QConnectWinapp

v. 1.0.0

Nguyen Huynh Tri Cuong

25.04.2023

Contents

Chapter 1

Introduction

QConnectWinapp is an extension library for QConnectBase library, designed to simplify and automate Windows application GUI testing. QConnectWinapp is built to provide seamless and efficient GUI testing experiences for Windows applications.

The library uses the powerful backend of WinAppDriver (https://github.com/microsoft/WinAppDriver) to drive the testing of your Windows applications, ensuring reliable and accurate results. QConnectWinapp is designed to support Python 3.7+, RobotFramework 3.2+, and QConnectBase 1.0.0+.

With QConnectWinapp, you can easily and quickly automate the testing of your Windows applications, saving time and effort while ensuring that your applications are of the highest quality. Whether you're a seasoned tester or just starting out, QConnectWinapp offers a range of features and tools to make GUI testing simple and efficient.

Chapter 2

Description

QConnectionWinapp

2.1 Prerequisites

To use the QConnectWinapp library, users need to install the following app prerequisites:

WinAppDriver: https://github.com/Microsoft/WinAppDriver/releases (version >=1.2.1)

Windows SDK: https://developer.microsoft.com/en-us/windows/downloads/windows-sdk/

2.2 Getting Started

You can checkout all QConnectWinapp sourcecode from the GitHub.

After checking out the source completely, you can install by running below command inside **robotframework-qconnect-winapp** directory.

```
python setup.py install
```

2.3 Usage

QConnectWinapp is a backend extension for the QConnectBase library that adds support for testing WinApp UI. From the user's perspective, this means they now have an additional connection type for WinApp testing when using the **connect** keyword in the QConnectBase library. With QConnectWinapp, users can now easily automate tests for Windows desktop applications, and take advantage of its integration with WinAppDriver to interact with UI elements and validate their behavior.

Please refer to QConnectBase for more information on how to use the **connect** keyword. In this section, we will focus on the **Winapp** connection type and how to configure it.

2.3.1 Configurations for Winapp connection type

QConnectBaseLibrary has already supported below connection types:

- **TCPIPClient**: Create a Raw TCPIP connection to TCP Server.
- **SSHClient**: Create a client connection to a SSH server.
- **SerialClient**: Create a client connection via Serial Port.
- **DLT**: Create a client connection to Diagnostic Log and Trace(DLT) Module.

QConnectWinapp add one more connection type for Winapp UI testing call **Winapp**

Below is the description of the configuration string for Winapp connection type:

```
{
    "host": [host ip],      # Optional. Default value is "localhost".
    "port": [listening port]  # Optional. Default value is 4723.
    "caps": [Desired Capabilities string in JSON format],  # E.g. { "app": "C:/Program ↩
    ↪ Files/BOSCH/CMST/CMST.exe"}
    "logfile": [Log file path. Possible values: 'nonlog', 'console', [user define path]]
}
```

Table 2.1: List of Capabilities

Capability	Description
app	The package full name or the executable file's full path of the application to automate (e.g. Microsoft.WindowsCalculator_8wekyb3d8bbwe!App).
deviceName	Mostly optional, but the name of your device
automationName	The name of the automation engine to use (e.g. windows).

2.3.2 Using the Automation Inspector Tool to define a GUI control

The Automation Inspector tool provides a convenient way to interact with controls of an Application Under Test (AUT) and create resource files for testing. Follow the steps below to effectively utilize the tool:

Accessing the Tool

The Automation Inspector tool can be found at the following location: C:/Program Files/RobotFramework/python39/Lib/site-packages/QConnectWinapp/tools/bin/release.

