

RobotFramework_DBus

v. 0.2.0

Nguyen Huynh Tri Cuong

23.02.2026

Contents

1	Introduction	1
1.1	Introduction	1
1.2	Key Features	1
2	Description	2
2.1	Prerequisites	2
2.2	Getting Started	2
2.3	Usage	2
2.3.1	connect	2
2.3.2	disconnect	3
2.3.3	set signal received handler	3
2.3.4	unset signal received handler	4
2.3.5	register signal	4
2.3.6	call dbus method	4
2.3.7	wait for signal	5
2.4	Remote testing	5
2.5	Example	5
2.5.1	Example 1 - Synchronized DBus Signal Waiting	6
2.5.2	Example 2 - Managing Multiple DBus Signals with 'Register Signal' Keyword	6
2.5.3	Example 3 - Non-blocking DBus Signal Handling with 'Set Signal Received Handler'	7
2.5.4	Example 4 - Invoking DBus Methods: Using 'Call Dbus Method' Keyword	8
2.5.5	Example 5 - Testing a Remote DBus Service: Setup and Steps	8
2.6	Configure Git and correct EOL handling	9
2.7	Feedback	9
2.8	About	9
2.8.1	Maintainers	9
2.8.2	Contributors	9
2.8.3	3rd Party Licenses	9
2.8.4	Used Encryption	9
2.8.5	License	9
3	__init__.py	11
4	priority_queue.py	12
5	register_keyword.py	13
5.1	robotframework-dbus-common-register-keyword-registerkeyword-get-kw-name	13

5.2	robotframework-dbus-common-register-keyword-registerkeyword-callback-func	13
6	scheduled_job.py	14
6.1	robotframework-dbus-common-scheduled-job-scheduledjob-stop	14
6.2	robotframework-dbus-common-scheduled-job-scheduledjob-run	14
7	thread_safe_dict.py	15
7.1	robotframework-dbus-common-thread-safe-dict-threadsafedict-clear	15
7.2	robotframework-dbus-common-thread-safe-dict-threadsafedict-pop	15
7.3	robotframework-dbus-common-thread-safe-dict-threadsafedict-popitem	15
7.4	robotframework-dbus-common-thread-safe-dict-threadsafedict-update	15
8	utils.py	17
8.1	robotframework-dbus-common-utils-singleton	17
8.2	robotframework-dbus-common-utils-dictto class	17
8.3	robotframework-dbus-common-utils-utils	17
8.3.1	robotframework-dbus-common-utils-utils-make-unique-token	17
8.3.2	robotframework-dbus-common-utils-utils-get-all-descendant-classes	17
8.3.3	robotframework-dbus-common-utils-utils-get-all-sub-classes	18
8.3.4	robotframework-dbus-common-utils-utils-caller-name	18
8.3.5	robotframework-dbus-common-utils-utils-load-library	18
8.3.6	robotframework-dbus-common-utils-utils-is-ascii-or-unicode	19
9	dbus_client_agent.py	20
9.1	robotframework-dbus-dbus-agent-dbus-client-agent-run-agent	20
9.2	robotframework-dbus-dbus-agent-dbus-client-agent-dbusclientexecutor	20
9.2.1	robotframework-dbus-dbus-agent-dbus-client-agent-dbusclientexecutor-connect	20
9.2.2	robotframework-dbus-dbus-agent-dbus-client-agent-dbusclientexecutor-disconnect	20
9.2.3	robotframework-dbus-dbus-agent-dbus-client-agent-dbusclientexecutor-quit	20
9.2.4	robotframework-dbus-dbus-agent-dbus-client-agent-dbusclientexecutor-get-monitoring-signal-payloads	21
9.2.5	robotframework-dbus-dbus-agent-dbus-client-agent-dbusclientexecutor-add-signal-to-captured-dict	21
9.2.6	robotframework-dbus-dbus-agent-dbus-client-agent-dbusclientexecutor-register-monitored-signal	21
9.2.7	robotframework-dbus-dbus-agent-dbus-client-agent-dbusclientexecutor-wait-for-signal	22
9.2.8	robotframework-dbus-dbus-agent-dbus-client-agent-dbusclientexecutor-call-dbus-method	22
9.3	robotframework-dbus-dbus-agent-dbus-client-agent-dbusclientagent	22
9.3.1	robotframework-dbus-dbus-agent-dbus-client-agent-dbusclientagent-get-session-token	22
9.3.2	robotframework-dbus-dbus-agent-dbus-client-agent-dbusclientagent-initialize-dbus-client	23
9.3.3	robotframework-dbus-dbus-agent-dbus-client-agent-dbusclientagent-connect	23
9.3.4	robotframework-dbus-dbus-agent-dbus-client-agent-dbusclientagent-disconnect	23
9.3.5	robotframework-dbus-dbus-agent-dbus-client-agent-dbusclientagent-quit	23
9.3.6	robotframework-dbus-dbus-agent-dbus-client-agent-dbusclientagent-get-monitoring-signal-payloads	23
9.3.7	robotframework-dbus-dbus-agent-dbus-client-agent-dbusclientagent-register-monitored-signal	24
9.3.8	robotframework-dbus-dbus-agent-dbus-client-agent-dbusclientagent-wait-for-signal	24
9.3.9	robotframework-dbus-dbus-agent-dbus-client-agent-dbusclientagent-call-dbus-method	25

