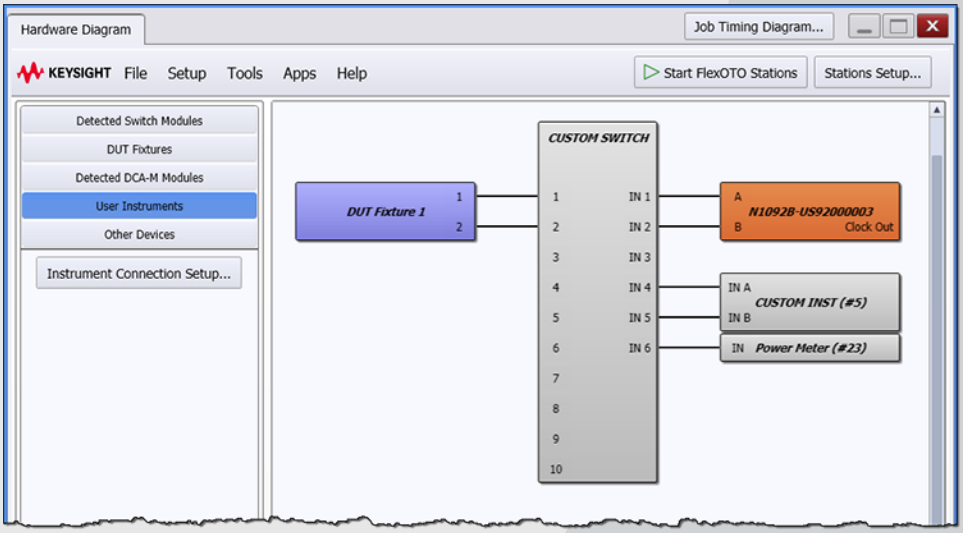
FlexOTO Custom Drivers



Adding generic switch and instrument drivers to FlexOTO.

Notices

Manual Part Number

N1002-90004

Edition

First, April 2023

Designed in USA

Keysight Technologies, Inc.
1400 Fountaingrove Parkway
Santa Rosa, CA 95403

Safety Notices

CAUTION A **CAUTION** notice denotes a hazard. It calls attention to an operating procedure, practice, or the like that, if not correctly performed or adhered to, could result in damage to the product or loss of important data. Do not proceed beyond a **CAUTION** notice until the indicated conditions are fully understood and met.

WARNING A **WARNING** denotes a hazard. It calls attention to an operating procedure, practice, or the like that, if not correctly performed or adhered to, could result in personal injury or death. Do not proceed beyond a WARNING notice until the indicated conditions are fully understood and met.

NOTE A NOTE calls the user’s attention to an important point or special information in the text.

Contents

1 Introduction

This document shows you how to extend the power of FlexOTO with hardware drivers. FlexOTO supports the writing of hardware drivers for optical switches that are not supported by FlexOTO and for measurement instruments such as a power meter. You can install up to eight measurement instruments drivers. These drivers can be written in Python (*.py) or any other language that can compile to an executable file (*.exe), such as C#.

Drivers are easy to write and you'll find all the details and examples within this document. All examples were written and tested in Python. The following figure illustrates the relationship between FlexOTO, a driver, and a non-supported optical switch (or measurement instrument).

NOTE All information in this document is included in FlexOTO's programmer's help. The same example drivers include a **Copy** button that places the code in Window's clipboard. No typing required! In the help's menu, click **SCPI Intro** > **Writing a Switch Driver** *or* **Writing a Instrument Driver**.

NOTE There is a small possibility that the commands, arguments, and responses described in this document may change. If edits are required for your scripts, they should be minor.