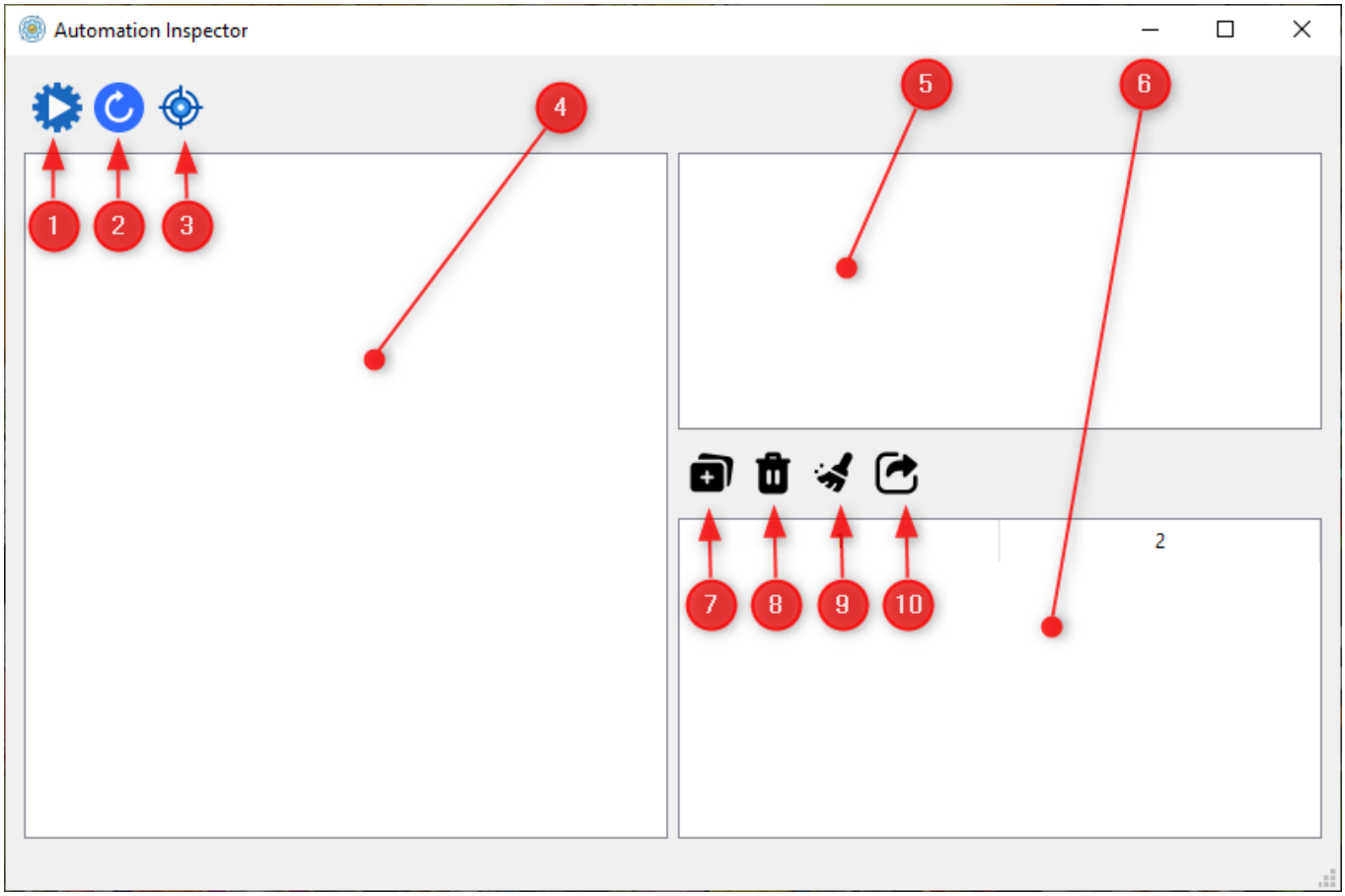

Figure 2.1: Automation Inspector tool.

Tool Overview

Upon launching the Automation Inspector tool, you will be presented with a graphical user interface that offers a range of functionalities. The main features (please refer to the Figure 2.1) of the tool are described below:

1. **Exec AUT button:** Press this button to start an Application Under Test (AUT).
2. **Refresh button:** Press this button to refresh the information of the AUT.
3. **Inspect controls button:** After pressing this button, users can hover their mouse over controls on the AUT. The tool will automatically highlight and display the attributes of the currently highlighted control.
4. **Elements treeview area:** This area displays the list of controls of the AUT in a hierarchical tree structure. Clicking on a node in the tree displays its attributes in the Properties table (5).
5. **Properties table:** This area displays the attributes of the node selected in the Elements treeview area (4) or highlighted using the Inspect controls feature (3). Users can check checkboxes to select attributes for defining the control to be tested and add them to the Definitions table (6) using the Add definition button (7).
6. **Definitions table:** This table contains the definitions of controls to be tested. Users can select attributes in the Properties table (5) and add them to this table using the Add definition button (7). Users can also check checkboxes to select definitions for exporting to a .resource file using the Export button (10).
7. **Add definition button:** Press this button to add the selected attributes from the Properties table (5) as definitions for a control in the Definitions table (6).
8. **Remove Definition button:** Press this button to remove the selected definition from the Definitions table (6).
9. **Clear Definitions button:** Press this button to remove all definitions from the Definitions table (6).
10. **Export button:** Export all selected definitions from the Definitions table (6) into a .resource file.