10	dbus_client.py	26
10.1	robotframework-dbus-dbus-client-dbusclient-connect	26
10.2	robotframework-dbus-dbus-client-dbusclient-disconnect	26
10.3	robotframework-dbus-dbus-client-dbusclient-quit	26
10.4	robotframework-dbus-dbus-client-dbusclient-set-signal-received-handler	26
10.5	robotframework-dbus-dbus-client-dbusclient-unset-signal-received-handler	27
10.6	robotframework-dbus-dbus-client-dbusclient-add-signal-to-captured-dict	27
10.7	robotframework-dbus-dbus-client-dbusclient-register-monitored-signal	27
10.8	robotframework-dbus-dbus-client-dbusclient-wait-for-signal	28
10.9	robotframework-dbus-dbus-client-dbusclient-call-dbus-method	28
10.10	robotframework-dbus-dbus-client-dbusclient-call-dbus-method-with-keyword-args	28
11	dbus_client_remote.py	29
11.1	robotframework-dbus-dbus-client-remote-dbusclientremote-connect	29
11.2	robotframework-dbus-dbus-client-remote-dbusclientremote-disconnect	29
11.3	robotframework-dbus-dbus-client-remote-dbusclientremote-quit	29
11.4	robotframework-dbus-dbus-client-remote-dbusclientremote-do-signal-check	29
11.5	robotframework-dbus-dbus-client-remote-dbusclientremote-set-signal-received-handler	30
11.6	robotframework-dbus-dbus-client-remote-dbusclientremote-unset-signal-received-handler	30
11.7	robotframework-dbus-dbus-client-remote-dbusclientremote-register-monitored-signal	30
11.8	robotframework-dbus-dbus-client-remote-dbusclientremote-wait-for-signal	30
11.9	robotframework-dbus-dbus-client-remote-dbusclientremote-call-dbus-method	31
12	dbus_manager.py	32
12.1	robotframework-dbus-dbus-manager-dbusmanager-quit	32
12.2	robotframework-dbus-dbus-manager-dbusmanager-add-connection	32
12.3	robotframework-dbus-dbus-manager-dbusmanager-remove-connection	32
12.4	robotframework-dbus-dbus-manager-dbusmanager-get-connection-by-name	33
12.5	robotframework-dbus-dbus-manager-dbusmanager-disconnect	33
12.6	robotframework-dbus-dbus-manager-dbusmanager-connect	33
12.7	robotframework-dbus-dbus-manager-dbusmanager-set-signal-received-handler	34
12.8	robotframework-dbus-dbus-manager-dbusmanager-unset-signal-received-handler	34
12.9	robotframework-dbus-dbus-manager-dbusmanager-register-signal	35
12.10	robotframework-dbus-dbus-manager-dbusmanager-call-dbus-method	35
12.11	robotframework-dbus-dbus-manager-dbusmanager-wait-for-signal	35
13	Appendix	37
14	History	38

Chapter 1

Introduction

1.1 Introduction

RobotFramework_DBus is a Python extension library for Robot Framework. It is designed to support the automation testing of DBus services. The library provides a set of keywords that enable testing of DBus methods and signals using both synchronous and asynchronous mechanisms.

Built upon the Python *dbus* library, **RobotFramework_DBus** simplifies the process of interacting with DBus services and facilitates efficient testing of their functionality. It offers a comprehensive set of features to streamline the testing process and ensure the reliability of DBus-based applications.

1.2 Key Features

- Integration with Robot Framework: **RobotFramework_DBus** seamlessly integrates with Robot Framework, allowing you to leverage its powerful test automation capabilities for DBus services.
- DBus Service Automation: With **RobotFramework_DBus**, you can easily automate the testing of DBus services. It provides a range of keywords specifically designed for this purpose.
- Testing DBus Methods and Signals: The library enables you to test both DBus methods and signals. You can validate the behavior of DBus methods and monitor the emission of signals during the testing process.
- Synchronous and Asynchronous Testing: **RobotFramework_DBus** supports both synchronous and asynchronous testing approaches. You can choose the appropriate mechanism based on your testing requirements.

Whether you are testing a single DBus service or a complex system of interconnected services, **RobotFramework_DBus** offers a reliable and efficient solution for DBus automation testing. It empowers you to ensure the quality and robustness of your DBus-based applications.

Chapter 2

Description

2.1 Prerequisites

Before using **RobotFramework_DBus**, make sure you have the following libraries installed:

- **pycairo**: This library provides Python bindings for the Cairo graphics library. It is used for rendering graphics and creating visual elements in your applications.
- **PyGObject**: PyGObject is a Python package that provides bindings for the GObject library. It allows you to use GObject-based libraries, such as GTK+, in Python applications.
- **dasbus**: **RobotFramework_DBus** is built upon the *dasbus* library, which is a Pythonic D-Bus library. Make sure you have *dasbus* installed before using **RobotFramework_DBus**.
- **pyinstaller**: PyInstaller is a tool used to package Python applications into standalone executables. If you plan to distribute your application as an executable, you will need to have *pyinstaller* installed.

Ensure that the above libraries are installed and properly configured in your Python environment before using **robotframework-dbus**. This will ensure the smooth functioning and compatibility of the library with your system.

2.2 Getting Started

You can checkout all [robotframework-dbus](#) sourcecode from the GitHub.

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

```
python setup.py install
```

2.3 Usage

RobotFramework_DBus Library support following keywords for testing connection in RobotFramework.

2.3.1 connect

Use for establishing a connection.

Syntax:

```
connect          conn_name=[connection name]
...              namespace=[namespace]
...              object_path=[object path]
...              mode=[test mode]
...              host=[remote host]
...              port=[remote port]
```

Arguments:

conn_name: The name or identifier of the connection instance used to interact with the DBus service. This parameter is optional and can be used to uniquely identify a specific connection when multiple connections are established. If not provided, a default connection will be used. Default is 'default_conn'.

namespace: The namespace of the DBus service. This identifies the specific service or group of services. It is used to differentiate between different service instances. The namespace should be a string that uniquely identifies the service.

E.g. namespace=org.example.HelloWorld

object_path: The object path of the DBus service. This identifies the specific object within the service that the action will be performed on. The object path should be a string that follows the DBus object path naming convention. It typically consists of a hierarchical structure separated by slashes (/).

E.g. object_path=/org/example/HelloWorld

mode: The mode of testing the DBus service. Possible values are 'local' or 'remote'. 'local' indicates testing on the current system, while 'remote' indicates testing on a remote system. Default is 'local'.

host: The IP address or hostname of the remote system where the DBus agent is running. This parameter is applicable only if 'mode' is set to 'remote'.

Default is 'localhost'.

2.3.2 disconnect

Use for disconnecting from the DBus service by connection name.

Syntax:

```
disconnect          conn_name
```

Arguments:

conn_name: The name or identifier of the connection instance to disconnect from. This parameter is optional and can be used to specify a specific connection to disconnect. If the connection name is 'ALL', all connections will be disconnected.

2.3.3 set signal received handler

Use to set a signal received handler for a specific DBus connection and signal.

