flexlogger

National Instruments

Dec 03, 2021

CONTENTS:

CHAPTER

ONE

GETTING STARTED

1. Create an instance of the *Application* class. There are two ways to do this:

- To launch a new FlexLogger application, call *Application.launch()*.
- To connect to an already-running FlexLogger application, use the standard initializer *Application()*. Note that before connecting to an already running instance of FlexLogger, the Automation server preference must be enabled. You can enable this preference by opening the "File>>Preferences" menu item, and then enabling the "Automation server" preference in the "General" tab.

2. Call *Application.open_project()* to open a project.

After this, here's how to do several common tasks:

- **Start and stop a test**: The *Project.test_session* property returns a *TestSession* object which has *start()* and *stop()* methods.
- **Configure test properties**: The *Project.open_logging_specification_document()* method returns a *LoggingSpecificationDocument* object which has *get_test_property()*, *set_test_property()*, and *remove_test_property()* methods. The *get_test_properties()* method returns the full *TestProperty* list for the project.
- **Configure the log file location**: The *Project.open_logging_specification_document()* method returns a *LoggingSpecificationDocument* object which has *set_log_file_base_path()* and *set_log_file_name()* methods.
- **Read and write channel values**: Calling *Project.open_channel_specification_document()* returns a *ChannelSpecificationDocument* object which has *get_channel_value()* and *set_channel_value()* methods.

CHAPTER

TWO

EXAMPLES

2.1 Communicating with FlexLogger

Launch FlexLogger and open a project

```python
import os
import sys

from flexlogger.automation import Application


def main(project_path):
    """Launch FlexLogger and open a project."""
    # Note that using the "with" statement here means that when the block
    # goes out of scope, the application will be closed.  To prevent this,
    # call app.disconnect() before the scope ends.
    with Application.launch() as app:
        project = app.open_project(path=project_path)
        print("Press Enter to close project...")
        input()
        project.close()
    return 0


if __name__ == "__main__":
    argv = sys.argv
    if len(argv) < 2:
        print("Usage: %s <path of project to open>" % os.path.basename(__file__))
        sys.exit()
    project_path_arg = argv[1]
    sys.exit(main(project_path_arg))
```

Connect to FlexLogger when it is already running

```python
import sys

from flexlogger.automation import Application


def main():
    """Connect to an already running instance of FlexLogger with an open project
```

(continues on next page)

(continued from previous page)

```
    Note that before connecting to an already running instance of FlexLogger,
    the Automation server preference must be enabled. You can enable this preference by opening the
    "File>>Preferences" menu item, and then enabling the "Automation server" preference in the
    "General" tab.
    """
    app = Application()
    project = app.get_active_project()
    if project is None:
        print("No project is open in FlexLogger!")
        return 1
    test_session_state = project.test_session.state
    print("The test session state is:")
    print(test_session_state)
    print("Press Enter to disconnect from FlexLogger...")
    input()
    app.disconnect()
    return 0


if __name__ == "__main__":
    sys.exit(main())
```

2.2 Test Session

Start and stop a test

```
import os
import sys

from flexlogger.automation import Application


def main(project_path):
    """Launch FlexLogger, open a project, start and stop the test session."""
    with Application.launch() as app:
        project = app.open_project(path=project_path)
        test_session = project.test_session
        test_session.start()
        print("Test started. Press Enter to stop the test and close the project...")
        input()
        test_session.stop()
        project.close()
    return 0


if __name__ == "__main__":
    argv = sys.argv
```

(continues on next page)

(continued from previous page)

```
    if len(argv) < 2:
        print("Usage: %s <path of project to open>" % os.path.basename(__file__))
        sys.exit()
    project_path_arg = argv[1]
    sys.exit(main(project_path_arg))
```

Adding a note to a log file

```
import os
import sys

from flexlogger.automation import Application


def main(project_path):
    """Launch FlexLogger, open a project, start the test session and add a note."""
    with Application.launch() as app:
        project = app.open_project(path=project_path)
        test_session = project.test_session
        test_session.start()
        note = input("Test started. Enter a note to log: ")
        test_session.add_note(note)
        print("Note added. Press Enter to stop the test and close the project...")
        input()

        test_session.stop()
        project.close()
    return 0


if __name__ == "__main__":
    argv = sys.argv
    if len(argv) < 2:
        print("Usage: %s <path of project to open>" % os.path.basename(__file__))
        sys.exit()
    project_path_arg = argv[1]
    sys.exit(main(project_path_arg))
```