Creating a Resource File

To create a resource file using the Automation Inspector tool, follow these steps:

1. Launch the Automation Inspector tool from C:/Program Files/RobotFramework/python39/Lib/site-packages/QConnectWinapp/tools/bin/release.
2. Start the Application Under Test (AUT) by clicking the **Exec AUT** button. After clicking, a dialog will open, allowing the user to select the path to the executable file for the AUT. In the following example, I will input the path for the Calculator application.

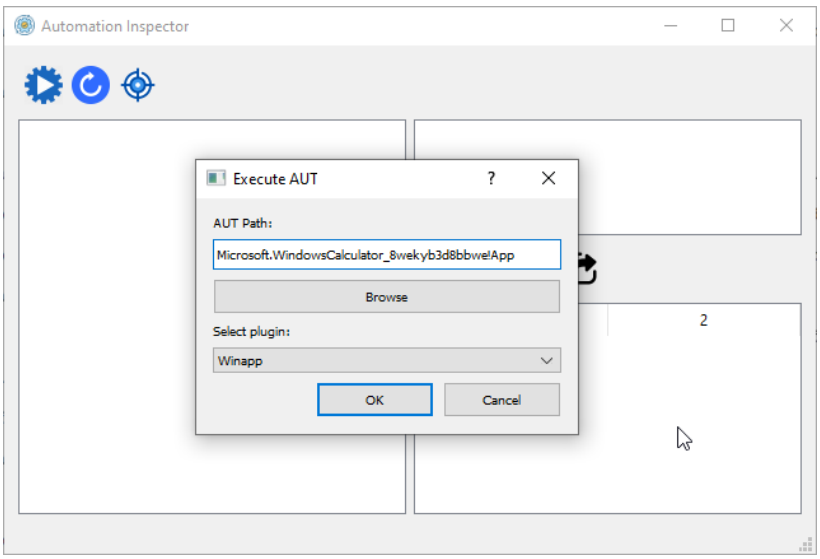

Figure 2.2: Select the AUT path to execute

3. After selecting the AUT executable, click the **OK** button to start the AUT. The **Elements treeview** will be populated, showing a hierarchical tree of controls present in the AUT.

4. To view the properties of a control, you can either:

   - Click on a node in the **Elements treeview** to display its properties in the **Properties table**.
   - Select the **Inspect controls** button and hover the mouse over a control in the AUT. The corresponding control in the **Elements treeview** will be selected and its properties will be shown in the **Properties table**.