Figure 1. FlexOTO, Driver Software, and Switch/Instrument

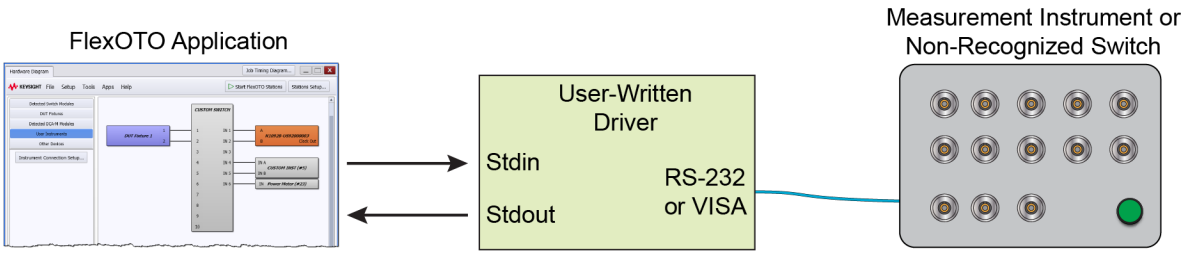
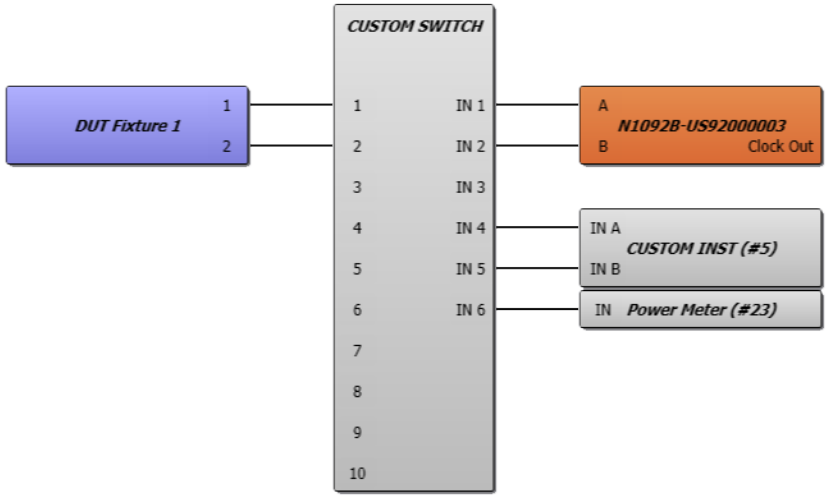


