Developer Manual

`entry.py`

Overview

`entry.py` is a Python script designed to serve as the entry point for a PyQt5-based application. It includes classes for the main window (`InfoWindow`), error handling (`Error`), and a success message (`SucessConnect`). The script integrates various modules for managing user interface, handling device connections, and displaying different types of information.

Dependencies

1. **PyQt5:** For GUI components.
2. **pyvisa:** For managing connections to instruments via VISA interface.
3. **Custom Modules:** `mainWindow`, `setParameter`, `measure`, `waveform_plot`.

Classes

1. **InfoWindow** (inherits `QMainWindow`)
    - **Purpose:** Main application window.
    - **Methods:**
        - `__init__`: Initializes the window and connects signals to slots.
        - `usr_btn_callback`: Opens a user manual web page.
        - `dev_btn_callback`: Opens a developer manual web page.
        - `contribution_btn_callback`: Opens a contribution page.
        - `source_btn_callback`: Opens a GitHub repository page.
        - `check_btn_callback`: Validates the connection to the instrument.
        - `connection_test`: Tests connection with the provided address.
        - `error_msg`: Displays an error message dialog.
        - `sucess_msg`: Displays a success message dialog.
2. **Error** (inherits `QDialog`)
    - **Purpose:** Displays an error message.
    - **Methods:**
        - `__init__`: Initializes the error dialog.
3. **SucessConnect** (inherits `QDialog`)

    - **Purpose:** Displays a success message and transitions to the main control panel.
    - **Methods:**
        - `__init__`: Initializes the success dialog.
        - `close_and_begin`: Closes the success dialog and opens the main window.

Main Execution

- The script initializes a `QApplication`, creates an `InfoWindow` instance, and enters the main application loop.

UI Files

- `info.ui`: UI layout for `InfoWindow`.
- `error.ui`: UI layout for `Error` dialog.
- `sucess.ui`: UI layout for `SucessConnect` dialog.

Error Handling

- The script includes basic error handling for instrument connection issues, displaying relevant messages through `Error` and `SucessConnect` dialogs.

Networking

- Web browser interactions for opening URLs.

Notes

- Ensure all UI files and modules (`mainWindow`, `setParameter`, `measure`, `waveform_plot`) are available in the application directory.
- The application requires a VISA-compatible instrument connected to the system for full functionality.

Potential Enhancements

- Improve error handling with more specific messages.
- Refactor to make the code more modular and maintainable.
- Add logging for better debugging and maintenance.

This manual provides a high-level overview. For detailed understanding, reviewing the individual methods and their interactions within the script is recommended.

`mainWindow.py`

Overview

`mainWindow.py` is a Python module designed for managing the main window of a PyQt5-based application. It primarily focuses on interfacing with measurement instruments, controlling various parameters, and displaying relevant information. It uses custom modules for specific functionalities and integrates with pyvisa for instrument communication.

Dependencies

1. **PyQt5:** For GUI components.
2. **pyvisa:** For instrument communication.
3. **Custom Modules:** `setParameter` (sp), `measure` (ms), `waveform_plot` (wp).

Classes

1. **MyWindowClass** (inherits `QMainWindow`)
    - **Purpose:** Manages the main application window.
    - **Key Features:**
        - Channel selection and control.
        - Impedance and slope settings.
        - Vertical and horizontal scaling and offset adjustments.
        - Memory depth and sampling rate settings.
        - Waveform display and data download options.
    - **Methods:**
        - `__init__`: Initializes the main window and UI components.
        - `open_manual`: Opens the user manual in a web browser.
        - Callback methods for UI interactions (e.g., `optionmenu_channel_callback`, `impedance_callback`, `auto_scale_callback`, etc.).
        - Methods for opening various dialog windows (e.g., `pop_window_measure`, `developing_pop_window_waveform_data`, etc.).
2. **Developing_screenshot** (inherits `QDialog`)
    - **Purpose:** Handles downloading screenshots from the instrument.
    - **Methods:**
        - `__init__`: Initializes the dialog.
        - `open_folder_dialog`: Opens a folder selection dialog.
        - `data_download_callback`: Handles the screenshot download process.

3. **Developing_wave_plot** and **Developing_Waveform_data** (inherit `QDialog`)
    - **Purpose:** Manage downloading of waveform plots and data.
    - **Methods:**
        - `__init__`: Initializes the respective dialog.
        - `open_folder_dialog`: Opens a folder selection dialog.
        - `data_download_callback`: Handles the download process for waveform data or plots.
4. **MeasureSelection** (inherits `QMainWindow`)
    - **Purpose:** Facilitates selection and display of various measurements.
    - **Methods:**
        - `__init__`: Initializes the measurement selection window.
        - `updateTheValue`: Updates the measurement type based on user selection.
        - `runMeasurements`: Executes the measurement process and displays results.
5. **Measure** (inherits `QDialog`)
    - **Purpose:** Manages data measurements and downloads.
    - **Methods:**
        - `__init__`: Initializes the data measurement dialog.
        - `download_callback`: Handles the downloading of measurement data.
6. **Screenshot** (inherits `QDialog`)
    - **Purpose:** Manages downloading screenshots.
    - **Methods:**
        - `__init__`: Initializes the screenshot download dialog.
        - `screenshot_download_callback`: Handles the screenshot download process.

UI Files

- Each class loads its respective `.ui` file, which defines the layout and design of the UI elements.

Networking

- The script includes functionality to open web pages for manuals and other resources.

Error Handling

- Basic error handling for user interactions. Further robustness can be added.