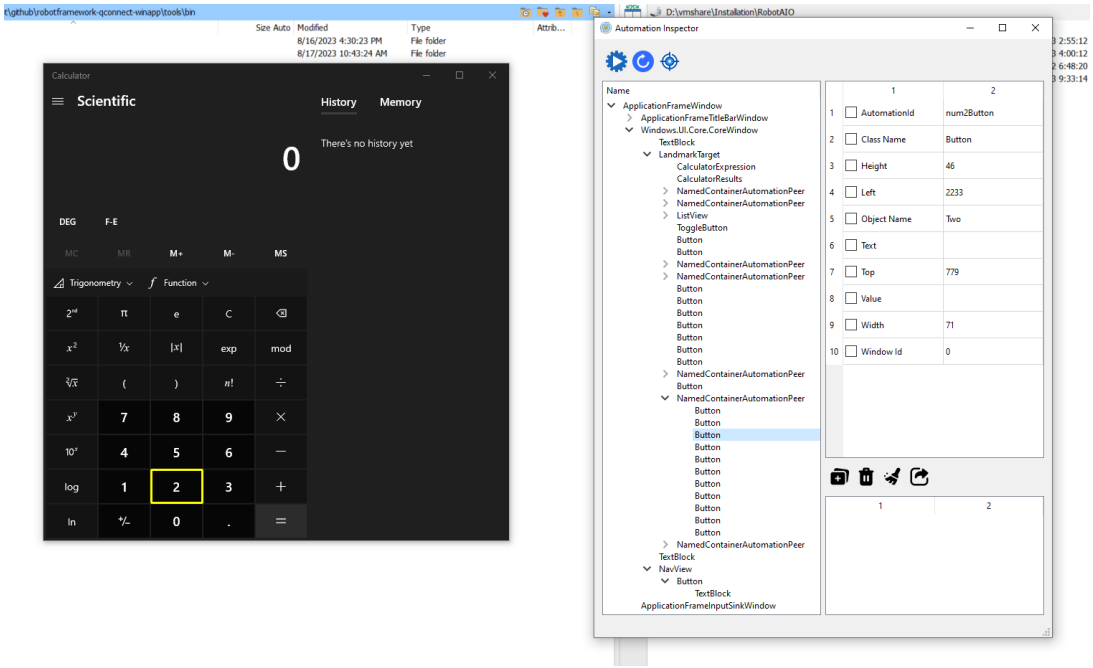

Figure 2.3: View control properties

5. Select the desired attributes for control definitions by checking the checkboxes in the **Properties table**.

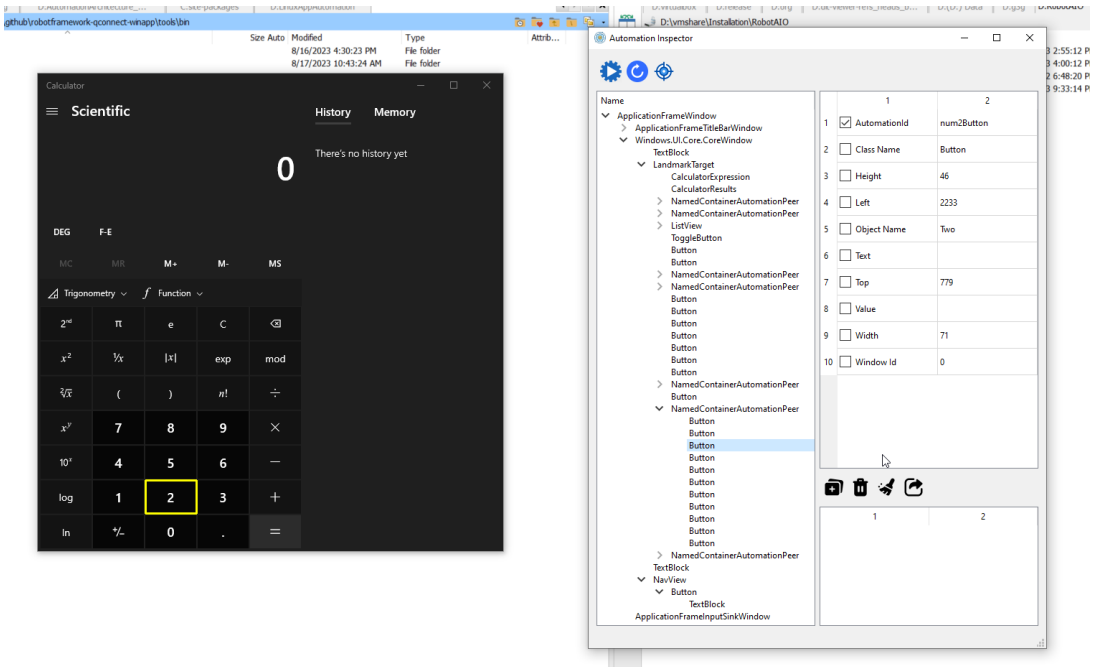

Figure 2.4: Select property for definition

6. Click the **Add definition** button to add the selected attributes to the **Definitions table**. After clicking the button, a dialog will appear for the user to declare a name corresponding with the definition. In the following example, I will fill the name as 'btn2Num', and the definition will be ""AutomationId":"num2Button"", which means that the 'btn2Num' is a control with the AutomationId property value of 'num2Button'.

