undefined", "Undefined", "Undefined") if fitting fails.

Return type

tuple(float, float, float, float) or tuple(str, str, str, str)

`getTempicoChannel()`

Assigns the Tempico device channels based on the selected values from the combo boxes. The selected start and stop channels are assigned to the variables *currentStartChannel* and *currentStopChannel*, respectively. It also enables the corresponding channels.

Returns

None

`getUnits(value)`

Receives a numerical value in picoseconds (ps) and returns a list with two values: the appropriate units for the value and a number indicating by how much to divide the value for conversion.

Parameters

value – The numerical value in picoseconds to convert.

Returns

A list containing the appropriate unit as a string and a number indicating the divisor for conversion.

`indexChangeStartChannel()`

Verifies that the options of the start and stop combo boxes are not the same when the start combo box is changed. If they are the same, it reverts to the previously selected option.

Returns

None

`indexChangeStopChannel()`

Verifies that the options of the start and stop combo boxes are not the same when the stop combo box is changed. If they are the same, it reverts to the previously selected option.

Returns

None

`initialParametersDialog()`

Displays a dialog for selecting initial parameters for fitting functions.

This dialog allows the user to select a type of fitting function and input the corresponding parameters. The parameters that can be set include: - I0 (initial value) - tau0 (decay constant) - beta (for Kohlrausch fit) - alpha (for Shifted Exponential fit) - b (for Shifted Exponential fit)

The dialog contains a ComboBox to select the function type, and the input fields dynamically update based on the selected function. It includes "Apply" and "Reset" buttons to apply changes or revert to default values.

Variables assignment: - The input values from the dialog are assigned to respective attributes, but are not directly returned.

Returns
None

`insertParameters`(*index*, *Parameter*, *Value*, *Cov*, *Units*)

Inserts the parameter values into the fitting table at the specified row.

This function updates the table used for displaying the fitted parameters by inserting the parameter name, its value, the covariance (uncertainty), and the units in the specified row of the table.

Variables assignment: - The parameter name is assigned to the first column of the specified row (*index*). - The value of the parameter is assigned to the second column. - The covariance of the parameter is assigned to the third column. - The units of the parameter are assigned to the fourth column.

Parameters

- **index** – The row index where the values will be inserted (int).
- **Parameter** – The name of the parameter to be displayed (str).
- **Value** – The value of the parameter to be displayed (str).
- **Cov** – The covariance (uncertainty) of the parameter (str).
- **Units** – The units of the parameter (str).

Returns
None

`kohl_decay`(*t*, *I0*, *tau0*, *beta*)

Computes the value of a Kohlrausch (stretched exponential) decay function.

This function calculates the value of the stretched exponential (Kohlrausch) decay function using the given parameters: time *t*, initial value *I0*, decay constant *tau0*, and stretching parameter *beta*.

Variables assignment: - The computed value is returned directly and not assigned to any variable within the function.

Parameters

- **t** – The time at which to evaluate the function (float).
- **I0** – The initial value of the exponential function (float).
- **tau0** – The decay constant (float).
- **beta** – The stretching exponent, controlling the deviation from a simple exponential decay (float).

Returns
The value of the stretched exponential decay function at time *t*.

Return type
float

`lostConnection`()

`maxRound`(*string*)

Determines the number of decimal places needed for at least three significant figures.

Parameters
string – The number as a string (str).

Returns
Number of decimal places (int).

`resetInitialDialog()`

Resets the initial parameters in the dialog based on the measured data.

This function is called when the user clicks the "Default Values" button in the parameter dialog. It resets the initial parameters to sensible defaults derived from the measured data. The following resets occur based on the selected fitting function:

- **For Exponential fit:**
  - I0 is set to the maximum value of measured data.
  - tau0 is set to the mean of measured time.
- **For Kohlrausch fit:**
  - I0 is set to the maximum value of measured data.
  - tau0 is set to the mean of measured time.
  - beta is set to 1.0 (default value).
- **For Shifted Exponential fit:**
  - I0 is set to the maximum value of measured data.
  - tau0 is set to the mean of measured time.
  - alpha is set to 0 (default value).
  - b is set to 0 (default value).
- **For Double Exponential fit:**
  - I0 is set to the maximum value of measured data.
  - tau0 is set to the mean of measured time.
  - alpha is set to 0 (default value).
  - tau1 is set to the mean of measured time.

The corresponding input fields in the dialog are updated with these new values. Flags indicating that the initial parameters have not changed are also reset to False for the relevant fit type.

Returns
None

`resetParametersLabels()`

Resets all the fit parameters to 'Undefined' and clears the current fit type.

This function sets the list *FitParameters* to contain "Undefined" values for all parameters and clears *currentFit*. Additionally, it resets the *R2* value to "Undefined". Depending on the selected fit type in *functionComboBox*, the respective parameter labels in the table are updated to reflect "Undefined" values.

Variables assignment: - *FitParameters* is reset to a list of "Undefined". - *currentFit* is cleared to an empty string. - *R2* is reset to "Undefined". - The *insertParameters* method is used to update the table with "Undefined" values for each parameter based on the selected function in *functionComboBox*.

Returns
None

`roundStringPCov(number)`

Converts a number to a string representation in scientific notation with two decimal places.

Parameters
`number` – The input number (float).

Returns
String representation in scientific notation (str).

`saveLifeTimeData()`

Saves the data in the selected format (TXT, CSV, or DAT) based on the user's choice.

Returns
None

`saveMode()`

Saves the current modes of the channels before starting a measurement. The modes are stored in the variables *oldChannelA*, *oldChannelB*, *oldChannelC*, and *oldChannelD*, allowing for restoration to the original settings after the measurement.

Returns
None

`savePlotLifeTime()`

Saves the graph image in the specified format (PNG, TIFF, or JPG) based on the user's selection. :return: None

`setOldMode()`

Restores the modes of the channels after a measurement is completed. The channels are set to their previous modes stored in *oldChannelA*, *oldChannelB*, *oldChannelC*, and *oldChannelD*.

Returns
None

`shifted_decay_function(t, I0, tau0, alpha, b)`

Computes the value of a shifted exponential decay function.

This function calculates the value of a shifted exponential decay function using the given parameters: time *t*, initial value *I0*, decay constant *tau0*, shift parameter *alpha*, and constant offset *b*.

Variables assignment: - The computed value is returned directly and not assigned to any variable within the function.

Parameters

- **t** – The time at which to evaluate the function (float).
- **I0** – The initial value of the exponential function (float).
- **tau0** – The decay constant (float).
- **alpha** – The shift parameter, determining the horizontal shift of the decay function (float).
- **b** – The constant offset added to the decay function (float).

Returns
The value of the shifted exponential decay function at time *t*.

Return type
float

`startMeasurement()`

Initiates the measurement process by clearing previous data, resetting the UI, and starting a new measurement thread.

This function performs several actions to prepare for and start a new measurement: - Stops the status timer and disables/enables relevant buttons. - Clears previous graph data and resets parameter labels. - Saves the current channel modes and resets save parameters. - Updates the status label and color to indicate a running

measurement. - Clears previously measured data and time, resetting the plot. - Retrieves the selected channels and initializes a new measurement thread with the selected parameters. - Connects signals from the worker thread to corresponding slots in the main thread for UI updates.

Returns
None

`startTimer()`

Starts the timer that triggers the update_timer function every second.

This function initiates the timer for the ongoing measurement, allowing the update_timer function to be executed at one-second intervals.

Returns
None

`startTimerConnection()`

Starts the timer to check the device connection status every 500 milliseconds.

This method initiates periodic checks to monitor if the connected device remains active and responsive. The interval is set to 500 ms.

Returns
None

`stopMeasurement()`

Stops the measurement thread if it is created and executes the function enableAfterFinishedThread() if the thread is not created.

Returns
None

`stopTimer()`

Stops the timer that triggers the update_timer function.

This function halts the timer responsible for updating the measurement time and sets the totalTime label to indicate that no measurement is currently running.

Returns
None

`stopTimerConnection()`

Stops the timer responsible for checking the device connection status.

This method halts the periodic checks performed by *timerConnection* to avoid unnecessary polling or to pause the connection validation during inactive states.

Returns
None

`timeRangeValue()`

Converts the value from the timeRange label to picoseconds (ps).

Returns
float The value converted to picoseconds.

`updateLabel(`*units*`)`

Updates the label of the x-axis in the graph with the correct units for the data.

This function takes the units as a string and updates the x-axis label accordingly.

Parameters
units – The units to be displayed on the x-axis label (str).

Returns
None

updateLabels(*totalMeasurements*, *totalStarts*)

Updates the labels for total measurements and total starts with the given values.

This function sets the text of the totalMeasurements and totalStart labels according to the parameters provided.

Parameters

- **totalMeasurements** – The text to set for the total measurements label (str).
- **totalStarts** – The text to set for the total starts label (str).

Returns
None

updateMeasurement(*listOfNewValues*, *domainMeasurement*)

Updates the measured data and time with new values and refreshes the graph.

This function receives a new list of values representing counts and their corresponding update times, assigning them to the variables measuredData and measuredTime, and updates the graph accordingly.

Parameters

- **listOfNewValues** – The new list of measurement values (list).
- **domainMeasurement** – The list of times corresponding to the measurement values (list).

Returns
None

update_timer()

Updates the timer for the ongoing measurement by adding one second.

This function increments the time variable by one second and updates the totalTime label to reflect the new time.

Returns
None

class LifeTimeGraphics.**WorkerThreadLifeTime**(*deviceStartChannel*, *deviceStopChannel*, *binwidthText*, *numberMeasurements*, *device*, *TimeRange*)

Bases: QThread

Worker thread for handling LifeTime (Fluorescence Lifetime Measurement) processing in a separate thread to ensure that the GUI remains responsive during measurements.

This class performs the measurement tasks in the background, processes the data, and communicates with the main thread to update the GUI with the measurement status, results, and other relevant information.

Signals:

- createdSignal: Signal emitted when the worker thread is created.
- statusSignal: Signal emitted to update the status message in the GUI.
- pointSignal: Signal emitted to update the current measurement point.
- updateValues: Signal emitted to update measurement values and corresponding times.

- updateLabel: Signal emitted to update a specific label in the GUI.
- updateMeasurementsLabel: Signal emitted to update the label showing the number of measurements taken.