2.3 Channels

Getting the value of a channel

```
import os
import sys

from flexlogger.automation import Application


def main(project_path):
    """Launch FlexLogger, open a project, and get the value of a channel."""
```

(continues on next page)

(continued from previous page)

```
    with Application.launch() as app:
        project = app.open_project(path=project_path)
        channel_name = input("Enter the name of the channel to get the value of: ")
        channel_specification = project.open_channel_specification_document()
        channel_value = channel_specification.get_channel_value(channel_name)
        print("Channel value:")
        print(channel_value)
        print("Press Enter to close the project...")
        input()
        project.close()
    return 0


if __name__ == "__main__":
    argv = sys.argv
    if len(argv) < 2:
        print("Usage: %s <path of project to open>" % os.path.basename(__file__))
        sys.exit()
    project_path_arg = argv[1]
    sys.exit(main(project_path_arg))
```

Setting the value of a channel

```
import os
import sys

from flexlogger.automation import Application


def main(project_path):
    """Launch FlexLogger, open a project, and sets the value of a channel."""
    with Application.launch() as app:
        project = app.open_project(path=project_path)
        channel_name = input("Enter the name of the channel to set the value of: ")
        channel_value = input("Enter the new channel value: ")
        channel_specification = project.open_channel_specification_document()
        channel_specification.set_channel_value(channel_name, float(channel_value))
        print("Channel value set. Press Enter to close the project...")
        input()
        project.close()
    return 0


if __name__ == "__main__":
    argv = sys.argv
    if len(argv) < 2:
        print("Usage: %s <path of project to open>" % os.path.basename(__file__))
        sys.exit()
    project_path_arg = argv[1]
    sys.exit(main(project_path_arg))
```

2.4 Logging

Getting the log file base path and name

```
import os
import sys

from flexlogger.automation import Application


def main(project_path):
    """Launch FlexLogger, open a project, and gets the log file base path and name."""
    with Application.launch() as app:
        project = app.open_project(path=project_path)
        logging_specification = project.open_logging_specification_document()
        log_file_base_path = logging_specification.get_log_file_base_path()
        log_file_name = logging_specification.get_log_file_name()
        print("Log file base path: " + log_file_base_path)
        print("Log file name: " + log_file_name)
        print("Press Enter to close the project...")
        input()
        project.close()
    return 0


if __name__ == "__main__":
    argv = sys.argv
    if len(argv) < 2:
        print("Usage: %s <path of project to open>" % os.path.basename(__file__))
        sys.exit()
    project_path_arg = argv[1]
    sys.exit(main(project_path_arg))
```

Setting the log file base path and name

```
import os
import sys

from flexlogger.automation import Application


def main(project_path):
    """Launch FlexLogger, open a project, and sets the log file base path and name."""
    with Application.launch() as app:
        project = app.open_project(path=project_path)
        log_file_base_path = input("Enter the log file base path: ")
        log_file_name = input("Enter the log file name: ")
        logging_specification = project.open_logging_specification_document()
        logging_specification.set_log_file_base_path(log_file_base_path)
        logging_specification.set_log_file_name(log_file_name)
        print("Log file base path and name set. Press Enter to close the project...")
        input()
        project.close()
```

(continues on next page)

(continued from previous page)

```
    return 0


if __name__ == "__main__":
    argv = sys.argv
    if len(argv) < 2:
        print("Usage: %s <path of project to open>" % os.path.basename(__file__))
        sys.exit()
    project_path_arg = argv[1]
    sys.exit(main(project_path_arg))
```

Getting a test property

```
import os
import sys

from flexlogger.automation import Application


def main(project_path):
    """Launch FlexLogger, open a project, and gets the value of a test property."""
    with Application.launch() as app:
        project = app.open_project(path=project_path)
        test_property_name = input("Enter the name of the test property to get the value␣
↪of: ")
        logging_specification = project.open_logging_specification_document()
        test_property = logging_specification.get_test_property(test_property_name)
        print("Test property:")
        print(test_property)
        print("Press Enter to close the project...")
        input()
        project.close()
    return 0


if __name__ == "__main__":
    argv = sys.argv
    if len(argv) < 2:
        print("Usage: %s <path of project to open>" % os.path.basename(__file__))
        sys.exit()
    project_path_arg = argv[1]
    sys.exit(main(project_path_arg))
```