Figure 2. Example Switch and Instrument Drivers Installed on Hardware Diagram

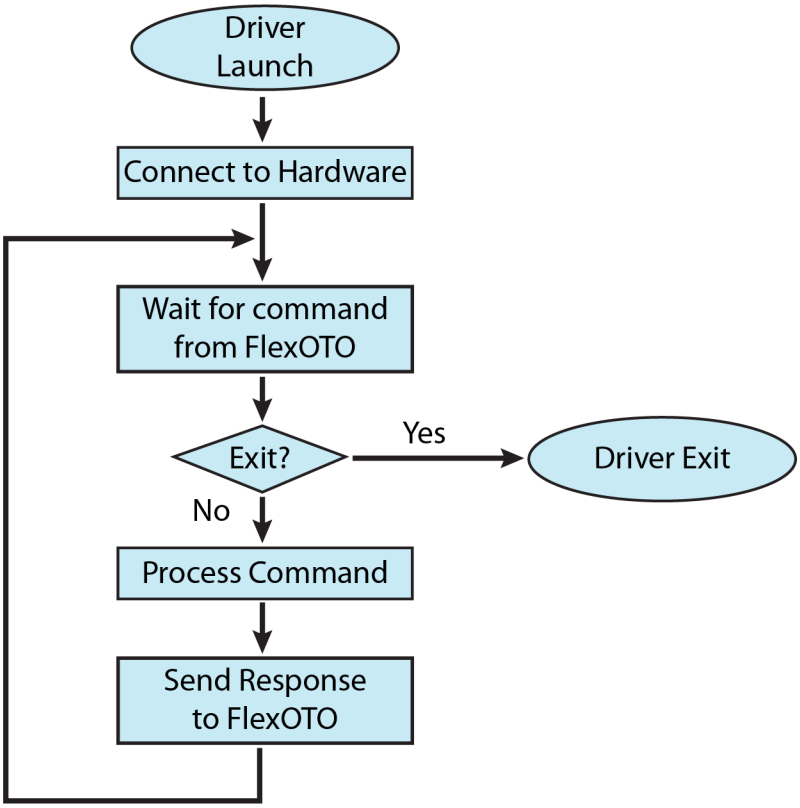


Driver Flowchart

The following flow chart provides the general process for both types of drivers: switch and instrument. Both drivers types have very similar structure. FlexOTO launches your driver from a dialog in which you have entered the driver's file name (along with path) and the COM port or VISA address of the instrument or switch. All communication between FlexOTO and your driver will be passed through *stdin* and *stdout*.

The driver runs in a continuous loop waiting for, and reacting to, commands sent from FlexOTO.

Figure 3. FlexOTO Driver Flowchart



When your driver is launched, FlexOTO immediately sends the `get_description` driver command to your driver. This command requests a description of your switch or instrument. This description allows FlexOTO to create a Switch block or Instrument block for the Hardware diagram. Your driver will return the following information in the form of a JavaScript Object Notation (JSON) string:

- Model name of the switch or instrument. In the case of a switch, the names of multiple internal switches can be included.
- Serial number.
- Settling time in seconds *(switches only)*.
- List of switch ports *(switches only)*.
- List of instrument measurement input connectors *(instruments only)*.

In response to a FlexOTO driver command, your driver will either send information back to FlexOTO as a JSON string, translate the command to send to your switch or instrument, or both.

Confirm with the "DONE" string

When your driver completes its initialization or finishes responding to FlexOTO command, which may include returning an error message or data, the driver *must* afterwards send the "DONE" string. Error messages, data, and the `"DONE"` string must be separately sent. The `"DONE"` string tells FlexOTO that the response is complete and that the driver will wait for the next command. For example, in response to receiving the `get_description` command a switch or instrument driver would send the following:

```
print(error_message)  # if needed
print(json_string)  # return description of
switch or instrument
print("DONE")
```

Error Messages

Errors that occur during initializing or running a driver can result in error messages. Driver initialization includes tasks such as FlexOTO finding and starting your driver, the driver parsing any command line arguments, and the driver establishing a connection to the switch or instrument. Errors are briefly displayed along the bottom of FlexOTO and listed in the Hardware Diagram's Message Log Viewer. To view the log, click Help > View Message Log. Error messages can also be read remotely by sending the `:SYSTem:ERRor:NEXT?` SCPI query.

The following are examples of errors that are detected by FlexOTO:

- 122, "File not found."
- 135, "Instrument Error;User driver initialization timed out"
- 136, "User driver command timed out: ""<*driver-command*>"""
  - Example: `"Instrument Error;User driver command timed out: set_routes ""1, 6"""`
- 137, "Instrument Error;Unable to parse description JSON: <*JSON-error*>"
  - Example: `"Instrument Error;Unable to parse description JSON: Invalid format"`

Errors encountered in the driver itself are reported to FlexOTO by sending the error message via `stdout`. If an error occurs, the driver creates the custom string that describes the error, and sends the error message to `stdout`. For example, here is the error message created for an incorrect switch or instrument COM port:

```
rm = visa.ResourceManager()
ser = rm.open_resource('COM-FOUR')
id = ser.query('*IDN?')
if not id:
    print("Switch COM port is not valid.")  #
error message
print("DONE")
```

If multiple message print statements were used in the above example, the strings would be concatenated by FlexOTO.