Parameters

- **deviceStartChannel** – The start channel of the Tempico device used for measurement (TempicoChannel).
- **deviceStopChannel** – The stop channel of the Tempico device used for measurement (TempicoChannel).
- **binwidthText** – The selected bin width for the measurement (str).
- **numberMeasurements** – The number of measurements to be taken (int).
- **device** – The Tempico device used for performing measurements (Tempico).
- **TimeRange** – The time range for the measurements (int).

Returns
None

checkDeviceStatus()

Checks the status of the device by attempting to read a parameter.

If an error occurs during the read operation, the measurement process is stopped.

Returns
None

clear()

Clears the recorded start-stop differences and resets the total measurement count.

This function empties the list that stores the differences between start and stop measurements, and sets the total number of measurements back to zero.

Returns
None

createLifeTimeData()

Generates LifeTime data from the start-stop differences and emits the calculated results to the main thread.

Processes the collected data to determine the number of counts within a given time based on the maximum value of start-stop differences. Normalizes the data according to the determined units and calculates the histogram counts.

Returns
None

createdSignal

disconectedSignal

getBinWidthNumber()

Calculates the bin width in picoseconds based on user input in different units.

Extracts the numerical value and its corresponding unit from the user-provided text, and converts it into picoseconds for further calculations.

Returns
None

getUnits(*picosecondsValue*)

Determines the appropriate units for a given value in picoseconds.

Converts a given value in picoseconds to the most suitable time unit (picoseconds, nanoseconds, microseconds, or milliseconds).

Parameters
picosecondsValue – The value in picoseconds (float).

Returns
A list containing the unit as a string and the factor by which to divide the value (list).

measurementMode()

Sets the measurement mode for the device channels based on the specified time range.

If the time range is less than or equal to 500000 ps , sets all channels to mode 1. Otherwise, sets all channels to mode 2.

Parameters
self – The instance of the class.

Returns
None

pointSignal

run()

Runs the measurement thread, continuously taking measurements based on user-selected values and passing them to the graphs.

Returns
None

staticMetaObject = <PySide2.QtCore.QMetaObject object>

statusSignal

stop()

Stops the measurement process by setting the running flag to False.

Emits a status message indicating that the measurement is ending and updates the status bar color to yellow.

Returns
None

takeMeasurements(*percentage*)

Takes measurements from the device and updates the start-stop differences list.

Configures the device for measurement runs and checks the input and stop channels for data. If no measurements are detected, emits a status signal to update the main thread about the current measurement state.

Parameters
percentage – The percentage of measurement completion (float).

Returns
None

updateLabel

updateMeasurementsLabel

updateValues

CHAPTER

TWO

STARTSTOPHIST MODULE

class StartStopHist.**StartStopLogic**(*parent*, *disconnect*, *device*, *check1*, *check2*, *check3*, *check4*, *startbutton*, *stopbutton*, *savebutton*, *save_graph_1*, *clear_channel_A*, *clear_channel_B*, *clear_channel_C*, *clear_channel_D*, *connect*, *mainWindow*, *statusValue*, *statusPoint*, *timerStatus*, **args*, ***kwargs*)

Bases: `object`

This class handles the functionality for managing buttons, checkboxes, and graphs for histograms related to the Start-Stop measurements. It controls which channels are active, manages the display of histograms, and records the data from the Tempico device based on the user's interactions. It also handles the zooming feature for creating bars according to the time range received from the device.

Parameters

- **parent** – The parent QWindow for the logic.
- **disconnect** – The QPushButton used to disconnect.
- **device** – The Tempico device class that handles the measurements.
- **check4** (*check1, check2, check3,*) – QCheckBoxes that control the visibility of each histogram.
- **stopbutton** (*startbutton,*) – QPushButtons to start and stop the measurement process.
- **save_graph_1** (*savebutton,*) – QPushButtons to save data and save the graph.
- **clear_channel_D** (*clear_channel_A, clear_channel_B, clear_channel_C,*) – QPushButtons to clear data for each respective channel.
- **connect** – QPushButton to connect to the Tempico device.
- **mainWindow** – The main window (QWindow) for the GUI.
- **statusPoint** (*statusValue,*) – QLabel widgets for displaying status information (e.g., values and points).

autoRangeA()

Automatically adjusts the zoom level of channel A to the maximum value in the data.

The function sets a sentinel for zoom changes, checks if there are any data points in channel A, and if so, finds the maximum value. It then calls the changeZoomMax function to adjust the zoom accordingly.

Returns
None

autoRangeB()

Automatically adjusts the zoom level of channel B to the maximum value in the data.

The function sets a sentinel for zoom changes, checks if there are any data points in channel B, and if so, finds the maximum value. It then calls the changeZoomMax function to adjust the zoom accordingly.

Returns
None

`autoRangeC()`

Automatically adjusts the zoom level of channel C to the maximum value in the data.

The function sets a sentinel for zoom changes, checks if there are any data points in channel C, and if so, finds the maximum value. It then calls the changeZoomMax function to adjust the zoom accordingly.

Returns
None

`autoRangeD()`

Automatically adjusts the zoom level of channel D to the maximum value in the data.

The function sets a sentinel for zoom changes, checks if there are any data points in channel D, and if so, finds the maximum value. It then calls the changeZoomMax function to adjust the zoom accordingly.

Returns
None

`changeColorThread(color)`

`changeStatusColor(color)`

Changes the color of the status point in the status bar based on the input value.

This method updates the status point's color by drawing a filled circle with the specified color. The mapping of numerical values to colors is as follows: - 0: Gray - 1: Green - 2: Yellow - 3: Orange

Parameters
color – A numerical value representing the desired color (int). 0 for gray, 1 for green, 2 for yellow, 3 for orange.

Returns
None

`changeStatusThread(newText)`

Updates the text displayed in the status bar.

This method sets the status value text to the specified new text, reflecting the current status or information relevant to the user.

Parameters
newText – The text to display in the status bar (str).

Returns
None

`changeZoomMax(newMaxValue, channel)`

Automatically adjusts the zoom level of the plot to the maximum value found.

The function sets the x-axis range of the specified channel's plot to [0, newMaxValue]. It considers whether measurements have been taken and limits the zoom change based on the sentinel zoom change counters.

Parameters

- **newMaxValue** – The new maximum value for the zoom range (float).
- **channel** – The channel to which the zoom change applies (str). Expected values are 'A', 'B', 'C', or 'D'.

Returns
None

`clear_a()`

Clears the lists containing measured data for channel A by resetting them to empty lists.

Returns
None

`clear_b()`

Clears the lists containing measured data for channel B by resetting them to empty lists.

Returns
None

`clear_c()`

Clears the lists containing measured data for channel C by resetting them to empty lists.

Returns
None

`clear_d()`

Clears the lists containing measured data for channel D by resetting them to empty lists.

Returns
None

`createDialog()`

Creates a dialog box to notify the user when the connection with the device has been lost.

This method displays a critical error message box indicating that the connection with the device has been lost. It also stops any ongoing graphic updates, hides relevant graphics, and disables the disconnect button while enabling the connect button.

Returns
None

`create_graphs()`

Creates histograms and plots for the selected channels (A, B, C, D) and arranges them on the GUI.

This function initializes the graphical components and assigns the appropriate properties to the plots based on the selected channels. It also manages the layout of the graphs, resizing them according to the number of selected channels (1, 2, 3, or 4). After setting up the graphical components, the function starts a worker thread that handles the measurement process in parallel with the GUI thread, ensuring the UI remains responsive.

Returns
None

`dialogChangeMode(`*channel*`)`

Opens a dialog box to ask the user if they want to change the mode for the selected channel.

The dialog informs the user that the collected data for the specified channel falls mostly outside the reliable range of mode 1 and prompts them to switch to mode 2. If the user confirms, the function updates the mode and processes the data accordingly.

Parameters
`channel` – The channel for which to change the mode (str). Expected values are 'channel A', 'channel B', 'channel C', or 'channel D'.

Returns
None

`disconnectedDevice()`

Disables device interaction controls when the device is disconnected.

This function updates the GUI to reflect that the device is no longer connected by disabling the disconnect and start buttons and enabling the connect button.

Returns
None

`hide_graphic2()`

Disables the start and stop buttons in the graphical interface.

This function is typically used to prevent user interaction during certain operations.

Returns
None

`save_graphic()`

Saves the current data according to the selected format specified in the dialog box.

The function starts by retrieving the default histogram name and the current date, which is formatted to create a unique filename. It then initializes lists to hold filenames, data, settings, and column names for the saved files.

A dialog box is displayed for the user to select the desired file format (txt, csv, or dat) for saving the data. Once the user accepts the selection, the function checks if the selected format has already been saved before.

If the selected format has not been saved, it collects data from the various channels (A, B, C, D) if their corresponding sentinel variables indicate they should be saved. It retrieves the average cycles, mode, number of stops, stop edge, and stop mask for each channel to include in the settings. The function then attempts to save the collected data in the specified format.

If the save operation is successful, a message box is displayed, confirming the successful save and showing the folder path and file names. If an error occurs during the save process, an error message box is shown.

If the selected format has already been saved, a message box is displayed with the previous save information.

Returns
None

`save_plots()`

Saves the current plots in the selected image format.

This method opens a dialog for the user to select an image format (PNG, TIFF, or JPG) and saves the plots for channels A, B, C, and D if their respective flags are set to True. The plots are saved with a timestamp in the specified format in the default folder path.

Returns
None

`show_graphic(`*device_new*`)`

Sets the device and its channels, enabling the start button for measurement.

Parameters
device_new – The new device to be set (DeviceType).

Returns
None

startTimerConnection()

Starts the timer that periodically checks the device connection.

The timer runs every 500 milliseconds to monitor connectivity. This is typically called after a measurement finishes or when the application is idle.

Returns
None

start_graphic()

Starts the graphical representation of the measurement based on the selected channels.

If no channels are selected, it prompts the user to select at least one channel before proceeding. It also disables certain buttons to prevent interaction during the measurement process.

Returns
None

stopTimerConnection()

Stops the timer responsible for checking the device connection.

This function is typically called when a measurement begins to avoid interference during the acquisition process.

Returns
None

stop_graphic()

Enables the functions to initiate a new measurement while disabling the buttons to clear graphs. It also updates the status bar text and color.