Syntax:

```
set signal received handler    conn name=[conn name]
...                           signal=[signal name]
...                           handler=[keyword handles signal emitted event]
```

Arguments:

conn_name: The name or identifier of the connection instance used to interact with the DBus service. This parameter is optional and can be used to uniquely identify a specific connection when multiple connections are established. If not provided, a default connection will be used. Default is 'default_conn'.

signal: The name of the DBus signal to be set emitted handler.

handler: The robotframework keyword to handle the received signal. The handler should accept the necessary parameters based on the signal being handled.

2.3.4 unset signal received handler

Use to unset a signal received handler for a specific signal.

Syntax:

```
unset signal received handler    conn name=[conn name]
...                             signal=[signal name]
...                             handler=[keyword handles signal emitted event]
```

Arguments:

conn_name: The name or identifier of the connection instance used to interact with the DBus service. This parameter is optional and can be used to uniquely identify a specific connection when multiple connections are established. If not provided, a default connection will be used. Default is 'default_conn'.

signal: The name of the DBus signal to be unset emitted handler.

handler: The robotframework keyword which is handling the signal emitted event.

2.3.5 register signal

Use to register a DBus signal or signals to be monitored for a specific connection.

Syntax:

```
register signal                  conn name=[conn name]
...                             signal=[signal name]
```

Arguments:

conn_name: The name or identifier of the connection instance used to interact with the DBus service. This parameter is optional and can be used to uniquely identify a specific connection when multiple connections are established. If not provided, a default connection will be used. Default is 'default_conn'.

signal: The name of the DBus signal(s) to register. It can be a single signal name as a string, or multiple signal names joined by ','. For example: "signal1,signal2,signal3".

2.3.6 call dbus method

Use to call a DBus method with the specified method name and input arguments.

Syntax:

```
call dbus method                [conn name]    [method name]    [args]
```

Arguments:

conn_name: The name or identifier of the connection instance used to interact with the DBus service. This parameter is optional and can be used to uniquely identify a specific connection when multiple connections are established. If not provided, a default connection will be used. Default is 'default_conn'.

method_name: The name of the DBus method to be called.

args: Input arguments to be passed to the method.

Return value:

Return from called method.

2.3.7 wait for signal

Use to wait for a specific DBus signal to be received within a specified timeout period.

Syntax:

```
wait for signal      conn name=[conn name]
...                  signal=[signal name]
...                  timeout=[timeout]
```

Arguments:

conn_name: The name or identifier of the connection instance used to interact with the DBus service. This parameter is optional and can be used to uniquely identify a specific connection when multiple connections are established. If not provided, a default connection will be used. Default is 'default_conn'.

signal: The name of the DBus signal to wait for.

timeout: The maximum time (in seconds) to wait for the signal.

Return value:

The signal payloads.

2.4 Remote testing

After installing the library, you can find the DBus Client Agent at `/opt/rfwaio/python39/install/bin/`.

For remote testing, follow these steps:

1. Start the DBus Agent on the remote system by the following command:

```
dbus_client_agent [-h] [-host HOST] [-port PORT]
```

The DBus Client Agent supports the following command-line arguments:

`-host` (str, optional) The host where the agent is running. Default is `0.0.0.0`.

`-port` (int, optional) The port where the agent is listening. Default is `2507`.

2. On the host test PC, using the `connect` keyword with the `remote` mode and specify the correct host using the `host` parameter.
3. Use the other keywords in the same way as local testing.

2.5 Example

Kindly be advised that all the examples presented within this session are designed to interact with the DBus service sample `server.py` residing in the `atest/` directory.

2.5.1 Example 1 - Synchronized DBus Signal Waiting

Scenario: In this example, I will use the `wait for signal` keyword from the `RobotFramework_DBus` library to wait for a DBus signal to be emitted using the synchronized mechanism. This means that the keyword following `wait for signal` will only be executed after the signal is emitted, and the `wait for signal` keyword will return a value.

```
*** Settings ***
Library      RobotFramework_DBus.DBusManager

*** Test Cases ***
Hello World
    connect    conn_name=test_dbus
    ...        namespace=org.example.HelloWorld
    ...        mode=local

    ${ret}=    Wait For Signal    conn_name=test_dbus
    ...        signal=YellowMessage
    ...        timeout=10

    Log To Console    ${ret}

    Disconnect    test_dbus
```

Explanation: In the aforementioned example, we establish a connection to the DBus service `org.example.HelloWorld` using the `'connect'` keyword and name this connection `'test_dbus'`. Subsequently, the `'Wait for signal'` keyword will wait for the DBus service to emit the **YellowMessage** signal within a 10-second timeframe and return the payloads of this signal immediately upon its emission within the specified timeframe (failing if the timeout is exceeded). The content of the payloads will be printed to the console.

2.5.2 Example 2 - Managing Multiple DBus Signals with 'Register Signal' Keyword

Scenario: In Example 1, if we include an additional `Wait for signal` keyword to wait for the **GreenMessage** signal to be emitted after waiting for **YellowMessage**, there is a possibility of missing the occurrence of **GreenMessage** if it is emitted before **YellowMessage**. In such a scenario, we can utilize the `Register Signal` keyword to register **GreenMessage** in the watchlist. This ensures continuous monitoring of the signal starting immediately after executing this keyword and prevents the occurrence of missing the emitted **GreenMessage** event.

```
*** Settings ***
Library      RobotFramework_DBus.DBusManager

*** Test Cases ***
Hello World
    connect    conn_name=test_dbus
    ...        namespace=org.example.HelloWorld
    ...        mode=local

    Register Signal    conn_name=test_dbus    signal=GreenMessage

    ${ret}=    Wait For Signal    conn_name=test_dbus
    ...        signal=YellowMessage
    ...        timeout=10
    Log To Console    ${ret}

    ${ret}=    Wait For Signal    conn_name=test_dbus
    ...        signal=GreenMessage
    ...        timeout=10
    Log To Console    ${ret}

    ${ret}=    Call Dbus Method    test_dbus    Hello    World
    Disconnect    test_dbus
```

Explanation:

In the above example, by using the **Register Signal** keyword, we have registered the **GreenMessage** signal in the watchlist. If the **GreenMessage** event is emitted during the **Wait for signal YellowMessage** timeframe, it will be

recorded. Then, when we execute the `Wait for signal` keyword for **GreenMessage**, it will immediately return because the **GreenMessage** event has already occurred.