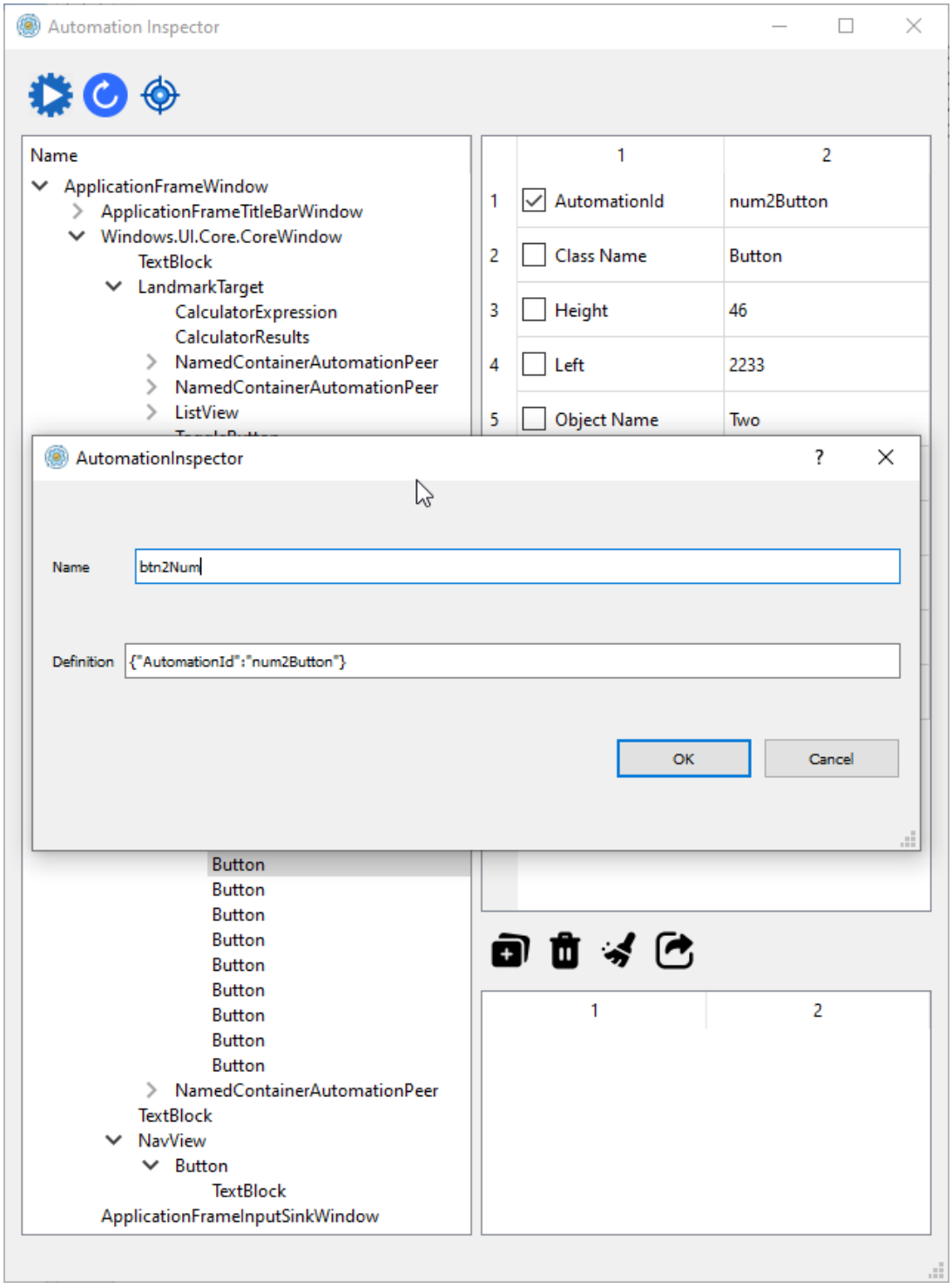

Figure 2.5: Declare a test automation control

7. Repeat the process to define multiple controls and their attributes.

8. Use the **Export** button to export selected definitions to a .resource file.

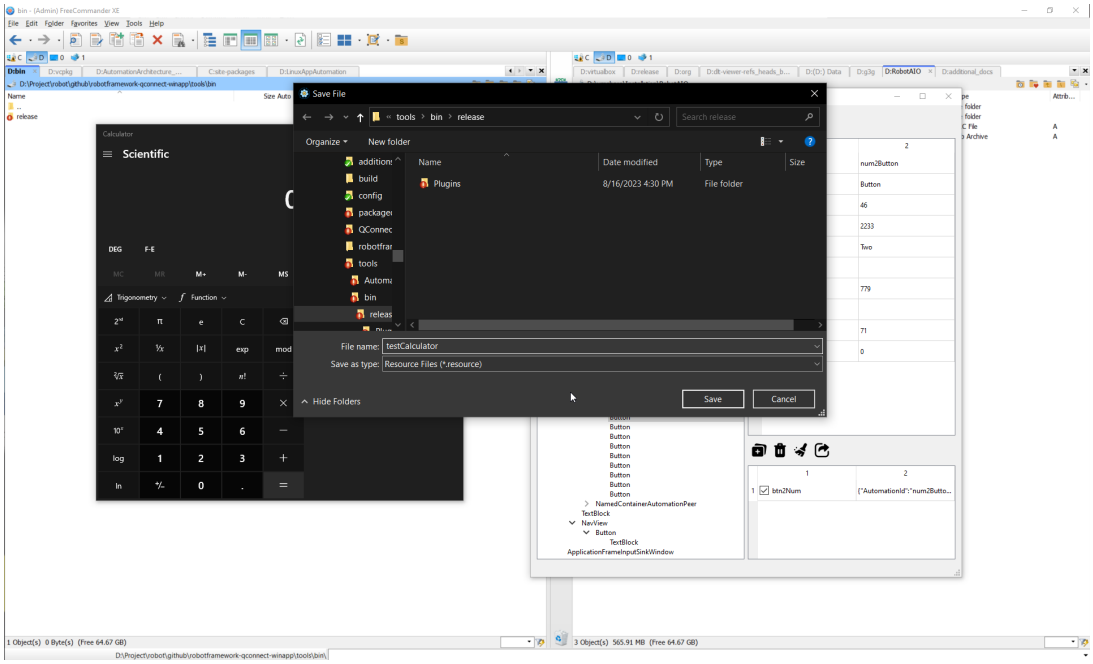

Figure 2.6: Export to resource file

By following these steps, you can effectively utilize the Automation Inspector tool to interact with controls, define attributes, and create resource files for testing purposes.

2.4 Example

In this example, I provide a test scenario that allows adding on the Calculator app as follows: the script will open the Calculator application on Windows, then press the '8' button, press the '+' button, press the '9' button, and then press the '=' button. Then, verify if the result textbox displays the result '17'.

```
*** Settings ***
Documentation    Suite description
Library     QConnectBase.ConnectionManager
Resource    QConnectWinapp/GUIAction.resource

*** Variables ***
${CONNECTION_NAME}  TEST_CONN

&{Number 8}       AutomationId=num8Button
&{Number 9}       AutomationId=num9Button
&{Equal}          AutomationId=equalButton
&{Result}         AutomationId=CalculatorResults
&{Plus}           AutomationId=plusButton


*** Test Cases ***
Test Adding
    ${config_string}=    catenate
    ...   {\n
    ...         "host":   "localhost",\n
    ...         "port": 4723,\n
    ...         "caps":\n
    ...         {\n
    ...               "app": "Microsoft.WindowsCalculator_8wekyb3d8bbwe!App"\n
    ...         }\n
    ...   }\n

    log to console       \nConnecting with below configurations:\n${config_string}
    ${config}=             evaluate       json.loads('''${config_string}''')    json
```

```
    connect             conn_name=${CONNECTION_NAME}