Returns
None

threadComplete()

Handles the completion of the measurement thread by enabling the ability to switch between windows. It also calls the stop_graphic() function to update the UI accordingly.

Returns
None

threadRunning(*status*)

Updates the thread creation status.

This method sets the threadCreatedSentinel attribute to indicate whether a thread has been created based on the provided status value.

Parameters
status – The status indicating whether a thread has been created (int). Use 0 to indicate that the thread has been created, and 1 to indicate that it has not.

Returns
None

updateDataPure(*value*, *channel*)

Updates the raw data values for the specified channel.

This method appends the new raw value (in picoseconds) to the list of data for the specified channel (A, B, C, or D).

Parameters

- **value** – The raw data value to append to the data list (float or int). This value is in picoseconds.
- **channel** – The channel identifier for which the value is to be updated (str). Accepted values are “A”, “B”, “C”, and “D”.

Returns
None

updateSignal(*value*, *channel*)

Updates the normalized data values for the specified channel and refreshes the corresponding histogram.

This method appends the new value to the list of normalized data for the specified channel (A, B, C, or D) and calls the update_histogram method to refresh the histogram with the updated data.

Parameters

- **value** – The normalized value to append to the data list (float or int).
- **channel** – The channel identifier for which the value is to be updated (str). Accepted values are “A”, “B”, “C”, and “D”.

Returns
None

update_histogram(*data*, *curve*, *indexChannel*)

Calculates and updates the histogram for the specified channel based on the measured data.

The function first checks the selected channel (A, B, C, or D) and retrieves the current view range from the corresponding view box. It then creates bins for the histogram based on this range, using a total of 61 bins.

Finally, it computes the histogram of the provided data and updates the specified curve in the plot with the histogram values.

Parameters

- **data** – The measured data used to calculate the histogram (list).
- **curve** – The curve item from the plot in PyQt5 with PyQtGraph that corresponds to the channel (PlotCurveItem).
- **indexChannel** – The identifier of the channel for the histogram, which can be “A”, “B”, “C”, or “D” (str).

Returns
None

zoom_changedA()

Automatically calculates and updates the histogram for channel A when the user zooms in on the corresponding graph.

The function checks if the zoom was triggered by code or user action. If it was triggered by the user, it increments the sentinel for zoom changes. It then retrieves the current x-range from the view box for channel A and computes the bin width for the histogram.

Next, it creates 61 bins based on the x-range and calculates the histogram using the measured data. The histogram data is then used to update the curve item for channel A in the plot.

Param
None

Returns
None

zoom_changedB()

Automatically calculates and updates the histogram for channel B when the user zooms in on the corresponding graph.

The function checks if the zoom was triggered by code or user action. If it was triggered by the user, it increments the sentinel for zoom changes. It then retrieves the current x-range from the view box for channel B and computes the bin width for the histogram.

Next, it creates 61 bins based on the x-range and calculates the histogram using the measured data. The histogram data is then used to update the curve item for channel B in the plot.

Param
None

Returns
None

zoom_changedC()

Automatically calculates and updates the histogram for channel C when the user zooms in on the corresponding graph.

The function checks if the zoom was triggered by code or user action. If it was triggered by the user, it increments the sentinel for zoom changes. It then retrieves the current x-range from the view box for channel C and computes the bin width for the histogram.

Next, it creates 61 bins based on the x-range and calculates the histogram using the measured data. The histogram data is then used to update the curve item for channel C in the plot.

Param
None

Returns
None

zoom_changedD()

Automatically calculates and updates the histogram for channel D when the user zooms in on the corresponding graph.

The function checks if the zoom was triggered by code or user action. If it was triggered by the user, it increments the sentinel for zoom changes. It then retrieves the current x-range from the view box for channel D and computes the bin width for the histogram.

Next, it creates 61 bins based on the x-range and calculates the histogram using the measured data. The histogram data is then used to update the curve item for channel D in the plot.

Param
None

Returns
None

class StartStopHist.**WorkerThreadStartStopHistogram**(*parent*, *device*, *sentinelSaveA*, *sentinelSaveB*, *sentinelSaveC*, *sentinelSaveD*)

Bases: QThread

This class represents a worker thread that processes Start-Stop measurements in a separate thread to avoid blocking the main GUI thread. It handles data collection from the Tempico device, processes the data, and emits signals to update the GUI with the results.

Parameters

- **parent** – The parent QWindow of the thread.

- **device** – The Tempico device class that handles the measurements.
- **sentinelSaveD** (*sentinelSaveA*, *sentinelSaveB*, *sentinelSaveC*,) – Boolean flags that determine if data for each channel should be saved.

addChannelWarning(*channel*)

Adds the specified channel to the list of channels that need to change mode.

Parameters
channel – The channel to be added to the warning list (str).

Returns
None

changeMaxValue(*channel*, *valueMax*)

Changes the maximum value for the specified channel.

Parameters

- **channel** – The channel for which the maximum value is to be changed (str).
- **valueMax** – The new maximum value to be set (float).

Returns
None

colorValue

dataPureSignal

dataSignal

dialogInit

dialogIsOpen()

Sets the 'openDialog' sentinel to False, indicating that no dialog is open.

Returns
None

dialogSignal

getNewData(*channel*, *channelIndex*, *stopNumber*)

Performs multiple measurements on the device, averages the results, and emits the corresponding data. If no data is available, it emits None. Increments the total number of measurements taken and activates sentinels to indicate whether the device is actively collecting data.

Parameters

- **channel** – The device channel used for measurements (Channel).
- **channelIndex** – The index of the channel, which can be 'A', 'B', 'C', or 'D' (str).
- **stopNumber** – The stop number to obtain the corresponding measurement (int).

Returns
The average measurement in milliseconds if data is available; otherwise, returns None (float or None).

newMaxValueSignal

run()

Executes the thread's main loop to update the graph based on measurements.

The function emits a signal indicating that the thread has been created. It then enters a loop that continues as long as the thread is running. Inside the loop, it calls the update method to refresh the graph and pauses for 0.5 seconds before the next iteration.

Returns
None

staticMetaObject = <PySide2.QtCore.QMetaObject object>

stop()

Stops the measurement process and closes the thread.

This method emits a signal to indicate that the thread has been stopped and sets the running flag to False.

Returns
None

stringValue

threadCreated

update()

Performs measurements and updates the graph with the new data.

The function verifies the device connection and retrieves measurements for each channel (A, B, C, D). It checks if the majority of measurements are out of range when in mode 1. If so, it emits a signal to open a dialog box asking the user if they want to change the mode. If there are no measurements, the function prioritizes the alert to change mode. Otherwise, it emits the measurement value to update the graph.

Returns
None

CHAPTER

THREE

ABOUTDIALOG MODULE

class aboutDialog.**Ui_AboutDialog**

Bases: object

acceptButton()

open_link(*url*)

Opens a specified URL in the default web browser.

This function is triggered when a link is clicked and opens the provided URL using the system's default web browser.

Parameters
link (*str*) – The URL to be opened.

Returns
None

setupUi(*AboutDialog*)

CHAPTER

FOUR

COUNTSESTIMATEDGRAPHICS MODULE

class CountsEstimatedGraphics.**CountEstimatedLogic**(*channelACheckBox: QCheckBox*, *channelBCheckBox: QCheckBox*, *channelCCheckBox: QCheckBox*, *channelDCheckBox: QCheckBox*, *startButton: QPushButton*, *stopButton: QPushButton*, *mergeRadio: QRadioButton*, *separateGraphics: QRadioButton*, *deatachedGraphics: QRadioButton*, *timeRangeComboBox: QComboBox*, *clearButtonChannelA: QPushButton*, *clearButtonChannelB: QPushButton*, *clearButtonChannelC: QPushButton*, *clearButtonChannelD: QPushButton*, *saveDataButton: QPushButton*, *savePlotButton: QPushButton*, *countChannelAValue: QLabel*, *countChannelBValue: QLabel*, *countChannelCValue: QLabel*, *countChannelDValue: QLabel*, *countChannelAUncertainty: QLabel*, *countChannelBUncertainty: QLabel*, *countChannelCUncertainty: QLabel*, *countChannelDUncertainty: QLabel*, *tableCounts: QTableWidget*, *graphicsFrame: QFrame*, *channelAFrameLabel: QFrame*, *channelBFrameLabel: QFrame*, *channelCFrameLabel: QFrame*, *channelDFrameLabel: QFrame*, *statusLabel: QLabel*, *pointStatusLabel: QLabel*, *deatachedCheckBox: QCheckBox*, *detachedLabelCheckBox: QCheckBox*, *helpButton: QPushButton*, *device: TempicoDevice*, *parent*, *timerConnection*)

Bases: `object`

Class responsible for managing the logic and graphical representation of the Count Estimation window.

This class handles user interactions and graphical updates related to photon/event count estimations across multiple channels (A, B, C, D). It coordinates the initialization of graphical elements, manages real-time measurements, and provides options for data saving, graphical visualization modes, and uncertainty tracking per channel. The main responsibilities include: - Initializing and updating graphical frames for each measurement channel. - Starting and stopping the count measurements. - Handling user interactions with checkboxes, radio buttons, and combo boxes. - Displaying and updating count values and their uncertainties. - Managing separate, merged, or

detached graphics. - Saving count data and graphical plots. - Monitoring the device connection and measurement state.