2.5.3 Example 3 - Non-blocking Dbus Signal Handling with 'Set Signal Received Handler'

Scenario: In the two examples above, the testcase is blocked at the `Wait for Signal` keyword until the signal event is emitted. In the next example, we will explore the usage of the `Set Signal Received Handler` keyword. This is a non-blocking keyword that allows us to register a user-defined keyword, such as a callback function. The registered keyword will be called when the emitted signal event occurs. The testcase will continue with the next keyword immediately after the `Set Signal Received Handler` keyword, without waiting.

```
*** Settings ***
Library      RobotFramework_DBus.DBusManager
Library      test/dbus/client.py

*** Test Cases ***
Hello World
    connect      conn_name=test_dbus
    ...          namespace=org.example.HelloWorld
    ...          mode=local

    Set Signal Received Handler      conn_name=test_dbus
    ...                             signal=RedMessage
    ...                             handler=On Received Red Signal

    Log To Console      The test is continuing...

    Sleep      10s

    Disconnect      test_dbus

*** Keyword ***
On Received Red Signal
    [Arguments]      ${arg1}=default 1
    log to console      Client received red signal. Payload: ${arg1}
    Unset Signal Received Handler      conn_name=test_dbus      signal=RedMessage      handler=↩
    ↪ On Received Red Signal
```

Explanation:

In the above example, by using the `Set Signal Received Handler` keyword, we have registered the user-defined keyword `On Received Red Signal` as a callback function. When the **RedMessage** is emitted, this keyword will be executed. During the time when the **RedMessage** has not been emitted yet, the testcase will continue running the subsequent keywords as usual. It is important to note that the user-defined keyword should have an argument to handle the payload of the signal. If there is no longer a need to invoke the callback function, we should unregister the signal handler using the `Unset Signal Received Handler` keyword.

2.5.4 Example 4 - Invoking DBus Methods: Using 'Call Dbus Method' Keyword

Scenario:

In the next example, we will explore how to use the `Call Dbus Method` keyword to invoke a method in the DBus service and retrieve the return value from this method.

```
*** Settings ***
Library      RobotFramework_DBus.DBusManager
Library      test/dbus/client.py

*** Test Cases ***
Hello World
    connect    conn_name=test_dbus
    ...        namespace=org.example.HelloWorld
    ...        mode=local

    ${ret}=    Call Dbus Method    test_dbus    Hello    World
    Log To Console    ${ret}

    Disconnect    test_dbus
```

2.5.5 Example 5 - Testing a Remote DBus Service: Setup and Steps

Scenario: In the next example, we will explore how to test a DBus service on a different PC. To be able to test on a remote system, we need to perform two steps:

1. On the System Under Test (SUT), run the following command:

dbus_client_agent 0.0.0.0 2507

The DBus Client Agent supports the following command-line arguments:

- host (str, optional) The host where the agent is running. Default is 0.0.0.0.
- port (int, optional) The port where the agent is listening. Default is 2507.

2. On the test PC, make slight modifications to the keyword's connect parameters as follows:

```
*** Settings ***
Library      RobotFramework_DBus.DBusManager
Library      test/dbus/client.py

*** Test Cases ***
Hello World
    connect    conn_name=test_dbus
    ...        namespace=org.example.HelloWorld
    ...        mode=remote
    ...        host=172.17.0.2
    ...        port=2507

    Set Signal Received Handler    conn_name=test_dbus
    ...                            signal=RedMessage
    ...                            handler=On Received Red Signal

    Log To Console    The test is continuing...

    Sleep    10s

    Disconnect    test_dbus

*** Keyword ***
On Received Red Signal
    [Arguments]    ${arg1}=default 1
    log to console    Client received red signal. Payload: ${arg1}
    Unset Signal Received Handler    conn_name=test_dbus    signal=RedMessage    handler=↵
    ↵ On Received Red Signal
```

Explanation:

In the above example, we set the value of the mode parameter for the connect keyword to remote, and add two parameters: host and port. The host parameter is set to the IP address of the System Under Test (SUT), which is 172.17.0.2, and the port parameter is set to 2507, which is the port that the agent is listening on.

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, then you can assure that all EOL are managed by git correctly as defined.

2.7 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.nguyenhuyhtri@vn.bosch.com.

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

2.8 About

2.8.1 Maintainers

[Nguyen Huynh Tri Cuong](#)

2.8.2 Contributors

[Nguyen Huynh Tri Cuong](#)

[Thomas Pollerspoeck](#)

2.8.3 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

2.8.4 Used Encryption

No encryption is used in this project.

2.8.5 License

This project is licensed under the Apache License, Version 2.0.

Copyright 2020-2026 Robert Bosch GmbH

Licensed under the Apache License, Version 2.0 (the “License”); you may not use this file except in compliance with the License. You may obtain a copy of the License at

<http://www.apache.org/licenses/LICENSE-2.0>

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an “AS IS” BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

Chapter 3

`__init__.py`

Class to manage all dbus communications.

Chapter 4

`priority_queue.py`

Chapter 5

register_keyword.py

A class that provides a keyword as a callback function for a DBus signal received.

5.1 robotframework-dbus-common-register-keyword-registerkeyword-get-kw-name

Get the handle keyword name.

Returns:

/ Type: str /

The handle keyword name.

5.2 robotframework-dbus-common-register-keyword-registerkeyword-callback-func

Constructor for RegisterKeyword class.

Arguments:

- `observer`

/ Condition: optional / Type: tuple / Default: None /

Input arguments to be passed to the callback method.

Returns:

(no returns)

Chapter 6

scheduled_job.py

A threaded job that executes a function at a specified interval.

6.1 robotframework-dbus-common-scheduled-job-scheduledjob-stop

Stop the execution of the job.

Returns:

(no returns)

6.2 robotframework-dbus-common-scheduled-job-scheduledjob-run

Start the job execution loop.

Returns:

(no returns)

Chapter 7

`thread_safe_dict.py`

This class provides a dictionary with thread-safe operations by utilizing locks for synchronized access.