    ...                 conn_type=Winapp
    ...                 conn_conf=${config}

    send command        conn_name=${CONNECTION_NAME}
    ...                 element_def=${Number 8}
    ...                 command=${Action.click}

    Sleep    1s

    send command        conn_name=${CONNECTION_NAME}
    ...                 element_def=${Plus}
    ...                 command=${Action.click}

    Sleep    1s

    send command        conn_name=${CONNECTION_NAME}
    ...                 element_def=${Number 9}
    ...                 command=${Action.click}

    Sleep    1s

    send command        conn_name=${CONNECTION_NAME}
    ...                 element_def=${Equal}
    ...                 command=${Action.click}

    ${res}=     verify      conn_name=${CONNECTION_NAME}
    ...                     element_def=${Result}
    ...                     search_pattern=17
    ...                     send_cmd=${Action.get_text}
    ...                     timeout=20

    log to console  \nCalculation result: ${res}[0]


*** Keyword ***
Close Connection
    disconnect  ${CONNECTION_NAME}
```

Explanation:

- **&{Number 8} AutomationId=num8Button**: This line is used to identify the 'Number 8' button as the control on the application with an Accessible Name (AutomationId) of 'num8Button'.
- **"app": "Microsoft.WindowsCalculator_8wekyb3d8bbwe!App"**: This line specifies that the AUT is the application with the package name 'Microsoft.WindowsCalculator_8wekyb3d8bbwe!App'. You can also use the full path to the exe file of the AUT.
- **send command conn_name=${CONNECTION_NAME}...command=${Action.click}**: This line means to click on the Number 8 button on the AUT.
- **${Action.click}** has been defined in the resource file **QConnectWinapp/GUIAction.resource**.