Setting a test property

```
import os
import sys

from flexlogger.automation import Application


def main(project_path):
```

(continues on next page)

(continued from previous page)

```
    """Launch FlexLogger, open a project, and sets the value of a test property."""
    with Application.launch() as app:
        project = app.open_project(path=project_path)
        test_property_name = input("Enter the name of the test property to set the value␣
↪of: ")
        test_property_value = input("Enter the test property value: ")
        logging_specification = project.open_logging_specification_document()
        logging_specification.set_test_property(test_property_name, test_property_value)
        print("Test property set. Press Enter to close the project...")
        input()
        project.close()
    return 0


if __name__ == "__main__":
    argv = sys.argv
    if len(argv) < 2:
        print("Usage: %s <path of project to open>" % os.path.basename(__file__))
        sys.exit()
    project_path_arg = argv[1]
    sys.exit(main(project_path_arg))
```

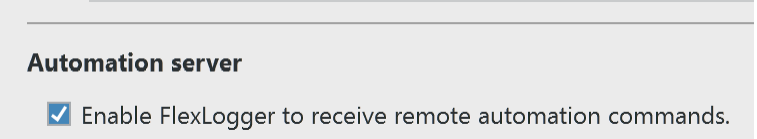
CHAPTER

THREE

TROUBLESHOOTING

3.1 Can't connect to running FlexLogger

If you get an error connecting to an already running instance of FlexLogger, the Automation server preference may not be enabled. You can enable this preference by opening the "File>>Preferences" menu item, and then enabling the "Automation server" preference in the "General" tab.



3.2 Exception on first call into FlexLogger

If you see an exception on the first call into FlexLogger similar to:

```
grpc._channel._InactiveRpcError: <_InactiveRpcError of RPC that terminated with:
      status = StatusCode.UNAVAILABLE
      details = "failed to connect to all addresses"
      debug_error_string = "{"created":"@1608052709.612000000","description":"Failed to␣
↪pick subchannel","file":"src/core/ext/filters/client_channel/client_channel.cc","file_
↪line":4143,"referenced_errors":[{"created":"@1608052633.077000000","description":
↪"failed to connect to all addresses","file":"src/core/ext/filters/client_channel/lb_
↪policy/pick_first/pick_first.cc","file_line":398,"grpc_status":14}]}"
```

the problem may be an HTTP proxy. To test this, in your Python script add the following lines before your FlexLogger API calls:

```
if os.environ.get('https_proxy'):
   del os.environ['https_proxy']
if os.environ.get('http_proxy'):
   del os.environ['http_proxy']
```

If this fixes the problem, try configuring your proxy to not affect traffic to localhost. See this GitHub issue for an example.

CHAPTER

FOUR

API REFERENCE

class flexlogger.automation.**Application**(*server_port=None*)

Represents the FlexLogger application.

__init__(*server_port=None*)

Connect to an already running instance of FlexLogger.

Parameters **server_port** (Optional[int]) – The port that the automation server is listening to. Omit this argument or pass None to detect the port of a running FlexLogger automatically.

Raises *FlexLoggerError* – if connecting fails.

close()

Close the application and disconnect from the automation server.

Further calls to this object will fail.

Return type None

disconnect()

Disconnect from the automation server, but leave the application running.

Further calls to this object will fail.

Return type None

get_active_project()

Gets the currently active (open) project.

Return type Optional[*Project*]

Returns The active project, or None if a project is not currently open.

Raises *FlexLoggerError* – if getting the active project fails.

classmethod launch(*, *timeout=40*, *path=None*)

Launch a new instance of FlexLogger.

Note that if this method is used to initialize a "with" statement, when the Application goes out of scope FlexLogger will be closed. To prevent this, call *disconnect()*.