7.1 `robotframework-dbus-common-thread-safe-dict-threadsafedict-clear`

Remove all key-value pairs from the dictionary.

Returns:

(no returns)

7.2 `robotframework-dbus-common-thread-safe-dict-threadsafedict-pop`

Remove and return the value associated with a given key.

Arguments:

- `key`
/ *Condition*: required / *Type*: Any /
The key of dictionary to be pop.

Returns:

- `o`
/ *Type*: Any /
Value of the given key.

7.3 `robotframework-dbus-common-thread-safe-dict-threadsafedict-popitem`

Remove and return an arbitrary key-value pair from the dictionary.

Returns:

(no returns)

7.4 `robotframework-dbus-common-thread-safe-dict-threadsafedict-update`

Update the dictionary with key-value pairs from another dictionary.

Arguments:

- `dict`
/ *Condition*: optional / *Type*: dict / *Default*: None /
Another dictionary-like object to update from.

Returns:

(no returns)

Chapter 8

utils.py

8.1 robotframework-dbus-common-utils-singleton

Class to implement Singleton Design Pattern. This class is used to derive the DBusManager as only a single instance of this class is allowed.

8.2 robotframework-dbus-common-utils-dicttoclass

Class for converting dictionary to class object.

8.3 robotframework-dbus-common-utils-utils

Class to implement utilities for supporting development.

8.3.1 robotframework-dbus-common-utils-utils-make-unique-token

Generates a unique session token of specified length.

The *make_unique_token* function generates a unique session token of the specified length. The session token can be used to identify and associate a session with a specific client.

The session token is a string value that is guaranteed to be unique for each invocation of this function. It can be used as a secure identifier to track and manage client sessions within the DBusAgent.

Arguments:

- `length`
/ *Condition*: optional / *Type*: int / *Default*: 16 /
The length of the session token. Defaults to 16.

Returns:

- `token`
/ *Type*: str /
A unique token.

8.3.2 robotframework-dbus-common-utils-utils-get-all-descendant-classes

Get all descendant classes of a class

Arguments:

- `cls`
/ *Condition*: required / *Type*: class /
Input class for finding children.

Returns:

/ *Type*: list /
 Array of descendant classes.

8.3.3 robotframework-dbus-common-utils-utils-get-all-sub-classes

Get all children classes of a class

Arguments:

- `cls`
 / *Condition*: required / *Type*: class /
 Input class for finding children.

Returns:

/ *Type*: list /
 Array of children classes.

8.3.4 robotframework-dbus-common-utils-utils-caller-name

Get a name of a caller in the format `module.class.method`

Arguments:

- `skip`
 / *Condition*: required / *Type*: int /

Specifies how many levels of stack to skip while getting caller name. `skip=1` means "who calls me", `skip=2` "who calls my caller" etc.

Returns:

/ *Type*: str /
 An empty string is returned if skipped levels exceed stack height

8.3.5 robotframework-dbus-common-utils-utils-load-library

Load native library depend on the calling convention.

Arguments:

- `path`
 / *Condition*: required / *Type*: str /
 Library path.
- `is_stdcall`
 / *Condition*: optional / *Type*: bool / *Default*: True /
 Determine if the library's calling convention is stdcall or cdecl.

Returns:

Loaded library object.

8.3.6 robotframework-dbus-common-utils-utils-is-ascii-or-unicode

Check if the string is ascii or unicode

Arguments: str_check: string for checking codecs: encoding type list

Returns:

/ *Type*: bool /

True : if checked string is ascii or unicode

False : if checked string is not ascii or unicode

Chapter 9

dbus_client_agent.py

9.1 robotframework-dbus-dbus-agent-dbus-client-agent-run-agent

Run the DBus Agent with the specified configuration.

Description:

The *run_agent* function starts the DBus Agent with the provided configuration. It parses the command-line arguments, such as the host and port options, and then starts the agent on the specified host and port.

The agent listens for incoming requests from clients and manages client sessions. It provides a mechanism to execute requests on corresponding DBus services through the assigned Executor instances.

Command-line Arguments:

--host (str, optional): The host where the agent is running. Default is '0.0.0.0'.

—port (int, optional): The port where the agent is listening. Default is 2507.

9.2 robotframework-dbus-dbus-agent-dbus-client-agent-dbusclientexecutor

The DBusClientExecutor class represents an executor responsible for handling client requests on specific DBus services. It receives requests from the DBusAgent and executes them on the corresponding DBus service.

9.2.1 robotframework-dbus-dbus-agent-dbus-client-agent-dbusclientexecutor-connect

Create a proxy object to DBus object.

Returns:

(no returns)

9.2.2 robotframework-dbus-dbus-agent-dbus-client-agent-dbusclientexecutor-disconnect

Disconnect the DBus proxy from the remote object.

Returns:

(no returns)

9.2.3 robotframework-dbus-dbus-agent-dbus-client-agent-dbusclientexecutor-quit

Quit the DBus client.

Returns:

(no returns)

9.2.4 robotframework-dbus-dbus-agent-dbus-client-agent-dbusclientexecutor-get-monitoring-signal-payloads

Get the payloads of a specific signal.

Arguments:

- `signal`
/ *Condition*: required / *Type*: str /
The name of the DBus signal to get payloads.

Returns:

- `payloads`
/ *Type*: str /
The signal's payloads.

9.2.5 robotframework-dbus-dbus-agent-dbus-client-agent-dbusclientexecutor-add-signal-to-captured-dict

Add a signal and its payloads to the captured dictionary when the signal be emitted.

Arguments:

- `signal`
/ *Condition*: required / *Type*: str /
The name of the DBus signal(s) which has been raised.
- `loop`
/ *Condition*: optional / *Type*: EventLoop / *Default*: None /
The Event loop which is running to wait for the raised signal.
- `payloads`
/ *Condition*: optional / *Type*: Any / *Default*: "" /
The payloads of the raised signal.

Returns:

(no returns)

9.2.6 robotframework-dbus-dbus-agent-dbus-client-agent-dbusclientexecutor-register-monitored-signal

Register a DBus signal or signals to be monitored for a specific connection.

Arguments:

- `signal`
/ *Condition*: optional / *Type*: str / *Default*: " /
The name of the DBus signal(s) to register. It can be a single signal name as a string, or multiple signal names joined by ','. For example: "signal1,signal2,signal3".