Parameters

- **channelACheckBox** – Checkbox for enabling/disabling measurements on channel A (QCheckBox).
- **channelBCheckBox** – Checkbox for enabling/disabling measurements on channel B (QCheckBox).
- **channelCCheckBox** – Checkbox for enabling/disabling measurements on channel C (QCheckBox).
- **channelDCheckBox** – Checkbox for enabling/disabling measurements on channel D (QCheckBox).
- **startButton** – Button to start the count measurement (QPushButton).
- **stopButton** – Button to stop the count measurement (QPushButton).
- **mergeRadio** – Radio button to merge all channel plots into one graph (QRadioButton).
- **separateGraphics** – Radio button to display plots separately per channel (QRadioButton).
- **deatachedGraphics** – Radio button to display detached external graphics (QRadioButton).
- **timeRangeComboBox** – Combo box for selecting the time range for plot visualization (QComboBox).
- **clearButtonChannelA** – Button to clear the graph and data of channel A (QPushButton).
- **clearButtonChannelB** – Button to clear the graph and data of channel B (QPushButton).
- **clearButtonChannelC** – Button to clear the graph and data of channel C (QPushButton).
- **clearButtonChannelD** – Button to clear the graph and data of channel D (QPushButton).
- **saveDataButton** – Button to save count data for all channels (QPushButton).
- **savePlotButton** – Button to save the plot images (QPushButton).
- **countChannelAValue** – Label displaying current count value on channel A (QLabel).
- **countChannelBValue** – Label displaying current count value on channel B (QLabel).
- **countChannelCValue** – Label displaying current count value on channel C (QLabel).
- **countChannelDValue** – Label displaying current count value on channel D (QLabel).
- **countChannelAUncertainty** – Label showing uncertainty of channel A (QLabel).
- **countChannelBUncertainty** – Label showing uncertainty of channel B (QLabel).
- **countChannelCUncertainty** – Label showing uncertainty of channel C (QLabel).
- **countChannelDUncertainty** – Label showing uncertainty of channel D (QLabel).
- **tableCounts** – Table displaying count and uncertainty data per measurement cycle (QTableWidget).
- **graphicsFrame** – Frame container for holding the graphical layouts (QFrame).
- **channelAFrameLabel** – Frame label associated with channel A graphics (QFrame).
- **channelBFrameLabel** – Frame label associated with channel B graphics (QFrame).
- **channelCFrameLabel** – Frame label associated with channel C graphics (QFrame).

- **channelDFrameLabel** – Frame label associated with channel D graphics (QFrame).
- **statusLabel** – Label showing the system status (QLabel).
- **pointStatusLabel** – Label showing the status of the current measurement point (QLabel).
- **deatachedCheckBox** – Checkbox to toggle between embedded and detached graphics layout (QCheckBox).
- **detachedLabelCheckBox** – Checkbox to toggle detached labels in the plot layout (QCheckBox).
- **helpButton** – Button that displays a help/information dialog (QPushButton).
- **device** – The connected measurement device.
- **parent** – The parent window containing the UI (usually a QMainWindow).
- **timerConnection** – Timer responsible for checking device connection status (QTimer).

Returns
None

captureFinalDate(*date*)

Stores the final date received from a signal.

Parameters
date – QDateTime - The end date of the measurement range.

Returns
None

captureInitalDate(*date*)

Stores the initial date received from a signal.

Parameters
date – QDateTime - The start date of the measurement range.

Returns
None

captureMeasurement(*secondsTime*, *dateTime*, *channelAValue*, *channelAUncertainty*, *channelBValue*, *channelBUncertainty*, *channelCValue*, *channelCUncertainty*, *channelDValue*, *channelDUncertainty*)

Captures and processes a new set of measurement values emitted via signal, updating the UI and internal data structures.

This function is connected to a QSignal from the background worker thread and is responsible for handling a single measurement cycle. It validates the presence of data, provides real-time user feedback when no measurements are detected for specific channels, updates plots with valid values, populates the measurement table, and refreshes visual labels.

This function performs the following actions: - Checks for zero values in any channel and informs the user if one or more channels are not receiving valid measurements. - Updates the status bar and color indicator depending on whether all channels are acquiring data. - Appends valid count values and timestamps to the corresponding lists for each channel. - Updates the graphical curves for each channel if valid data is present. - Converts zero and -1 values into status messages like “Low Counts” or “Not Selected” for display purposes. - Adds the latest measurement row to both the main table (*tableCounts*) and its clone (*cloneTable*). - Updates the live count and uncertainty labels for any actively selected channel. - Refreshes the visible X-axis range of the graphs using *updateGraphic()*.

Parameters

- **secondsTime** – Timestamp in seconds since measurement start (float).
- **dateTime** – Human-readable datetime string for the measurement (str).
- **channelAValue** – Count value for channel A (float or int).
- **channelAUncertainty** – Uncertainty value for channel A (float).
- **channelBValue** – Count value for channel B (float or int).
- **channelBUncertainty** – Uncertainty value for channel B (float).
- **channelCValue** – Count value for channel C (float or int).
- **channelCUncertainty** – Uncertainty value for channel C (float).
- **channelDValue** – Count value for channel D (float or int).
- **channelDUncertainty** – Uncertainty value for channel D (float).

Returns
None

changeStatusColor(*color*)

Changes the color of the point in the status bar. Each color is assigned a specific numeric value.

Parameters
color – The numeric value corresponding to the desired color (int). - 0: Gray - 1: Green - 2: Yellow - 3: Orange

Returns
None

changeStatusLabel(*textValue*)

Updates the status label with a new message.

This function sets the text of the *statusLabel* widget to the given string. It is used throughout the application to inform the user of the current measurement status, errors, warnings, or progress.

Parameters
textValue – str - The message to display in the status label.

Returns
None

checkBoxListenerChannels()

Updates the visibility of graphical elements and table columns based on the state of the channel checkboxes.

This function performs the following actions: - Checks the state of each channel's checkbox (A, B, C, D). - Shows or hides the corresponding frame label for each channel depending on whether the checkbox is checked. - Calls *hideColumns()* to hide the table columns associated with unchecked channels. - Calls *updateGraphicsLayout()* to refresh the displayed plots based on the selected channels.

Returns
None

clearChannelA()

Clears all stored measurement data for channel A and updates the corresponding plot.

This function is called when the user clicks the Clear button for channel A. It resets the timestamp lists and count values associated with channel A, and refreshes the plot to reflect the cleared state.

This function performs the following actions: - Empties the list of raw timestamps and formatted date timestamps for channel A. - Clears the list of count values associated with channel A. - Updates the channel A curve to remove all plotted data.

Returns
None

`clearChannelB()`

Clears all stored measurement data for channel B and updates the corresponding plot.

This function is called when the user clicks the Clear button for channel B. It resets the timestamp lists and count values associated with channel B, and refreshes the plot to reflect the cleared state.

This function performs the following actions: - Empties the list of raw timestamps and formatted date timestamps for channel B. - Clears the list of count values associated with channel B. - Updates the channel B curve to remove all plotted data.

Returns
None

`clearChannelC()`

Clears all stored measurement data for channel C and updates the corresponding plot.

This function is called when the user clicks the Clear button for channel C. It resets the timestamp lists and count values associated with channel C, and refreshes the plot to reflect the cleared state.

This function performs the following actions: - Empties the list of raw timestamps and formatted date timestamps for channel C. - Clears the list of count values associated with channel C. - Updates the channel C curve to remove all plotted data.

Returns
None

`clearChannelD()`

Clears all stored measurement data for channel D and updates the corresponding plot.

This function is called when the user clicks the Clear button for channel D. It resets the timestamp lists and count values associated with channel D, and refreshes the plot to reflect the cleared state.

This function performs the following actions: - Empties the list of raw timestamps and formatted date timestamps for channel D. - Clears the list of count values associated with channel D. - Updates the channel D curve to remove all plotted data.

Returns
None

`closeDialogChannels(`*channel*`)`

Handles cleanup and UI updates when a detached graphics dialog is closed.

This function is triggered when a user manually closes a detached dialog window corresponding to a specific channel. It clears the internal reference to the dialog and unchecks the associated channel checkbox to reflect that the channel is no longer active in the detached view.

This function performs the following actions: - Sets the corresponding *dialogXCreated* variable to *None*, where X is the channel (A–D). - Unchecks the checkbox for the specified channel to visually indicate it is no longer selected.

Parameters
channel – The identifier of the channel whose dialog was closed (‘A’, ‘B’, ‘C’, or ‘D’).

Returns
None

`closeLabelDialog()`

Resets the dialog state and unchecks the measurement label checkbox if the dialog was closed while still marked as active.

Returns
None

`closeTableDialog`()

Handles cleanup when the detached table dialog is closed.

This function is triggered when the detached table dialog (*dialogTableOpen*) is closed by the user. It performs the following actions to maintain consistency in the UI state:

- Sets the internal dialog reference *dialogTableOpen* to *None*.
- Unchecks the *deatachedCheckBox* to reflect that the detached view is no longer active.

This ensures that the application's interface remains synchronized and avoids inconsistencies when reopening or reattaching the measurement table view.

Returns
None

`connectedDevice`(*device*)

Handles the UI and internal state updates when the device is successfully connected.

This function performs the following actions: - Enables the disconnect button and disables the connect button in the main window. - Updates the internal sentinel to indicate that the device is connected and measurements are allowed. - Stores the reference to the connected device. - Enables the start button to allow measurements to begin. - (Optional) Starts a timer for updating measurement status (currently commented out).

Parameters
`device` – The connected device instance.

Returns
None

`constructAllGraphics`()

Constructs and initializes the unified graphical layout for all measurement channels.

This function calls *factoryGraphsAllChannels* to generate a single plot widget that contains curves for channels A, B, C, and D. Each curve is customized by color and visibility based on whether its corresponding checkbox is enabled. The resulting components are stored as instance variables for centralized access and updates.

This function performs the following actions: - Calls *factoryGraphsAllChannels* to create the shared plot and individual channel curves. - Stores the plot widget, plot item, and all channel-specific curves as instance variables.

Returns
None

`constructGraphicA`()

Constructs and initializes the graphical components for channel A.

This function uses the *factoryGraphChannels* method to generate the graphical layout for channel A, including the main plot widget, the plot area, and the curve used to display count data. The resulting components are stored as instance variables for later access and updates during measurement.

This function performs the following actions: - Calls *factoryGraphChannels* with channel 'A' to generate the graph. - Stores the returned plot widget, plot item, and curve for channel A.

Returns
None

constructGraphicB()

Constructs and initializes the graphical components for channel B.

This function uses the *factoryGraphChannels* method to generate the graphical layout for channel B, including the main plot widget, the plot area, and the curve used to display count data. The resulting components are stored as instance variables for later access and updates during measurement.

This function performs the following actions: - Calls *factoryGraphChannels* with channel 'B' to generate the graph. - Stores the returned plot widget, plot item, and curve for channel B.

Returns
None

constructGraphicC()

Constructs and initializes the graphical components for channel C.

This function uses the *factoryGraphChannels* method to generate the graphical layout for channel C, including the main plot widget, the plot area, and the curve used to display count data. The resulting components are stored as instance variables for later access and updates during measurement.