Potential Enhancements

- Improve error handling and user feedback.
- Refactor for better modularity and separation of concerns.
- Implement logging for debugging and operational insights.

This manual provides a general overview and structure of `mainWindow.py`. For a comprehensive understanding, it is recommended to review the individual methods and their interactions within the script, along with the corresponding UI layouts.

`measure.py`

Overview

`measure.py` is a Python script designed for interfacing with measurement instruments and recording various measurements such as rise time, fall time, frequency, etc. It outputs the collected data to a CSV file. The script uses the `pyvisa` library for instrument communication and `pandas` for data handling.

Dependencies

1. **pyvisa:** For communication with measurement instruments.
2. **pandas:** For data manipulation and exporting to CSV.
3. **sys:** Used for system-level operations, although its direct usage isn't apparent in the current script.

Key Functions

1. **initialize(obj)**
    - Initializes the global `scope` variable with the provided instrument object.
    - **Parameters:**
        - `obj`: The instrument object to be used for subsequent measurements.
2. **Measurement Functions**
    - Functions to measure specific parameters from the instrument.
    - Includes `mea_rise_time`, `mea_fall_time`, `mea_frequency`, `mea_period`, `mea_amplitude`, `mea_pulse_width`, `mea_duty_cycle`.
    - **Parameter:**
        - `source`: The channel from which the measurement is taken, defaulting to "CHANnel1".
3. **measure(source, path, debug)**

- Main function to measure various parameters and save them to a CSV file.
- **Parameters:**
  - `source`: The measurement source channel.
  - `path`: Path for the output CSV file.
  - `debug`: Boolean to activate debug prints.

4. **do_command(command, hide_params)**
   - Sends a command to the instrument.
   - **Parameters:**
     - `command`: The command string.
     - `hide_params`: Boolean to hide parameters in logs.
5. **do_query_string(query)**
   - Sends a query and returns the response as a string.
   - **Parameter:**
     - `query`: Query string to be sent.
6. **do_query_number(query)**
   - Sends a query and returns the response as a numerical value.
   - **Parameter:**
     - `query`: Query string to be sent.

Error Handling

- Basic error handling is implemented, primarily through the `pyvisa` library's inherent error management. Further robust error handling can be implemented for specific use cases.

Debugging

- The `debug` parameter in the `measure` function allows for basic debugging by printing measurement results to the console.

Data Handling

- Measurement data is organized and written to a CSV file using `pandas`, making it easy to analyze and share.

Potential Enhancements

1. Implementing more robust error handling and exception management.
2. Adding units to measurement results for clarity.
3. Addressing the noted bug in getting the duty cycle and expanding the range of measurements.
4. Refactoring for better modularity and readability.

- -

Notes

- The script contains commented-out sections for direct execution and testing. These can be utilized for standalone testing or debugging purposes.
- The script also includes placeholders for future improvements, indicating an ongoing development process.

This manual provides a basic understanding of `measure.py`. For in-depth knowledge, review the individual functions and their interactions, especially how they communicate with the measurement instrument.

`setParameter.py`

Overview

`setParameter.py` is a Python script primarily aimed at setting various parameters for an oscilloscope using the `pyvisa` library. It provides a set of functions to control different aspects of the oscilloscope such as autoscaling, channel selection, impedance, trigger slope, points acquisition, sample rate, and scaling.

Dependencies

1. **pyvisa:** For communication with the oscilloscope.
2. **sys, pandas, tkinter:** Included but not directly used in the current script.

Key Functions

1. **initialize(obj)**
   - Initializes the global `scope` variable with the provided oscilloscope object.
2. **do_command(command, hide_params)**
   - Sends a command to the oscilloscope.
   - **Parameters:**
     - `command`: Command string.
     - `hide_params`: Boolean to hide command parameters in logs.
3. **Frontend Functions**
   - Functions that map user interface actions to oscilloscope settings.
   - Includes `auto_scale_click`, `channel_control_select`, `impedance_control_select`, `trigger_slope_select`, etc.
4. **Backend Functions**

    - Functions that directly interact with the oscilloscope to apply settings.
    - Includes `autoScle`, `choose_channel`, `average_on_off`, `impedance_select`, `choose_trigger_slope`, `points_acquire`, `points_auto_clicked`, `sample_rate`, `sample_auto_clicked`, `vertical_scaling`, `vertical_offset`, `horizontal_scaling`, `horizontal_offset`.

GUI Components

- The script contains commented-out sections for a basic graphical user interface using `tkinter`. These components are intended for direct user interaction for setting oscilloscope parameters.

Error Handling

- Basic error handling is implemented through `pyvisa` library's inherent error management. Additional error handling can be implemented for specific use cases.

Usage Notes

- The script currently has a global `source` variable set to `"CHANnel1"` by default, which can be modified based on the channel being used.
- The script contains several commented-out lines and placeholders, indicating potential areas for future development or testing.

Potential Enhancements

1. Implement robust error handling and validation for oscilloscope commands.
2. Activate and utilize the commented-out GUI components for a more interactive parameter setting experience.
3. Remove unused imports and clean up the code for better readability and performance.
4. Document each function with detailed comments explaining their purpose and usage.

Debugging and Testing

- The script provides a structure to test and debug oscilloscope settings directly via Python scripts, which can be advantageous for automation and remote control scenarios.

This manual provides a basic outline of `setParameter.py`. To fully understand and utilize this script, it is recommended to review the individual functions, especially how they interact with the oscilloscope hardware, and consider activating the GUI components for a more interactive experience.