Returns:

(no returns)

9.2.7 robotframework-dbus-dbus-agent-dbus-client-agent-dbusclientexecutor-wait-for-signal

Wait for a specific DBus signal to be received within a specified timeout period.

Arguments:

- `wait_signal`
/ *Condition*: optional / *Type*: str / *Default*: " /
The name of the DBus signal to wait for.
- `timeout`
/ *Condition*: optional / *Type*: int / *Default*: 0 /
The maximum time (in seconds) to wait for the signal.

Returns:

- `payloads`
/ *Type*: str /
The signal payloads.

9.2.8 robotframework-dbus-dbus-agent-dbus-client-agent-dbusclientexecutor-call-dbus-method

Call a DBus method with the specified method name and input arguments.

Arguments:

- `method_name`
/ *Condition*: optional / *Type*: str / *Default*: " /
The name of the DBus method to be called.
- `args`
/ *Condition*: optional / *Type*: tuple / *Default*: None /
Input arguments to be passed to the method.

Returns:

- `ret`
/ *Type*: Any /
Return from called method.

9.3 robotframework-dbus-dbus-agent-dbus-client-agent-dbusclientagent

The `DBusClientAgent` class acts as a mediator between clients and the corresponding DBus services they request. It manages client connections, session tokens, and assigns the appropriate `DBusClientExecutor` for each client session.

9.3.1 robotframework-dbus-dbus-agent-dbus-client-agent-dbusclientagent-get-session-token

Generates a unique session token for a client and returns it.

Returns:

/ *Type*: str /

The random and unique token.

9.3.2 robotframework-dbus-dbus-agent-dbus-client-agent-dbusclientagent-initialize-dbus-client

Initializes an DBusClientExecutor instance for a specific client.

An DBusClientExecutor is an object responsible for executing requests from clients on corresponding DBus services. By initializing a separate DBusClientExecutor for each client, the DBusClientAgent ensures that requests from different clients are handled independently.

The client session token is a unique identifier that can be used to associate the DBusClientExecutor with the specific client. This allows the DBusClientAgent to route incoming requests to the correct DBusClientExecutor based on the client session token.

Arguments:

- `session`

/ Condition: required / Type: str /

The client's session token.

- `namespace`

/ Condition: optional / Type: str / Default: " /

The namespace of the DBus service. This identifies the specific service or group of services. It is used to differentiate between different service instances. The namespace should be a string that uniquely identifies the service.

- `object_path`

/ Condition: optional / Type: str / Default: None /

The object path of the DBus service. This identifies the specific object within the service that the action will be performed on. The object path should be a string that follows the DBus object path naming convention. It typically consists of a hierarchical structure separated by slashes (/).

Returns:

(no returns)

9.3.3 robotframework-dbus-dbus-agent-dbus-client-agent-dbusclientagent-connect

Create a proxy object to DBus service.

Returns:

(no returns)

9.3.4 robotframework-dbus-dbus-agent-dbus-client-agent-dbusclientagent-disconnect

Disconnect the DBus proxy from the remote object.

Returns:

(no returns)

9.3.5 robotframework-dbus-dbus-agent-dbus-client-agent-dbusclientagent-quit

Quit the DBus client.

Returns:

(no returns)

9.3.6 robotframework-dbus-dbus-agent-dbus-client-agent-dbusclientagent-get-monitoring-signal-payloads

Get the payloads of a specific signal.

Arguments:

- `session`
/ *Condition*: required / *Type*: str /
The client's session token.
- `signal`
/ *Condition*: required / *Type*: str /
The name of the DBus signal to get payloads.

Returns:

- `payloads`
/ *Type*: str /
The signal's payloads.

9.3.7 robotframework-dbus-dbus-agent-dbus-client-agent-dbusclientagent-register-monitored-signal

Register a DBus signal or signals to be monitored for a specific connection.

Arguments:

- `session`
/ *Condition*: required / *Type*: str /
The client's session token.
- `signal`
/ *Condition*: required / *Type*: str /
The name of the DBus signal(s) to register. It can be a single signal name as a string, or multiple signal names joined by ','. For example: "signal1,signal2,signal3".

Returns:

(no returns)

9.3.8 robotframework-dbus-dbus-agent-dbus-client-agent-dbusclientagent-wait-for-signal

Wait for a specific DBus signal to be received within a specified timeout period.

Arguments:

- `session`
/ *Condition*: required / *Type*: str /
The client's session token.
- `wait_signal`
/ *Condition*: optional / *Type*: str / *Default*: "" /
The name of the DBus signal to wait for.
- `timeout`
/ *Condition*: optional / *Type*: int / *Default*: 0 /
The maximum time (in seconds) to wait for the signal.

Returns:

- `payloads`
/ *Type*: str /
The signal payloads.

9.3.9 robotframework-dbus-dbus-agent-dbus-client-agent-dbusclientagent-call-dbus-method

Call a DBus method with the specified method name and input arguments.

Arguments:

- `session`
/ *Condition*: required / *Type*: str /
The client's session token.
- `method_name`
/ *Condition*: optional / *Type*: str / *Default*: " /
The name of the DBus method to be called.
- `args`
/ *Condition*: optional / *Type*: tuple / *Default*: None /
Input arguments to be passed to the method.

Returns:

/ *Type*: Any /
Return from called method.

Chapter 10

dbus_client.py

A client class for interacting with a specific DBus service.

10.1 robotframework-dbus-dbus-client-dbusclient-connect

Create a proxy object to DBus object.

Returns:

(no returns)

10.2 robotframework-dbus-dbus-client-dbusclient-disconnect

Disconnect the DBus proxy from the remote object.

Returns:

(no returns)

10.3 robotframework-dbus-dbus-client-dbusclient-quit

Quit the DBus client.

Returns:

(no returns)

10.4 robotframework-dbus-dbus-client-dbusclient-set-signal-received-handler

Set a signal received handler for a specific signal.

Arguments:

- `signal`
/ *Condition*: required / *Type*: str /
The name of the DBus signal to handle.
- `handler`
/ *Condition*: required / *Type*: str /
The keyword to handle the received signal. The handler should accept the necessary parameters based on the signal being handled.