This function performs the following actions: - Calls *factoryGraphChannels* with channel 'C' to generate the graph. - Stores the returned plot widget, plot item, and curve for channel C.

Returns
None

constructGraphicD()

Constructs and initializes the graphical components for channel D.

This function uses the *factoryGraphChannels* method to generate the graphical layout for channel D, including the main plot widget, the plot area, and the curve used to display count data. The resulting components are stored as instance variables for later access and updates during measurement.

This function performs the following actions: - Calls *factoryGraphChannels* with channel 'D' to generate the graph. - Stores the returned plot widget, plot item, and curve for channel D.

Returns
None

createCloneTable()

Creates and configures a cloned table instance for storing a copy of the measurement data.

This function initializes a separate *QTableWidget* that mirrors the structure of the main count table. It includes predefined column headers for the date and channels A, B, C, and D. The cloned table is set to stretch its columns to fit the available space and is made read-only to prevent user edits.

This function performs the following actions: - Creates a new *QTableWidget* instance with 5 columns. - Sets the horizontal headers to ['Date', 'A', 'B', 'C', 'D']. - Enables automatic stretching of column widths. - Disables editing to preserve the integrity of the data.

Returns
None

createDialogFactory(*channel*)

Creates and returns a non-modal dialog window for displaying detached graphics for the specified channel.

This factory function is used when the user selects the "detached graphics" mode. It generates a dedicated *QDialog* for a given channel (A, B, C, or D), configures its appearance and behavior, and positions it to avoid overlap with other dialogs. The dialog is linked to a callback that ensures proper cleanup when it is closed.

This function performs the following actions: - Instantiates a *QDialog* as a child of the main window. - Sets the dialog title and default size. - Connects the *finished* signal to a cleanup method for the channel. - Assigns a fixed screen position for each channel to avoid visual overlap.

Parameters
`channel` – A string indicating the target channel ('A', 'B', 'C', or 'D').

Returns
A configured, ready-to-display *QDialog* for the given channel.

`createRow`(*leftLabel*, *rightLabel*)

Creates a horizontal row frame containing two labels.

This function constructs a *QFrame* that represents a single row in a dialog or panel. It arranges two given *QLabel* widgets — one aligned to the left and the other to the right — using a *QHBoxLayout*. The labels are styled with a consistent font size and the vertical margins are removed to minimize vertical spacing between rows.

This method is used to visually structure measurement data (such as estimated counts and uncertainties) in a clean, compact format.

Parameters

- `leftLabel` – QLabel - The label to place on the left side of the row.
- `rightLabel` – QLabel - The label to place on the right side of the row.

Returns
QFrame - A frame containing the two labels arranged horizontally.

`deatachedTable`()

Manages the visibility of the estimated counts table in a detached dialog window.

This function is triggered when the *deatachedCheckBox* is toggled. If the checkbox is checked, it opens a new non-modal *QDialog* containing the cloned measurement table (*cloneTable*). If the checkbox is unchecked, it closes the dialog and resets the internal reference to *None*.

This function performs the following actions: - If the detached table mode is enabled:

- Creates and configures a new *QDialog* with the cloned table.
- Displays the dialog to the user.
- Connects the dialog's close event to a cleanup function.
- **If the detached table mode is disabled:**
  - Closes the existing dialog if it is open.
  - Clears the internal dialog reference.

Returns
None

`detachedLabels`()

Displays or hides a dialog showing current measurement values for selected channels.

When the 'Detached current measurement' checkbox is checked, this function creates and displays a non-modal QDialog containing the current estimated count rates and their uncertainties for each selected channel (A–D). Each visible row in the dialog corresponds to a channel, displaying the estimated counts (in cps) and the associated uncertainty. If a channel is not selected, its row remains hidden.

The dialog layout is organized using a sunken panel-style QFrame with a vertical layout. A bold header row indicates the meaning of each column. This interface allows users to quickly inspect the most recent count values without modifying the ongoing measurement.

If the checkbox is unchecked, any open measurement dialog is closed automatically.

Returns
None

`disconnectedDevice()`

Handles the UI and internal state updates when the device is disconnected.

This function performs the following actions: - Disables the disconnect button and enables the connect button in the main window. - Disables the start button to prevent new measurements from being initiated. - (Optional) May start a timer for monitoring status or reconnection attempts (currently commented out).

Returns
None

`eliminateCheckBoxChannels(`*channelList*`)`

Handles the removal of channels with insufficient measurement signals and prompts the user to continue or stop.

This function is triggered when one or more selected channels fail to meet the minimum pulse requirements for reliable count estimation (e.g., fewer than 500 pulses per second). It presents a dialog informing the user about the affected channels and asks whether to continue the measurement with the remaining valid channels or to stop entirely.

This function performs the following actions: - Builds a human-readable list of the channels that failed to provide sufficient data. - Displays a warning dialog with the option to continue or cancel the measurement. - If the user chooses to continue:

- Disables the affected channels in the device.
- Unchecks and disables the associated checkboxes.
- Disables the "Clear" button for each affected channel.
- Marks the *measurementChannelX* sentinel as *False* for each affected channel.
- **If the user chooses not to continue:**
  - Stops the measurement thread.
- Signals the worker thread to proceed based on the user's decision via *continueEvent*.

Parameters
`channelList` – A list of channel identifiers (e.g., ['A', 'C']) that failed to meet the minimum measurement conditions.

Returns
None

`enableButtons()`

Enables or disables the clear buttons for each channel based on the current selection state.

This function checks which channels (A–D) have been marked as selected for measurement and enables their corresponding "Clear" buttons. If a channel is not selected, its clear button remains disabled. Additionally, it disables the disconnect button to prevent accidental disconnection during an active measurement setup.

This function performs the following actions: - Enables the "Clear" button for each selected channel. - Disables the "Clear" button for any unselected channel. - Disables the disconnect button in the main window.

Returns
None

`factoryGraphChannels`(*channel*)

Creates and returns a configured plot widget for the specified measurement channel.

This factory function generates a graphical layout for visualizing count data from one of the four available channels (A, B, C, or D). The plot is customized with a unique color per channel, grid display, axis labeling, and a legend. It returns the PyQtGraph components necessary for real-time plotting.

This function performs the following actions: - Selects a predefined color for the specified channel. - Initializes a *GraphicsLayoutWidget* with a white background. - Adds a plot to the widget with grid lines, axis labels, and legend. - Creates a stylized curve object for plotting count data using markers and colored lines. - Returns the components needed to integrate the plot into the GUI and update it dynamically.

Parameters
`channel` – A single-character string indicating the channel ('A', 'B', 'C', or 'D').

Returns
A tuple containing: - *GraphicsLayoutWidget*: the plot container. - *PlotItem*: the configured plot. - *PlotDataItem*: the curve used for plotting the counts.

`factoryGraphsAllChannels`()

Creates and returns a single plot widget containing curves for all selected measurement channels.

This factory function generates a unified graphical layout that displays the estimated counts for channels A, B, C, and D in a single plot. Each channel is represented with a distinct color and labeled curve, depending on whether its checkbox is enabled. This approach consolidates the visualization into one graph rather than creating separate ones per channel.

This function performs the following actions: - Defines a unique color for each channel (A, B, C, D). - Initializes a *GraphicsLayoutWidget* with a white background and grid lines. - Configures a single plot with appropriate axis labels and a legend. - Creates curve objects for each channel using stylized markers and pens. - Only assigns a name to a curve if its corresponding channel checkbox is checked. - Returns the complete widget and all channel curves for display and real-time updates.

Returns
A tuple containing: - *GraphicsLayoutWidget*: the shared plot container for all channels. - *PlotItem*: the unified plot area. - *PlotDataItem*: curve for channel A. - *PlotDataItem*: curve for channel B. - *PlotDataItem*: curve for channel C. - *PlotDataItem*: curve for channel D.

`finishedThread`()

Handles the post-execution cleanup and UI updates when the measurement thread finishes.

This function is automatically triggered when the background measurement thread completes. It resets internal sentinels, updates the UI to reflect the end of the measurement, enables or disables appropriate buttons depending on whether data was captured, and invokes the standard stop measurement routine.

This function performs the following actions: - Resets all *selectChannelX* sentinels (A–D) to *False*. - Enables the Start button and disables the Stop button. - If any channel has collected data, enables the save buttons for data and plots. - Disables all "Clear" buttons for individual channels. - Updates the status label and color to indicate that no measurement is running. - Calls *stopMeasure()* to finalize the stop process and restore the interface to its initial state.

Returns
None

`getChannelsMeasure()`

Determines which measurement channels are selected and updates internal sentinels accordingly.

This function evaluates the state of each channel's checkbox (A, B, C, D) to establish which channels will participate in the measurement. It sets the corresponding selection and measurement sentinels to *True* or *False* based on the checkboxes. Additionally, it disables the checkbox for any unselected channel to prevent user changes during the active measurement session.

This function performs the following actions: - Assumes all channels are selected by default. - Checks the state of each channel checkbox and updates:

- *selectChannelX*: sentinel to include channel X in the measurement loop.
- *measurementChannelX*: sentinel to track measurement activity per channel.
- Disables the checkbox for any channel that was not selected to lock in the configuration.

Returns
None

`helpButtonDialog()`

Displays an informational dialog with instructions for using the Count Estimation window.

This function shows a modal QMessageBox with an information icon to guide the user on how to correctly connect the signal sources for pulse estimation. It clarifies that a periodic signal should be connected to the Start input, and the sources to be measured should be connected to the Stop channels.

Returns
None

`hideColumns()`

Updates the visibility of the table columns based on the selected measurement channels.

This function ensures that only the columns corresponding to active channels (A–D) are visible in both the main and cloned data tables. It first shows all channel columns, then hides those whose associated checkboxes are unchecked.

This function performs the following actions: - Ensures all channel columns (1–4) are visible initially in both *tableCounts* and *cloneTable*. - Checks the state of each channel's checkbox. - Hides the column in both tables for any channel that is not currently selected.

Returns
None

`lostConnection()`

Displays a critical error message indicating that the device connection has been lost.

This function sets the sentinel flag *disconnectedMeasurement* to True and shows a modal QMessageBox with a critical icon to inform the user that the connection to the device has been lost. The dialog must be acknowledged before continuing.