Parameters

- **timeout** (float) – The length of time, in seconds, to wait for FlexLogger to launch before raising an exception. Defaults to 40.
- **path** (Union[str, pathlib.Path, None]) – The path to the FlexLogger executable to launch. Defaults to None, meaning the latest installed version will be launched.

Return type *Application*

Returns The created Application object

Raises *FlexLoggerError* – if launching FlexLogger or connecting to it fails.

open_project(*path*)
Open a project.

Parameters **path** (Union[str, pathlib.Path]) – The path to the project you want to open.

Return type *Project*

Returns The opened project.

Raises *FlexLoggerError* – if opening the project fails.

property server_port: int
The port that the automation server is listening to.

Return type int

class flexlogger.automation.**ChannelDataPoint**(*name*, *value*, *timestamp*)
The value for a channel at the specified timestamp.

property name: str
The name of the channel.

Return type str

property timestamp: datetime.datetime
The timestamp when the value occurred.

Return type datetime.datetime

property value: float
The value of the channel.

Return type float

class flexlogger.automation.**ChannelSpecificationDocument**(*channel*, *raise_if_application_closed*, *identifier*)
Represents the document that describes data channels.

Do not create this class directly; instead, use the return value of *Project.open_channel_specification_document()*.

get_channel_names()
Get all the channel names in the document.

Raises *FlexLoggerError* – if getting the channel names fails.

Return type List[str]

get_channel_value(*channel_name*)
Get the current value of the specified channel.

Parameters **channel_name** (str) – The name of the channel.

Raises *FlexLoggerError* – if getting the channel value fails.

Return type *ChannelDataPoint*

set_channel_value(*channel_name*, *channel_value*)
Set the current value of the specified channel.

Parameters

- **channel_name** (str) – The name of the channel.
- **channel_value** (float) – The value to set the channel to.

Raises *FlexLoggerError* – if setting the channel value fails.

Return type None

exception flexlogger.automation.**FlexLoggerError**(*message*)
Represents errors that occur when calling FlexLogger APIs.

__init__(*message*)
Initialize an exception.

Parameters **message** (str) – The message describing the error.

property message: str
The error message.

Return type str

with_traceback()
Exception.with_traceback(tb) – set self.to tb and return self.

class flexlogger.automation.**LoggingSpecificationDocument**(*channel*, *raise_if_application_closed*, *identifier*)
Represents a document that describes how data is logged.

Do not create this class directly; instead, use the return value of *Project.open_logging_specification_document()*.

get_log_file_base_path()
Get the log file base path.

Return type str

Returns The base path for the log file.

Raises *FlexLoggerError* – if getting the log file base path fails.

get_log_file_name()
Get the log file name.

Return type str

Returns The file name that will be logged to.

Raises *FlexLoggerError* – if getting the log file name fails.

get_resolved_log_file_base_path()
Get the resolved log file base path.

Return type str

Returns The resolved base path for the log file. The resolved base path will have any placeholders replaced with actual values. Note that time sourced placeholders such as {Second} are resolved at the time of the call, and may resolve to a different time on a subsequent call or when a log file is created.

Raises *FlexLoggerError* – if getting the resolved log file base path fails.

get_resolved_log_file_name()
Get the resolved log file name.

Return type str

Returns The resolved file name that will be logged to. The resolved file name will have any placeholders replaced with actual values. Note that time sourced placeholders such as {Second} are resolved at the time of the call, and may resolve to a different time on a subsequent call or when a log file is created.

Raises *FlexLoggerError* – if getting the resolved log file name fails.

get_test_properties()
Get all test properties.

Return type List[*TestProperty*]

Returns A list of the test properties on this document.

Raises *FlexLoggerError* – if getting the test properties fails.

get_test_property(*test_property_name*)
Get the test property with the specified name.

Throws a *FlexLoggerError* if a property with the specified name does not exist.

Parameters **test_property_name** (str) – The name of the test property.

Return type *TestProperty*

Returns The *TestProperty* with the specified name.

Raises *FlexLoggerError* – if a property with the specified name does not exist, or if getting the property fails.

remove_test_property(*test_property_name*)
Removes the test property with the specified name.

Parameters **test_property_name** (str) – The name of the test property.

Raises *FlexLoggerError* – if a property with the specified name does not exist, or if removing the property fails.