Returns:

(no returns)

10.5 robotframework-dbus-dbus-client-dbusclient-unset-signal-received-handler

Unset a signal received handler for a specific signal.

Arguments:

- `signal`
/ *Condition*: required / *Type*: str /
The name of the DBus signal to handle.
- `handle_keyword`
/ *Condition*: optional / *Type*: str / *Type*: None /
The keyword which is handling for signal emitted event.

Returns:

(no returns)

10.6 robotframework-dbus-dbus-client-dbusclient-add-signal-to-captured-dict

Add a signal and its payloads to the captured dictionary when the signal be emitted.

Arguments:

- `signal`
/ *Condition*: required / *Type*: str /
The name of the DBus signal(s) which has been raised.
- `loop`
/ *Condition*: optional / *Type*: EventLoop / *Default*: None /
The Event loop which is running to wait for the raised signal.
- `payloads`
/ *Condition*: optional / *Type*: Any / *Default*: "" /
The payloads of the raised signal.

Returns:

(no returns)

10.7 robotframework-dbus-dbus-client-dbusclient-register-monitored-signal

Register a DBus signal or signals to be monitored for a specific connection.

Arguments:

- `signal`
/ *Condition*: optional / *Type*: str / *Default*: "" /
The name of the DBus signal(s) to register. It can be a single signal name as a string, or multiple signal names joined by ','. For example: "signal1,signal2,signal3".

Returns:

(no returns)

10.8 robotframework-dbus-dbus-client-dbusclient-wait-for-signal

Wait for a specific DBus signal to be received within a specified timeout period.

Arguments:

- `wait_signal`
/ *Condition*: optional / *Type*: str / *Default*: " /
The name of the DBus signal to wait for.
- `timeout`
/ *Condition*: optional / *Type*: int / *Default*: 0 /
The maximum time (in seconds) to wait for the signal.

Returns:

- `payloads`
/ *Type*: str /
The signal payloads.

10.9 robotframework-dbus-dbus-client-dbusclient-call-dbus-method

Call a DBus method with the specified method name and input arguments.

Arguments:

- `method_name`
/ *Condition*: optional / *Type*: str / *Default*: " /
The name of the DBus method to be called.
- `args`
/ *Condition*: optional / *Type*: tuple / *Default*: None /
Input arguments to be passed to the method.

Returns:

/ *Type*: Any /
Return from called method.

10.10 robotframework-dbus-dbus-client-dbusclient-call-dbus-method-with-keyword-args

Call a DBus method with the specified method name and input arguments.

Arguments:

- `method_name`
/ *Condition*: optional / *Type*: str / *Default*: " /
The name of the DBus method to be called.
- `args`
/ *Condition*: optional / *Type*: tuple / *Default*: None /
Input arguments to be passed to the method.

Returns:

- `ret_obj`
/ *Type*: Any /
Connection object.

Chapter 11

dbus_client_remote.py

A client class for interacting with a specific DBus service on a remote machine.

11.1 robotframework-dbus-dbus-client-remote-dbusclientremote-connect

Create a proxy object to DBus object.

Returns:

(no returns)

11.2 robotframework-dbus-dbus-client-remote-dbusclientremote-disconnect

Disconnect the DBus proxy from the remote object.

Returns:

(no returns)

11.3 robotframework-dbus-dbus-client-remote-dbusclientremote-quit

Quit the DBus client.

Returns:

(no returns)

11.4 robotframework-dbus-dbus-client-remote-dbusclientremote-do-signal-check

Checking if the signal was emitted.

Arguments:

- `signal`
/ *Condition*: required / *Type*: str /
The name of the DBus signal to check.
- `call_back_func`
/ *Condition*: required / *Type*: callable /
The function to be callback when receiving the signal.

Returns:

(no returns)

11.5 robotframework-dbus-dbus-client-remote-dbusclientremote-set-signal-received-handler

Set a signal received handler for a specific signal.

Arguments:

- `signal`
/ *Condition*: required / *Type*: str /
The name of the DBus signal to handle.
- `handler`
/ *Condition*: required / *Type*: str /
The keyword to handle the received signal. The handler should accept the necessary parameters based on the signal being handled.

Returns:

(no returns)

11.6 robotframework-dbus-dbus-client-remote-dbusclientremote-unset-signal-received-handler

Unset a signal received handler for a specific signal.

Arguments:

- `signal`
/ *Condition*: required / *Type*: str /
The name of the DBus signal to handle.

Returns:

(no returns)

11.7 robotframework-dbus-dbus-client-remote-dbusclientremote-register-monitored-signal

Register a DBus signal or signals to be monitored for a specific connection.

Arguments:

- `signal`
/ *Condition*: optional / *Type*: str / *Default*: " /
The name of the DBus signal(s) to register. It can be a single signal name as a string, or multiple signal names joined by ','. For example: "signal1,signal2,signal3".

Returns:

(no returns)

11.8 robotframework-dbus-dbus-client-remote-dbusclientremote-wait-for-signal

Wait for a specific DBus signal to be received within a specified timeout period.

Arguments:

- `wait_signal`
/ *Condition*: optional / *Type*: str / *Default*: " /
The name of the DBus signal to wait for.
- `timeout`
/ *Condition*: optional / *Type*: int / *Default*: 0 /
The maximum time (in seconds) to wait for the signal.

Returns:

- `payloads`
/ *Type*: str /
The signal payloads.

11.9 robotframework-dbus-dbus-client-remote-dbusclientremote-call-dbus-method

Call a DBus method with the specified method name and input arguments.

Arguments:

- `method_name`
/ *Condition*: optional / *Type*: str / *Default*: " /
The name of the DBus method to be called.
- `args`
/ *Condition*: optional / *Type*: tuple / *Default*: None /
Input arguments to be passed to the method.

Returns:

- `ret_obj`
/ *Type*: Any /
Connection object.

Chapter 12

dbus_manager.py

Class to manage all Dbus connections.

12.1 robotframework-dbus-dbus-manager-dbusmanager-quit

Quit connection manager.

Returns:

(no returns)

12.2 robotframework-dbus-dbus-manager-dbusmanager-add-connection

Add a connection to managed dictionary.