Returns
None

`noChannelsSelected()`

Displays a warning dialog indicating that no channels have been selected for measurement.

This function is called when the user attempts to start a measurement without selecting any channels. It generates a modal warning dialog to inform the user that at least one channel must be selected to proceed.

This function performs the following actions: - Opens a *QMessageBox* warning with a title and informative message. - Prevents the measurement process from starting until the issue is resolved.

Returns
None

`noMeasurementsFounded()`

Displays a warning dialog and disables channel interactions when no measurements are detected.

This function is triggered when the system fails to obtain valid measurements from any of the selected channels. It informs the user of the issue, disables the "Clear" buttons for all channels, and resets the internal sentinels that indicate active measurement states.

This function performs the following actions: - Shows a *QMessageBox* warning explaining that no measurements were found and outlines the minimum requirements for detection (e.g., 500 pulses per second). - Disables the "Clear" buttons for channels A–D to prevent further user interaction. - Resets *measurementChannelX* sentinels (A–D) to *False*, marking all channels as inactive.

Returns
None

`resetSentinels()`

Resets internal channel selection sentinels and re-enables all channel checkboxes.

This function is typically called after a measurement is stopped. It clears the selection flags (*selectChannelX*) for all channels (A–D) and re-enables their corresponding checkboxes in the UI, allowing the user to configure new measurement selections.

This function performs the following actions: - Sets all *selectChannelX* sentinels to *False*, indicating no active selections. - Enables the checkboxes for channels A, B, C, and D to restore full configuration flexibility.

Returns
None

`resetValues()`

Resets all measurement-related data, graphics, tables, and save-state sentinels to their initial state.

This function is typically called before starting a new measurement session. It ensures that all previous data and UI elements are cleared to avoid inconsistencies or overlapping results. It resets internal storage lists, clears the curves on the plots, empties the result tables, and resets file save status indicators.

This function performs the following actions: - Clears all count values and timestamps for channels A–D. - Clears the corresponding date-formatted timestamp lists. - Resets the row count of the main and cloned data tables. - Clears all data from the plotted curves for each channel. - Resets internal sentinels that track whether the data has been saved in TXT, CSV, or DAT format.

Returns
None

`returnSettings()`

Restores the previously saved configuration settings to the Tempico device.

This function is typically called after a measurement session ends. It applies the saved settings—previously captured via *saveSettings()*—back to the Tempico device for all channels (A–D). This includes restoring the number of runs, number of stops, mode, averaging cycles, and stop mask values. If any of the required variables were not initialized (e.g., due to a missing *saveSettings()* call), the function catches the *NameError* and prints the exception.

This function performs the following actions: - Restores the number of acquisition runs for the entire device. - Restores the number of averaging cycles for each channel. - Restores the number of stop events for each

channel. - Restores the operation mode for each channel. - Restores the stop mask configuration for each channel. - Catches and prints *NameError* if any setting is undefined.

Returns
None

`rightAlignedItem`(*value*)

Creates a QTableWidgetItem with right-aligned and vertically centered text.

This function takes a value, converts it to a string, wraps it in a QTableWidgetItem, and sets the text alignment to align the content to the right and center it vertically. It is used to ensure consistent formatting of numerical or textual data in table cells.

Parameters
value (*Any*) – The value to be displayed in the table cell.

Returns
A QTableWidgetItem with the specified alignment.

Return type
QTableWidgetItem

`saveData`()

Saves the current data according to the selected format specified in the dialog box.

The function starts by retrieving the default histogram name and the current date, which is formatted to create a unique filename. It then initializes lists to hold filenames, data, settings, and column names for the saved files.

A dialog box is displayed for the user to select the desired file format (txt, csv, or dat) for saving the data. Once the user accepts the selection, the function checks if the selected format has already been saved before.

If the selected format has not been saved, it collects data from the various channels (A, B, C, D) if their corresponding sentinel variables indicate they should be saved. It retrieves the average cycles, mode, number of stops, stop edge, and stop mask for each channel to include in the settings. The function then attempts to save the collected data in the specified format.

If the save operation is successful, a message box is displayed, confirming the successful save and showing the folder path and file names. If an error occurs during the save process, an error message box is shown.

If the selected format has already been saved, a message box is displayed with the previous save information.

Returns
None

`savePlots`()

Saves the current plots in the selected image format.

This method opens a dialog for the user to select an image format (PNG, TIFF, or JPG) and saves the plots for channels A, B, C, and D if their respective flags are set to True. The plots are saved with a timestamp in the specified format in the default folder path.

Returns
None

`saveSettings`()

Saves the current configuration settings of the Tempico device for all channels.

This function queries the device and stores key configuration parameters for each channel (A–D), such as the number of stops, acquisition mode, averaging cycles, and stop mask. These values are cached in instance variables so they can be restored later (e.g., after a measurement session). If the device is unavailable or any read operation fails, the function safely ignores the error.

This function performs the following actions: - Retrieves and stores the global number of acquisition runs. - Retrieves and stores the number of stops per channel. - Retrieves and stores the operation mode per channel. - Retrieves and stores the number of averaging cycles per channel. - Retrieves and stores the stop mask per channel. - Ignores exceptions silently if the device is not ready.

Returns
None

`startMeasure()`

Starts the count measurement process by configuring the UI, preparing internal states, and launching the background worker thread.

This function is triggered when the user clicks the Start button. It verifies that at least one measurement channel (A–D) is selected. If so, it proceeds to initialize the measurement by disabling conflicting buttons and tabs, saving current settings, resetting values, and setting up the worker thread responsible for data acquisition. If no channels are selected, it displays a dialog warning the user.

This function performs the following actions: - Verifies that at least one channel checkbox is active. - Disables the Start button and enables the Stop button. - Disables data saving buttons and tab navigation to prevent user interference. - Saves current settings and stops the connection monitoring timer. - Resets measurement values and identifies the selected channels. - Updates the UI status bar to indicate active measurement. - Creates and configures a *WorkerThreadCountsEstimated* thread to perform background data acquisition. - Connects various thread signals to corresponding update methods in the class. - Starts the measurement thread. - If no channels are selected, displays a warning dialog requiring the user to activate at least one channel.

Returns
None

`startTimerConnection()`

Starts the connection monitoring timer to periodically check the device status.

This function activates the *timerConnection*, which verifies the connection with the Tempico device every 500 milliseconds. It is typically called after a measurement has ended or when the device is reconnected, ensuring continuous monitoring of the device's availability.

Returns
None

`stopMeasure()`

Stops the ongoing measurement process and restores the UI and device to their initial configuration.

This function is triggered when the user clicks the Stop button. It halts the background measurement thread, resets relevant internal flags (sentinels), re-enables UI tabs and controls, and restores the initial settings of the Tempico device if the disconnection did not occur. If the device was disconnected during measurement, it triggers the disconnection handling routine.

This function performs the following actions: - Re-enables the disabled UI tabs to allow user interaction post-measurement. - Calls *resetSentinels()* to clear internal flags related to active measurement channels. - Disables the Stop button and signals the worker thread to stop. - If the device is still connected:

- Restarts the connection-monitoring timer.
- Re-enables the disconnect button.
- **If the device was disconnected during measurement:**
  - Triggers the disconnection handling logic in the main window.
- Updates the main window status to indicate that no measurement is currently running.

- Calls *returnSettings()* to restore device parameters to their original configuration.

Returns
None

`stopTimerConnection()`

Stops the connection monitoring timer during an active measurement.

This function is called when a measurement begins to prevent interference from the periodic connection checks. It halts the *timerConnection*, which normally runs in the background to monitor the status of the Tempico device.

Returns
None

`updateGraphic()`

Updates the visible X-axis range of all channel plots based on the selected time window.

This function is triggered when the user changes the time range using the *timeRangeComboBox*. It determines the most recent timestamp in channel A and adjusts the X-axis range for all plots (A–D) accordingly, displaying only the data within the selected time interval.

This function performs the following actions: - Verifies that channel A has data; if not, it exits early. - Attempts to extract the number of seconds from the selected combo box option. - Calculates the minimum and maximum X-axis values based on the most recent timestamp. - Updates the X-axis range of all plots (A, B, C, and D) to reflect the selected time window.

Returns
None

`updateGraphicsLayout()`

Updates the graphical layout based on the selected visualization mode and active channel checkboxes.

This function dynamically reconfigures how the channel plots are displayed, depending on the selected radio button mode: separate graphics, merged graph, or detached dialogs. It ensures that the interface reflects the current user selections by showing or hiding the corresponding curves, windows, or layouts.

This function performs the following actions: - If **separate graphics** mode is selected:

- Closes any open detached dialogs.
- Clears the current layout inside the *graphicsFrame*.
- Creates individual graphs for each selected channel (A–D).
- **Arranges them responsively:**
  - 1 channel → full frame.
  - 2 channels → one above the other.
  - 3 channels → two top, one bottom.
  - 4 channels → 2×2 grid layout.
- **If merge graphics mode is selected:**
  - Closes detached dialogs if any exist.
  - Clears the layout in *graphicsFrame*.
  - Creates a single graph containing all curves.
  - Displays or hides each curve depending on its checkbox status.

- **If detached graphics mode is selected:**
  - Clears the main *graphicsFrame* layout.
  - For each selected channel (A–D), opens a dedicated dialog window.
  - Loads previous data into the newly created plots.
  - Closes dialog windows for unchecked channels.

Returns
None

`updateLabels`(*channel*, *value*, *uncertainty*)

Updates the UI labels displaying the latest measurement value and its uncertainty for the specified channel.

This function formats and sets the value and uncertainty labels for a given channel (A–D). If the value is a string (e.g., "Low Counts" or "Not Selected"), it is displayed as-is. If it is a numeric value, the function rounds it for clearer presentation before updating the corresponding labels.

This function performs the following actions: - Determines if the value is a string or numeric. - Rounds numeric values (value to 2 decimals, uncertainty to 5 decimals). - Updates the *QLabel* elements associated with the selected channel to reflect the latest measurement results.

Parameters

- **channel** – The target channel identifier ('A', 'B', 'C', or 'D').
- **value** – The most recent measured value (float, int, or str).
- **uncertainty** – The associated uncertainty of the measurement (float or str).