Return type None

set_log_file_base_path(*log_file_base_path*)
Set the log file base path.

Parameters **log_file_base_path** (str) – The log file base path.

Raises *FlexLoggerError* – if setting the log file base path fails.

Return type None

set_log_file_name(*log_file_name*)
Set the log file name.

Parameters **log_file_name** (str) – The log file name.

Raises *FlexLoggerError* – if setting the log file name fails.

Return type None

set_test_property(*property_name*, *property_value*, *prompt_on_start=False*)
Set the information for a test property.

Use this method to add a new test property or to modify an existing test property.

Parameters

- **property_name** (str) – The name of the test property. If a test property already exists with the same *name*, that test property will be updated with the new information passed to this method. Otherwise, a new test property will be created to reflect the specified test information.
- **property_value** (str) – The property value to set.

- **prompt_on_start** (bool) – Whether this property should be set when the test session starts. Defaults to False. If this is set to True, the operator should be prompted to define this property when the test session starts.

Raises *FlexLoggerError* – if setting the property fails.

Return type None

class flexlogger.automation.**Project**(*channel*, *raise_if_application_closed*, *identifier*)

Represents a FlexLogger project.

Do not create this class directly; instead, use the return value of *Application.open_project()*.

close()

Close the project.

Raises *FlexLoggerError* – if closing the project fails.

Return type None

open_channel_specification_document()

Open the channel specification document in the project.

Return type *ChannelSpecificationDocument*

Returns The opened document.

Raises *FlexLoggerError* – if opening the document fails.

open_logging_specification_document()

Open the logging specification document in the project.

Return type *LoggingSpecificationDocument*

Returns The opened document.

Raises *FlexLoggerError* – if opening the document fails.

open_screen_document(*filename*)

Open the specified screen document in the project.

Parameters **filename** (str) – The name of the screen document to open. Including the .flxscr extension in this argument is optional.

Return type *ScreenDocument*

Returns The opened document.

Raises *FlexLoggerError* – if a screen document of the specified name does not exist, or if opening the document fails.

open_test_specification_document()

Open the test specification document in the project.

Return type *TestSpecificationDocument*

Returns The opened document.

Raises *FlexLoggerError* – if opening the document fails.

property project_file_path: Optional[pathlib.Path]

Get the project file path on disk

Returns: The saved project file path if it exists, None otherwise

Return type Optional[pathlib.Path]

property test_session: *flexlogger.automation._test_session.TestSession*
Get the test session for the project.

Return type *TestSession*

class flexlogger.automation.**ScreenDocument**(*channel*, *raise_if_application_closed*, *identifier*)
Represents a document that displays data.

Do not create this class directly; instead, use the return value of *Project.open_screen_document()*.

Note that this class currently has no functionality; more functionality may be added in the future.

class flexlogger.automation.**TestProperty**(*name*, *value*, *prompt_on_start*)
Information about a test property.

__init__(*name*, *value*, *prompt_on_start*)
Create a new TestProperty.

Parameters

- **name** (str) – The name of the property.
- **value** (str) – The value of the property.
- **prompt_on_start** (bool) – Whether the operator should be prompted to define this property when the test session starts.

property name: str
The name of the property.

Return type str

property prompt_on_start: bool
Whether this property should be set when the test session starts.

If this is set to true, the operator should be prompted to define this property when the test session starts.

Return type bool

property value: str
The value of the property.

Return type str

class flexlogger.automation.**TestSession**(*channel*, *raise_if_application_closed*)
Represents a test session for a project.

Do not create this class directly; instead, use the property *Project.test_session*.

add_note(*note*)
Add a note to the current log file.

This method requires the test session to be in the *TestSessionState.RUNNING* state.

Parameters **note** (str) – The note to add to the log file.

Raises *FlexLoggerError* – if the test session is not in the *TestSessionState.RUNNING* state, or if adding the note fails.

Return type None

property elapsed_test_time: datetime.timedelta
Queries the elapsed test time

Return type datetime.timedelta

Returns The current tests's elapsed time if a test is running or paused, the most recent test's elapsed time if a test has been run and stopped.

Raises *FlexLoggerError* – if no test has ever been run since the project was loaded

pause()
Pauses the test session, if possible.

Return type bool

Returns True if the test was paused, otherwise False.