The following errors can be reported from drivers:

- 133, "Instrument Error;Connection failed: Invalid: *stdout-string*"
  - Example: `"Instrument Error;Connection failed Invalid: Only one command line argument allowed."`
- 134, 'Instrument Error;User driver command error: "*driver-command*" returned "*stdout-string*"'
  - Example: `"Instrument Error;User driver command timed out: ""set_routes"" returned ""The switch route could not be made!"""`

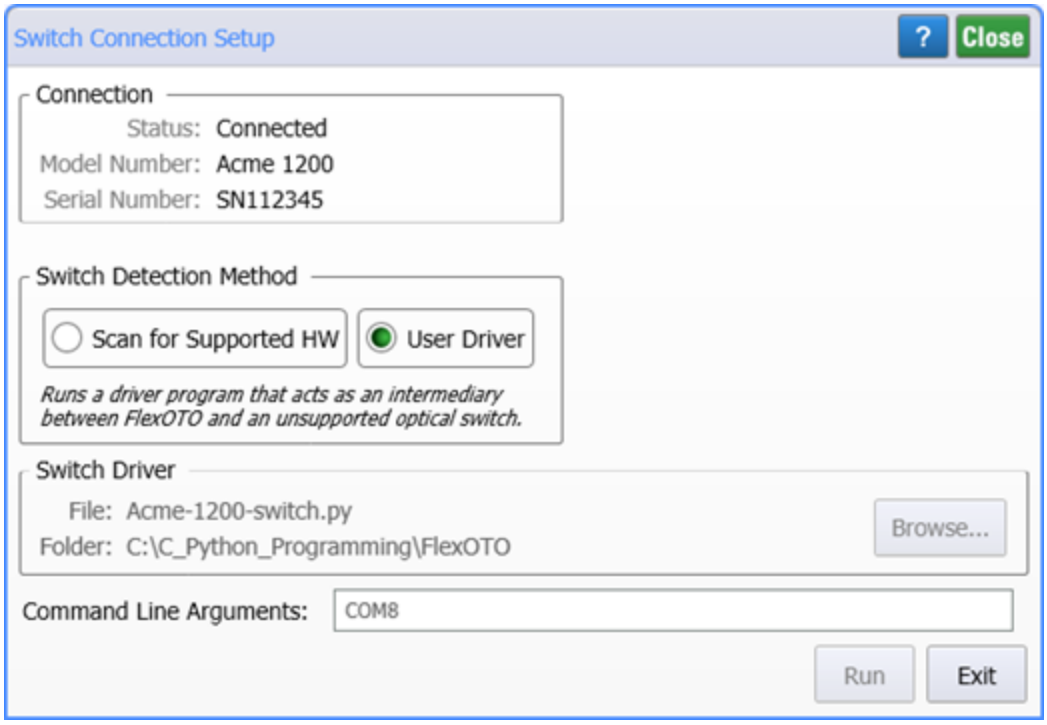
NOTE The maximum length for error messages is 255 characters, but it is recommended to keep your messages short.

To run your Switch driver

After creating your switch driver, use FlexOTO's **Switch Connection Setup** dialog to run your driver.

1. Open the FlexOTO application and click **Setup** > **Switch Connection Setup**.
2. In the dialog, select **User Driver**.

Figure 4. Switch Connection Setup dialog



3. Click **Browse**, search for your driver file, and click **OK**.
4. In the *Command Line Arguments* field, enter any command line arguments that your driver expects, such as the COM or VISA address of your switch hardware.
5. Click **Run**.