Returns
None

`class` CountsEstimatedGraphics.**WorkerThreadCountsEstimated**(*channelASentinel*, *channelBSentinel*, *channelCSentinel*, *channelDSentinel*, *device: TempicoDevice*)

Bases: `QThread`

This class represents a worker thread that processes Start-Stop measurements in a separate thread to avoid blocking the main GUI thread. It handles data acquisition from the Tempico device, determines the feasibility of performing measurements per channel, and continuously collects and emits measurement data while running.

The thread evaluates which channels meet the minimum requirements for valid measurements (e.g., at least 2 stops within a defined time window), disables those that do not, and allows the user to decide whether to proceed with the remaining ones.

Signals are extensively used to communicate with the GUI, updating labels, graphs, and table data in real time, or notifying when no valid measurements are available.

Additional attributes track the number of stop events per channel and enable synchronization with the main interface using a *threading.Event()* object.

Parameters

- **channelASentinel** – Boolean indicating if Channel A is initially selected for measurement.
- **channelBSentinel** – Boolean indicating if Channel B is initially selected for measurement.
- **channelCSentinel** – Boolean indicating if Channel C is initially selected for measurement.
- **channelDSentinel** – Boolean indicating if Channel D is initially selected for measurement.

- **device** – The TempicoDevice instance used to perform the measurement operations.

Signals:

- newValue(str, float, float): Emitted with updated values for display (label updates).
- updateLabel(str, float, float): Updates the label values for each channel.
- newMeasurement(float, datetime, float, float, float, float, float, float, float, float): Emitted with measurement timestamp, and count/uncertainty data for each channel.
- noTotalMeasurements(): Emitted when none of the channels provide valid stop data.
- noPartialMeasurements(list): Emitted with a list of channels that failed the threshold.
- changeStatusText(str): Updates the system status label with a custom message.
- changeStatusColor(int): Updates the system status label color code.
- disconnectedDevice(): Emitted when the Tempico device is no longer reachable.
- initialDate(str): Emitted with the starting timestamp of the measurement session.
- finalDate(str): Emitted with the ending timestamp of the measurement session.

Attributes:

- channelsMeasure: List of channels accepted for measurement after validation.
- channelsWithoutMeasurements: List of channels that failed the validation criteria.
- continueEvent: Event object used to pause/resume thread logic when user interaction is needed.
- running: Boolean flag to control the thread's main loop.
- numberStopsChannelA/B/C/D: Internal counters to track stop events for each channel.

Returns
None

`calculateIntervalWithStops`(*currentMeasure*, *numberStops*)

Calculates stop time intervals based on pulse timestamps from the measurement.

The function receives a raw measurement and the expected number of stops, then iteratively computes the time differences between consecutive valid pulse timestamps. Each interval is converted to a frequency in Hz by dividing 10^12 by the difference in timestamp units (assuming picoseconds).

Parameters

- **currentMeasure** – A list containing pulse data for a single run.
- **numberStops** – The number of stop intervals expected in the measurement.

Returns
A list of frequency values (in Hz) calculated from the stop intervals.

`changeStatusColor`

`changeStatusText`

`determineStopsNumber`(*channelTest*)

Determines the optimal number of stop pulses for a given channel to ensure valid measurements.

This function tests different stop configurations by progressively reducing the number of required stop pulses (from 5 down to 2). For each configuration, it performs multiple test measurements and checks if at

least half of them return valid data. If a configuration with sufficient measurements is found, it is returned. If none meet the condition, 1 is returned by default. If the thread is stopped during execution, the process halts.

Parameters
channelTest – A string indicating the channel to test (“A”, “B”, “C”, or “D”).

Returns
An integer representing the number of stop pulses that reliably produce measurements.

disconnectedDevice

enableDisableChannels()

Enables or disables device channels based on active measurement sentinels.

This function disables all channels initially, then enables only the selected ones based on the sentinel flags. It configures each enabled channel with a default stop mask and averaging settings, and adds them to the list of channels to be measured. This reduces processing time by avoiding unnecessary measurements from inactive channels.

Returns
None

finalDate

getMeasurements()

Retrieves and processes the latest measurement data from the Tempico device.

For each enabled channel, it verifies that the received data is valid, calculates stop intervals based on the number of pulses received, and computes both the mean and standard deviation of the intervals. These values represent the estimated count and its uncertainty.

If valid data is found, the results are emitted via a signal along with the timestamp of the measurement. If no data is available, default values are emitted depending on the state of each channel.

Returns
None

initialDate

newMeasurement

newValue

noPartialMeasurements

noTotalMeasurements

run()

Executes the thread’s main loop to perform count estimations from the Tempico device.

First, it evaluates each selected channel to determine whether valid measurements can be obtained. If a channel does not meet the required threshold, it is disabled and removed from the measurement process. If at least one channel is valid, the measurement process begins.

The loop runs continuously while the thread is active, triggering measurements every second. If the device becomes unresponsive or the user stops the measurement, the thread exits.

Returns
None

staticMetaObject = <PySide2.QtCore.QMetaObject object>

stop()

Stops the measurement process by setting the running sentinel to False.

This function is typically called from the GUI when the user requests to stop the measurement. Once the sentinel is set to False, the running loop in the thread will exit, effectively terminating the background measurement process.

Returns
None

updateLabel

CHAPTER

FIVE

CREATESAVEFILE MODULE

class createsavefile.**createsavefile**

Bases: object

check_folder_and_file()

Checks whether the folder 'TempicoSoftwareData' and the file 'data_constants.txt' exist in the user's 'Documents' directory.

The function constructs the paths for both the folder and the file, and checks if both exist: - Folder path: <Documents>/TempicoSoftwareData - File name: data_constants.txt

If both the folder and the file exist, the function returns True. Otherwise, it returns False.

Returns
bool: True if the folder and file exist, False otherwise.

create_folder()

Checks if the specified folder and file already exist. If they do exist, it creates the folder and file by calling the 'create_folder_and_file' function.

Returns
None

create_folder_and_file()

Creates a folder named 'TempicoSoftwareData' inside the user's 'Documents' directory, and creates a file named 'data_constants.txt' (or '.data_constants.txt' on Unix-based systems) inside that folder. The file is populated with predefined default values for histogram and data names. The file is hidden depending on the operating system.

The folder and file are created with the following details: - Folder path: <Documents>/TempicoSoftwareData - File name: data_constants.txt (or '.data_constants.txt' for Unix-based systems) - The file contains the following default data: - Folder Path - Default Histogram Name - Default g2 Name - Default Lifetime Name

In case of a Windows system, the file is made hidden using the Windows API. For Unix-based systems (Linux, macOS), the file is renamed to be hidden by adding a dot ('.') before the filename.

Raises
OSError – If an error occurs during folder or file creation, or when accessing the filesystem.

Returns
None

read_default_data()

Reads the default data from the file 'data_constants.txt' located in the 'TempicoSoftwareData' folder inside the user's 'Documents' directory.

This function attempts to read the file line by line and extracts key-value pairs formatted as "key: value". The first line, which contains the folder path, is processed separately. The extracted key-value pairs are stored in a dictionary, which is returned as the output.

If the file is not found or an error occurs while reading the file, the function returns None.

Returns
dict or None: A dictionary containing the data from the file, or None if an error occurred.

`save_LifeTime_data`(*data*, *file_name*, *folder_path*, *settings*, *extension*, *textLabel*)

Saves LifeTime data (time and LifeTime values) into a text file in a specified folder. The function ensures that the provided time and LifeTime values have the same length and writes them into a file along with specified settings and a label for the LifeTime values.

The file is saved in the specified folder path, with the provided file name and extension.

Parameters

- `data` – A tuple where the first element is a list of time values and the second element is a list of corresponding LifeTime values (tuple of lists).
- `file_name` – The name of the output file (str).
- `folder_path` – The path to the folder where the file will be saved (str).
- `settings` – A string representing the settings to be written in the first line of the file (str).
- `extension` – The file extension for the output file (str).
- `textLabel` – A label to be written before the LifeTime values in the file (str).

Raises
`ValueError` – If the lengths of the time and LifeTime value lists do not match.

Returns
None

`save_counts_data`(*time_stamp*, *data*, *dataUncertainties*, *filenames*, *folder_path*, *settings*, *extension*, *channels*)

`save_g2_data`(*data*, *file_name*, *folder_path*, *settings*, *extension*, *textLabel*)

Saves g2 data (tau and g2 values) into a text file in a specified folder. The function ensures that the provided tau and g2 values have the same length and writes them into a file along with specified settings and a label for the g2 values.

The file is saved in the specified folder path, with the provided file name and extension.

Parameters

- `data` – A tuple where the first element is a list of tau values and the second element is a list of corresponding g2 values (tuple of lists).
- `file_name` – The name of the output file (str).
- `folder_path` – The path to the folder where the file will be saved (str).
- `settings` – A string representing the settings to be written in the first line of the file (str).
- `extension` – The file extension for the output file (str).
- `textLabel` – A label to be written before the g2 values in the file (str).

Raises
`ValueError` – If the lengths of the tau and g2 value lists do not match.

Returns
None

`save_lists_as_columns_txt`(*data_lists*, *file_names*, *column_names*, *path*, *settings*, *extension*)

Saves multiple lists of data as separate text files, where each list is written as a column in the text file. The files are saved in the specified directory with the provided file names, and the columns are named based on the given column names.

This function ensures that the specified directory exists, and raises an error if the lengths of the input lists do not match.

Parameters

- **data_lists** – A list of lists, where each list contains the data to be written to a file (list).
- **file_names** – A list of strings specifying the names of the output files (list of str).
- **column_names** – A list of strings specifying the names of the columns in the text files (list of str).
- **path** – The directory path where the files will be saved (str).
- **settings** – A list of settings to be written as the first line in each file (list of str).
- **extension** – The file extension for the output files (str).

Raises
ValueError – If the lengths of data_lists, file_names, and column_names do not match.

Returns
None

CHAPTER

SIX

FINDDEVICES MODULE

class findDevices.**PyTempicoManager**

Bases: object

Class for managing the search and connection of Tempico devices connected to serial ports on the PC.

This class is responsible for scanning the available serial ports on the system and searching for Tempico devices connected to those ports. It facilitates interaction with Tempico devices by allowing their identification and connection.