Arguments:

- name
/ *Condition*: required / *Type*: str /
Connection's name.
- conn
/ *Condition*: required / *Type*: DbusClient /
Connection object.

Returns:

(no returns)

12.3 robotframework-dbus-dbus-manager-dbusmanager-remove-connection

Remove a connection by name.

Arguments:

- connection_name
/ *Condition*: required / *Type*: str /
Connection's name.

Returns:

(no returns)

12.4 robotframework-dbus-dbus-manager-dbusmanager-get-connection-by-name

Get an exist connection by name.

Arguments:

- `connection_name`
/ *Condition*: required / *Type*: str /
Connection's name.

Returns:

- `conn`
/ *Type*: socket.socket /
Connection object.

12.5 robotframework-dbus-dbus-manager-dbusmanager-disconnect

Keyword for disconnecting a connection by name.

Arguments:

- `connection_name`
/ *Condition*: required / *Type*: str /
Connection's name.

Returns:

(no returns)

12.6 robotframework-dbus-dbus-manager-dbusmanager-connect

Keyword used to establish a DBus connection.

Arguments:

- `conn_name`
/ *Condition*: optional / *Type*: str / *Default*: 'default_conn' /
The name or identifier of the connection instance used to interact with the DBus service. This parameter is optional and can be used to uniquely identify a specific connection when multiple connections are established. If not provided, a default connection will be used.
- `namespace`
/ *Condition*: optional / *Type*: str / *Default*: "" /
The namespace of the DBus service. This identifies the specific service or group of services. It is used to differentiate between different service instances. The namespace should be a string that uniquely identifies the service.
- `object_path`
/ *Condition*: optional / *Type*: str / *Default*: None /
The object path of the DBus service. This identifies the specific object within the service that the action will be performed on. The object path should be a string that follows the DBus object path naming convention. It typically consists of a hierarchical structure separated by slashes (/).
- `mode`
/ *Condition*: optional / *Type*: str / *Default*: 'local' /
The mode of testing the DBus service. Possible values are 'local' or 'remote'. 'local' indicates testing on the current system, while 'remote' indicates testing on a remote system.

- `host`
/ Condition: optional / Type: str / Default: 'localhost' /
 The IP address or hostname of the remote system where the DBus agent is running.
 This parameter is applicable only if *mode* is set to 'remote'.
- `port`
/ Condition: optional / Type: int / Default: 2507 /
 The port number on which the DBus agent is listening on the remote system.
 This parameter is applicable only if *mode* is set to 'remote'.

Returns:*(no returns)*

12.7 robotframework-dbus-dbus-manager-dbusmanager-set-signal-received-handler

Keyword used to set a signal received handler for a specific DBus connection and signal.

Arguments:

- `conn_name`
/ Condition: optional / Type: str / Default: 'default_conn' /
 The name of the DBus connection.
- `signal`
/ Condition: optional / Type: str / Default: "" /
 The name of the DBus signal to handle.
- `handler`
/ Condition: optional / Type: str / Default: None /
 The keyword to handle the received signal. The handler should accept the necessary parameters based on the signal being handled.

Returns:*(no returns)*

12.8 robotframework-dbus-dbus-manager-dbusmanager-unset-signal-received-handler

Keyword used to set a signal received handler for a specific DBus connection and signal.

Arguments:

- `conn_name`
/ Condition: optional / Type: str / Default: 'default_conn' /
 The name of the DBus connection.
- `signal`
/ Condition: optional / Type: str / Default: "" /
 The name of the DBus signal to handle.
- `handler`
/ Condition: optional / Type: str / Default: None /
 The robotframework keyword which is handling the signal emitted event.

Returns:*(no returns)*

12.9 robotframework-dbus-dbus-manager-dbusmanager-register-signal

Keyword used to register a DBus signal or signals to be monitored for a specific connection.

Arguments:

- `conn_name`
/ *Condition*: optional / *Type*: str / *Default*: 'default_conn' /
The name of the DBus connection.
- `signal`
/ *Condition*: optional / *Type*: str / *Default*: "" /
The name of the DBus signal(s) to register. It can be a single signal name as a string, or multiple signal names joined by ','. For example: "signal1,signal2,signal3".

Returns:

(no returns)

12.10 robotframework-dbus-dbus-manager-dbusmanager-call-dbus-method

Keyword used to call a DBus method with the specified method name and input arguments.

Arguments:

- `conn_name`
/ *Condition*: optional / *Type*: str / *Default*: 'default_conn' /
The name of the DBus connection.
- `method_name`
/ *Condition*: optional / *Type*: str / *Default*: "" /
The name of the DBus method to be called.
- `args`
/ *Condition*: optional / *Type*: tuple / *Default*: None /
Input arguments to be passed to the method.

Returns:

- `ret_obj`
/ *Type*: Any /
Return from called method.

12.11 robotframework-dbus-dbus-manager-dbusmanager-wait-for-signal

Keyword used to wait for a specific DBus signal to be received within a specified timeout period.

Arguments:

- `conn_name`
/ *Condition*: optional / *Type*: str / *Default*: 'default_conn' /
The name of the DBus connection.
- `signal`
/ *Condition*: optional / *Type*: str / *Default*: "" /
The name of the DBus signal to wait for.

- `timeout`
/ *Condition*: optional / *Type*: int / *Default*: 0 /
The maximum time (in seconds) to wait for the signal.

Returns:

- `payloads`
/ *Type*: str /
The signal payloads.

Chapter 13

Appendix

About robotframework-dbus:

Author

Nguyen Huynh Tri Cuong (cuong.nguyenuynhtri@vn.bosch.com)

Version

0.2.0 (23.02.2026)

Short Description

Robot Framework library for automated testing of Dbus services

Homepage

[robotframework-dbus](#)

Documentation

[Readme](#)

[Main Documentation](#)

Sources

[Repository](#)

Issues

[Issues](#)

Python required

≥ 3.11

License

Apache-2.0

Chapter 14

History

0.1.0	05/2023
<i>Initial version</i>	
0.2.0	23.02.2026
<i>Usage of Setuptools replaced by usage of PIP/TOML/Setuptools</i>	

RobotFramework_DBus.pdf*Created at 24.06.2026 - 06:06:59**by GenPackageDoc v. 0.44.0 / 05.06.2026*