Raises *FlexLoggerError* – if pausing the test session fails.

resume()
Resumes the test session, if possible.

Return type bool

Returns True if the test was resumed, otherwise False.

Raises *FlexLoggerError* – if resuming the test session fails.

start()
Start the test session, if possible.

Return type bool

Returns True if the test was started, otherwise False.

Raises *FlexLoggerError* – if starting the test session fails.

property state: *flexlogger.automation._test_session_state.TestSessionState*
Get the current state of the test session.

Raises *FlexLoggerError* – if getting the current state fails.

Return type *TestSessionState*

stop()
Stop the test session, if possible.

Return type bool

Returns True if the test was stopped, otherwise False.

Raises *FlexLoggerError* – if stopping the test session fails.

class flexlogger.automation.**TestSessionState**(*value*)
An enumeration describing the possible states of a *TestSession*.

IDLE = 1
The *TestSession* is idle and not running.

Configuration changes can occur during this state.

INVALID_CONFIGURATION = 3
The project has a configuration error.

NO_VALID_LOGGED_CHANNELS = 4
No channels have been configured, or all channels are disabled or not available.

PAUSED = 5
The *TestSession* is paused.

Logging and events are not active while the test session is paused.

Configuration changes are not allowed when the test session is paused.

RUNNING = 2

The *TestSession* is running.

Logging and events are active while the test session is running.

Configuration changes are not allowed when the test session is running.

class flexlogger.automation.**TestSpecificationDocument**(*channel*, *raise_if_application_closed*, *identifier*)

Represents a document that describes a test.

Do not create this class directly; instead, use the return value of *Project.open_test_specification_document()*.

Note that this class currently has no functionality; more functionality may be added in the future.

4.1 Indices and tables

- genindex
- modindex

CHAPTER

FIVE

PACKAGE INFO

Info	NI FlexLogger API for Python
Author	NI

CHAPTER

SIX

ABOUT

The **niflexlogger-automation** package contains an API (Application Programming Interface) and examples for using Python to automate NI FlexLogger. The automation API supports modifying the configuration of existing FlexLogger projects and controlling the execution of FlexLogger test sessions. The package is implemented in Python. NI created and supports this package.

CHAPTER

SEVEN

REQUIREMENTS

niflexlogger-automation has the following requirements:

- FlexLogger 2021 R3+
- CPython 3.6 - 3.9

CHAPTER

EIGHT

INSTALLATION

To install **niflexlogger-automation**, use one of the following methods:

- pip:

```
$ python -m pip install niflexlogger-automation
```

- **easy_install** from setuptools:

```
$ python -m easy_install niflexlogger-automation
```

- Download the project source and run:

```
$ python setup.py install
```

CHAPTER

NINE

USAGE

Refer to the documentation for detailed information on how to use **niflexlogger-automation**.

Refer to Getting Started with CompactDAQ and FlexLogger, for more information on installing FlexLogger, using hardware, or downloading FlexLogger examples.

CHAPTER

TEN

AUTOMATED TESTS

The Python API package contains a number of automated tests which should be used to validate API changes before submitting a pull request. If a pull request contains new API functionality, new automated tests that exercise the new functionality should be included with the pull request.

To run the automated tests for the Python API, you must first configure FlexLogger to load the test plugins that the test projects use. To do this, copy `tests/assets/pythonTests.config` to `%public%\Documents\National Instruments\FlexLogger\Plugins\IOPlugins`, and in that file replace `<path to git repo>` with the path to the cloned repo.

After this is done, you can run the tests with tox.

CHAPTER

ELEVEN

SUPPORT / FEEDBACK

The **niflexlogger-automation** package is supported by NI. For support for **niflexlogger-automation**, open a request through the NI support portal at ni.com.

CHAPTER

TWELVE

BUGS / FEATURE REQUESTS

To report a bug or submit a feature request, use the GitHub issues page.

CHAPTER

THIRTEEN

LICENSE

niflexlogger-automation is licensed under an MIT-style license (see LICENSE). Other incorporated projects may be licensed under different licenses. All licenses allow for non-commercial and commercial use.

PYTHON MODULE INDEX

INDEX

Symbols

A

C

D

E

F

G

I

L

M

N