Parameters
None – No parameters are required for the initialization of the class.

Returns
None

find_devices(*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

get_pytempico_devices()

Searches for Tempico devices among the available serial ports and returns a list of their port names.

This function scans the system's serial ports, checks if they correspond to a Tempico device by retrieving the VID and PID, and verifies if they match the Tempico device specifications. Bluetooth devices are excluded from the search.

Returns
A list of port names corresponding to Tempico devices.

Return type
list of str

get_vid_pid(*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

verify_pyTempico(*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

CHAPTER

SEVEN

GENERALSETTINGS MODULE

class generalsettings.**GeneralSettingsWindow**(*device*)

Bases: QDialog

Represents the General Settings window for the Tempico device.

This class creates a dialog window that allows users to configure general settings that affect all channels (A, B, C, D) and the start settings of the Tempico device. It provides a user interface to adjust various parameters applicable across the entire device.

Parameters
device (*object*) – The Tempico device instance that the settings will apply to.

disableSettings()

Disables the settings in the General Settings dialog.

This function sets the threshold voltage and number of runs spinbox to read-only mode, preventing any changes to these settings by the user.

Returns
None

enableSettings()

Enables the settings in the General Settings dialog.

This function allows the user to modify the threshold voltage and number of runs by enabling the corresponding spinbox and button.

Returns
None

event(*event*)

Handles the event when the "?" (What's This) help button is clicked.

This function intercepts the event triggered when the user clicks the "?" button (entering What's This mode). It exits What's This mode immediately, changes the mouse cursor back to the arrow, and displays a help window with relevant information.

Parameters
event (*QEvent*) – The event object representing the triggered event.

Returns
True if the event is handled; otherwise, it passes the event to the parent class for default processing.

Return type
bool

getsettings()

Retrieves the device's current settings, such as the number of runs and threshold voltage, and updates the corresponding values in the graphical interface.

The function fetches the number of runs and threshold voltage from the connected device, and then sets these values in the dialog's spinbox and combobox in the user interface.

Returns
None

preDefinedSettings(*thresholdVoltage*, *numberRuns*)

setsettings()

Applies the settings provided by the user to the connected device.

The function retrieves the user-defined values for the number of runs and threshold voltage from the graphical interface and sends these values to the device. After applying the settings, the function closes the settings dialog.

Returns
None

showHelp()

Displays a help dialog with information about general settings.

This function is triggered when the "?" (What's This) help button is clicked. It creates and displays a message box containing detailed information about the general settings, including instructions for the threshold voltage and the number of runs.

The message box provides the following details: - **Threshold voltage**: Specifies the acceptable range for input values (0.60 V to 1.60 V). - **Number of runs**: Describes the number of measurements performed in each channel during one data collection.

Returns
None

staticMetaObject = <PySide2.QtCore.QMetaObject object>

CHAPTER

EIGHT

MAIN MODULE

class main.**MainWindow**(*parent=None*, **args*)

Bases: QMainWindow

Main application window class.

This class is responsible for creating the main window of the application, including its tabs and graphical user interface (GUI) components. It serves as a bridge between the UI classes that handle the design elements and the logic classes that manage the application's functionalities.

The main tasks of this class include: - Initializing the main window and its layout. - Creating and managing tabs for different sections of the application. - Integrating UI design elements with the logical operations for each functionality.

Parameters

- **parent** (*QWidget, optional*) – The parent widget (optional).
- **args** (*tuple*) – Additional arguments.

Helpg2Button()

about_settings()

Opens the About window displaying information about the company, version, and repository.

This function opens a window showing details about the company (Tausand), the software version, and the location of the repository. It does not perform any checks or validations and directly displays the About window.

It does not take any parameters and does not return a value. :returns: None

activeMeasurement()

Activates the sentinel indicating that a measurement is currently running.

This function sets the internal flag to True, which is used to track the active state of a measurement process.

Returns
None

clicked_tabs()

Executes an action when a tab is clicked and creates the corresponding window.

This function checks which tab is currently active and constructs the associated window by invoking the appropriate function. If the tab corresponds to the Start/Stop Histogram, it stops the LifeTime timer and constructs the Start/Stop Histogram window. If the tab corresponds to the Lifetime Measurements, it constructs the Lifetime window and sets up the LifeTime logic if it has not been initialized yet.

It does not take any parameters and does not return a value. :returns: None

closeEvent(*event*)

Handles the close event of the main window and prompts the user for confirmation.

This function is triggered when the user attempts to close the main window. It displays a dialog asking if the user is sure about closing the application (Tempico software). If the user confirms by selecting "Yes", the event is accepted and the application closes. If the user selects "No", the close event is ignored, and the application remains open.

Parameters
event (*QCloseEvent*) – The close event triggered when attempting to close the window.

Returns
None

construct_counts_estimated(*parent*)

Constructs the Counts Estimated window.

This function takes a *QTabWidget* parent, and if the sentinel is not set, it creates an instance of the *Ui_CountsEstimated* class and sets up the UI using the given parent.

It ensures the UI is initialized only once by checking the *sentinel3* flag.

Parameters
parent (*QWidget*) – The parent widget (typically a *QTabWidget*) for the counts estimated window.

Returns
None

construct_g2(*parent*)

Constructs the g2 Measurements window.

This function takes a *QTabWidget* parent, and if the sentinel is not set, it creates an instance of the *Uig2* class and sets up the UI using the given parent.

It does not return a value.

Parameters
parent (*QWidget*) – The parent widget (typically a *QTabWidget*) for the lifetime measurements window.

Returns
None

construct_lifetime(*parent*)

Constructs the Lifetime Measurements window.

This function takes a *QTabWidget* parent, and if the sentinel is not set, it creates an instance of the *UiLifeTime* class and sets up the UI using the given parent.

It does not return a value.

Parameters
parent (*QWidget*) – The parent widget (typically a *QTabWidget*) for the lifetime measurements window.

Returns
None

construct_start_stop_histogram(*parent*)

Constructs the Start/Stop Histogram window.

This function takes a *QTabWidget* parent, and if the sentinel is not set, it creates an instance of the *Ui_HistogramaStartStop* class and sets up the UI using the given parent.

It does not return a value.

Parameters
parent (*QWidget*) – The parent widget (typically a *QTabWidget*) for the histogram window.

Returns
None

`disconnect_button_click()`

Handles the disconnect button click event.

This function hides the graphical display, disables the disconnect button, and re-enables the connect button. It also closes the connected device and resets its reference to *None*. If additional graphics like *g2Graphic* or *LifeTimeGraphic* are active, it will disconnect them as well.

It does not take any parameters and does not return a value. :returns: None

`disconnectedDevice()`

Handles device disconnection without displaying any user interface dialogs.

This function is triggered when the connection with the device is lost. It notifies all relevant components to handle the disconnection and sets the connected device reference to None.

Returns
None

`enableSettings()`

Re-enables the device and general settings dialogs if they are open.

If either the device configuration or general settings dialog is currently visible, this function refreshes and reactivates their interactive fields by calling the corresponding *get_settings()* and *enableSettings()* methods.

Returns
None

`general_settings_clicked()`

Opens the general settings window for device-wide configurations.

This function checks if a device is connected. If no measurement is currently running, it opens the general settings window where configurations that affect the entire device (regardless of the channel) can be adjusted. If a measurement is running, a message box is displayed to inform the user that changes cannot be made while the measurement is in progress. If no device is connected, a message box is shown indicating that no device was found.

It does not take any parameters and does not return a value. :returns: None

`manageConection()`

Manages the connection status with the measurement device.

This function checks if the device is still connected. If the connection is lost, it stops the timer responsible for monitoring the device, removes the device instance, disables measurement functionalities in all tabs, and displays an error dialog to inform the user.

Returns
None

noMeasurement()

Deactivates the sentinel indicating that no measurement is currently running.

This function sets the internal flag to False, marking that there is no active measurement process at the moment.

Returns
None

open_dialog()

Opens a dialog window to detect and connect a measurement device.

This function creates and displays a dialog window that lists available measurement devices and their corresponding ports. Users can select a device and either connect or cancel the operation.

If the 'Connect' button is clicked: - The selected device is connected, and all relevant software options are enabled. - The function tries to open the device and, upon success, interacts with graphical components (if any are present) to reflect the connected status.

If the 'Cancel' button is clicked: - No options are activated.

In case of a connection error, a message box is displayed informing the user of the failure.

Returns
None

parameters_action()

resizeEvent(*event*)

Handles the window resize event and adjusts the column widths of the parameters table.

This function is triggered when the user resizes the main window. It resizes the columns of the parameters table in the LifeTime tab based on the current window width. The column widths are scaled proportionally to ensure the table adapts to the new window size.

Parameters
event (*QResizeEvent*) – The resize event triggered when the window is resized.

Returns
None

saveSettings()

Retrieves and stores the current device configuration into internal variables.

This function queries the connected device for channel-specific settings such as averaging cycles, mode, number of stops, stop edge type, stop mask, threshold voltage, and number of runs. These values are saved into instance attributes for later use, such as restoring settings or displaying them in the UI.

Returns
None

settings_clicked()

Opens the settings window when the settings option is clicked.

This function checks if a device is connected. If no measurement is currently running, it displays the settings window for channel configuration. If a measurement is running, a message box is shown informing the user that changes cannot be made while a measurement is in progress. If no device is connected, a message box alerts the user that no device was found.

It does not take any parameters and does not return a value. :returns: None

staticMetaObject = <PySide2.QtCore.QMetaObject object>

class main.SplashScreen

Bases: QMainWindow

Splash screen class for the application.

This class creates a splash screen that displays an image for a brief period before fading out and transitioning to the main application window.

The key responsibilities of this class include: - Displaying a splash screen with a specified image. - Timing the display duration and handling the transition to the main application window.

Parameters
parent (*QWidget, optional*) – The parent widget (optional).

show_main_window()

Displays the main window of the application.

This function creates an instance of the main window (*MainWindow*) and displays it. After showing the main window, it closes the current window.

It does not take any parameters and does not return a value. :returns: None

staticMetaObject = <PySide2.QtCore.QMetaObject object>

main.execProgram()

CHAPTER

NINE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX

D

E

F

G

H

I

K

L

M

N

O

P

R

S

T

U

V

W

Z