  Currently supported in the GUIAction.resource:

Action	Description
${Action.click}	Clicks on a GUI element
${Action.get_text}	Gets the text of a GUI element
${Action.get_visible}	Get the visibility of a GUI element
${Action.get_enable}	Check if a GUI element is eanbled

Table 2.2: Actions supported in GUIAction.resource

2.5 Contribution Guidelines

QConnectBaseLibrary is designed for ease of making an extension library. By that way you can take advantage of the QConnectBaseLibrary's infrastructure for handling your own connection protocal. For creating an extension library for QConnectBaseLibrary, please following below steps.

1. Create a library package which have the prefix name is **robotframework-qconnect-***[your specific name]*.
2. Your hadling connection class should be derived from **QConnectionLibrary.connection_base.ConnectionBase** class.
3. In your *Connection Class*, override below attributes and methods:

   - **_CONNECTION_TYPE**: name of your connection type. It will be the input of the conn_type argument when using **connect** keyword. Depend on the type name, the library will detemine the correct connection handling class.
   - **__init__(self, _mode, config)**: in this constructor method, you should:
     - Prepare resource for your connection.
     - Initialize receiver thread by calling **self._init_thread_receiver(cls._socket_instance, mode="")** method.
     - Configure and initialize the lowlevel receiver thread (if it's necessary) as below

       ```
       self._llrecv_thrd_obj = None
       self._llrecv_thrd_term = threading.Event()
       self._init_thrd_llrecv(cls._socket_instance)
       ```

     - Incase you use the lowlevel receiver thread. You should implement the **thrd_llrecv_from_connection_interface** method. This method is a mediate layer which will receive the data from connection at the very beginning, do some process then put them in a queue for the **receiver thread** above getting later.
     - Create the queue for this connection (use Queue.Queue).
   - **connect()**: implement the way you use to make your own connection protocol.
   - **_read()**: implement the way to receive data from connection.
   - **_write()**: implement the way to send data via connection.
   - **disconnect()**: implement the way you use to disconnect your own connection protocol.
   - **quit()**: implement the way you use to quit connection and clean resource.

2.6 Configure Git and correct EOL handling

Here you can find the references for Dealing with line endings.

Every time you press return on your keyboard you're actually inserting an invisible character called a line ending. Historically, different operating systems have handled line endings differently. When you view changes in a file, Git handles line endings in its own way. Since you're collaborating on projects with Git and GitHub, Git might produce unexpected results if, for example, you're working on a Windows machine, and your collaborator has made a change in OS X.

To avoid problems in your diffs, you can configure Git to properly handle line endings. If you are storing the .gitattributes file directly inside of your repository, than you can asure that all EOL are manged by git correctly as defined.

2.7 Sourcecode Documentation

For investigating sourcecode, please refer to QConnectWinapp Documentation

2.8 Feedback

If you have any problem when using the library or think there is a better solution for any part of the library, I'd love to know it, as this will all help me to improve the library. Connect with me at cuong.nguyenhuynhtri@vn.bosch.com.

Do share your valuable opinion, I appreciate your honest feedback!

2.9 About

2.9.1 Maintainers

Nguyen Huynh Tri Cuong

2.9.2 Contributors

Nguyen Huynh Tri Cuong

Thomas Pollerspoeck

2.9.3 3rd Party Licenses

You must mention all 3rd party licenses (e.g. OSS) licenses used by your project here. Example:

Name	License	Type
Apache Felix.	Apache 2.0 License.	Dependency

2.9.4 Used Encryption

Declaration of the usage of any encryption (see BIOS Repository Policy §4.a).

2.9.5 License

Chapter 3

element_handler.py

3.1 Class: ElementActionHandler

Imported by:

```
from QConnectWinapp.ActionHandlers.element_handler import ElementActionHandler
```

3.1.1 Method: get_supported_level

Get the supported level of this handler for a specific element object.

Arguments:

- `ele_obj`
  / *Condition*: required / *Type*: WebElement /
  Winapp GUI element object.

Returns:

/ *Type*: int /
Supported level of this action handler for the element object.

3.1.2 Method: get_attribute

Get element object's property.

Arguments:

- `attr`
  / *Condition*: required / *Type*: str /
  Property's name to be got value.

Returns:

/ *Type*: str /
Property's value.

3.1.3 Method: divert_action

Divert action string to the corresponding method, execute the method and return value.

Arguments:

- `action`
  / *Condition*: required / *Type*: str /
  Action string to be diverted.

Returns:

/ *Type*: str /
Corresponding method's return.

Chapter 4

winapp_client.py

4.1 Class: CustomWebDriver

Imported by:

```
from QConnectWinapp.WinappDriver.winapp_client import CustomWebDriver
```

Customer WebDriver class for Winapp.

4.1.1 Method: start_session

Creates a new session with the desired capabilities.

Override for Winapp

Arguments:

- `capabilities`
  / *Condition*: required / *Type*: Union /
  Read https://github.com/appium/appium/blob/master/docs/en/writing-running-appium/caps.md for more details.
- `browser_profile`
  / *Condition*: optional / *Type*: str / *Default*: None /
  Browser profile

4.2 Class: WinappConfig

Imported by:

```
from QConnectWinapp.WinappDriver.winapp_client import WinappConfig
```

Class to store the configuration for SSH connection.

4.3 Class: WinAppClient

Imported by:

```
from QConnectWinapp.WinappDriver.winapp_client import WinAppClient
```

Winapp client connection class.

4.3.1 Method: connect

Connect to WinappDriver which is listening on configured port.

Returns:

(*no returns*)

4.3.2 Method: perform_action

Perform an action on user defined element.

Arguments:

- `obj_defined_dict`
  / *Condition*: required / *Type*: dict /
  User's definition for a GUI element.
- `cmd`
  / *Condition*: required / *Type*: str /
  Action to be perform on GUI element.
- `time_wait`
  / *Condition*: optional / *Type*: int / *Default*: 0 /
  Timeout to find a GUI element based on user's definitions.

Returns:

/ *Type*: WebElement /
GUI element.

4.3.3 Method: send_obj

Action to be send to a GUI element.

Arguments:

- `obj`
  / *Condition*: required / *Type*: str /
  Data to be sent.
- `element_def`
  / *Condition*: required / *Type*: dict /
  User's definition for a GUI element.
- `args`
  / *Condition*: optional / *Type*: tuple /
  Optional arguments.

Returns:

(*no returns*)

4.3.4 Method: wait_4_trace

Perform an action on GUI element and wait to receive a return which matches to a specified regular expression.

Arguments:

- `search_obj`

  / *Condition*: required / *Type*: str /

  Regular expression all received trace messages are compare to. Can be passed either as a string or a regular expression object. Refer to Python documentation for module 're'.

- `use_fetch_block`

  / *Condition*: optional / *Type*: bool / *Default*: False /

  Determine if 'fetch block' feature is used.

- `end_of_block_pattern`

  / *Condition*: optional / *Type*: str / *Default*: '.*' /

  The end of block pattern.

- `filter_pattern`

  / *Condition*: optional / *Type*: str / *Default*: '.*' /

  Pattern to filter message line by line.

- `timeout`

  / *Condition*: optional / *Type*: int / *Default*: 0 /

  Timeout parameter specified as a floating point number in the unit 'seconds'.

- `element_def`

  / *Condition*: required / *Type*: dict /

  User's definition for a GUI element.

- `args`

  / *Condition*: optional / *Type*: tuple /

  Optional arguments.

Returns:

- `match`

  / *Type*: re.Match /

  If no return value matched to the specified regular expression and a timeout occurred, return None.

  If a return value has matched to the specified regular expression, a match object is returned as the result.The complete trace message can be accessed by the 'string' attribute of the match object. For access to groups within the regular expression, use the group() method. For more information, refer to Python documentation for module 're'.

4.3.5 Method: disconnect

Abstract method for disconnecting connection.

Arguments:

- `_device`

  / *Condition*: required / *Type*: str /

  Unused.

Returns:

(*no returns*)

4.3.6 Method: quit

Quiting the connection.

Returns:

(*no returns*)

Chapter 5

Appendix

About this package:

Table 5.1: Package setup

Setup parameter	Value
Name	QConnectWinapp
Version	1.0.0
Date	25.04.2023
Description	Robot Framework QConnect library extension for Winapp GUI testing
Package URL	robotframework-qconnect-winapp
Author	Nguyen Huynh Tri Cuong
Email	cuong.nguyenhuynhtri@vn.bosch.com
Language	Programming Language :: Python :: 3
License	License :: OSI Approved :: Apache Software License
OS	Operating System :: Microsoft :: Windows
Python required	>=3.0
Development status	Development Status :: 4 - Beta
Intended audience	Intended Audience :: Developers
Topic	Topic :: Software Development

Chapter 6

History

1.0.0	07/2022
Initial version	