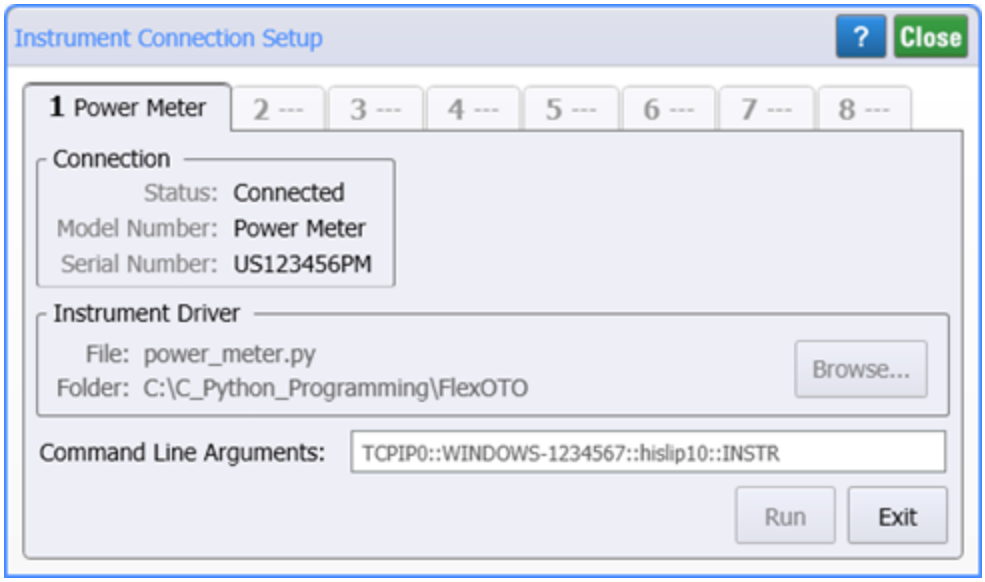
NOTE If, for some reason, your driver needs additional start-up information besides the switch address, you can append additional strings to the *Command Line Arguments* field and parse them within your driver.

To run your Instrument driver

After creating your instrument driver, use FlexOTO's **Instrument Connection Setup** dialog to run your driver.

1. Open the FlexOTO application and click **Setup** > **Instrument Connection Setup**.
2. In the dialog, select a tab. Each tab represents a driver that you can install.

Figure 5. Instrument Connection Setup dialog



3. Click **Browse**, search for your driver file, and click **OK**.
4. In the *Command Line Arguments* field, enter any command line arguments that your driver expects, such as the COM or VISA address of your instrument hardware.
5. Click **Run**.

NOTE If, for some reason, your driver needs additional startup information besides the instrument address, you can append additional strings to the *Command Line Arguments* field and parse them within your driver.

JSON Strings

JSON-formatted strings are used to send hardware descriptions and measurement results from your driver to FlexOTO. If your unfamiliar with the JSON format, you can find many tutorials on the internet.

In your driver, you can either directly create a string in JSON format or you can convert strings, variables, and data structures to a JSON string using a JSON library method such as `json.dumps()` in Python. The Python examples in this section demonstrate both of these methods for creating the identical JSON strings in your script. The scripts also validate the resulting JSON strings so that you can see that they both produce valid JSON string.

NOTE In your driver, you'll probably want to query the switch or instrument's serial number so that you can insert it into your JSON string.

In JSON strings:

- Are Unicode which is native to Python 3.0 and above (UTF-8).
- For readability, white space is legal in JSON strings as are the newline ('\n'), the carriage return ('\r'), and Python's line continuation character '\'.
- Multiple entries for `"InputPorts"` names, `"OutputPorts"` names, or instrument `"Inputs"` connectors names are entered as JSON arrays. If there is only one entry, you can list the element as either a single element array (`["A"]`) or as a name string ("A") without the brackets. Either method is works.
- JSON format errors or missing elements will cause your driver to fail when imported into FlexOTO.

Switch with Multiple Internal Switches

This section describes switch models that have multiple internal switches. In the following two scripts, notice that the required `Groups` element is a list that describes two internal switches. The `Name` elements provide the name of each internal switch.

Building and Validating JSON from a String

NOTE In JSON strings, string elements must be enclosed in double quotes. Single quoted string elements will invalidate the JSON.

Validate JSON from String

```
import json

def valid_json(jsonstr):
    try:
        json.loads(jsonstr)
    except ValueError as e:
        return False
    return True

json_str = """
{
    "ModelNumber": "My Switch",
    "SerialNumber": "12455",
    "SettlingTimeSeconds": "50e-3",
    "Groups": [
        {
            "Name": "SW1",
            "InputPorts": ["1", "2", "3", "4", "5", "6", "7", "8"],
            "OutputPorts": ["IN"],
            "Wavelengths": ["1350 nm", "1550 nm"]
        },
        {
            "Name": "SW2",
            "InputPorts": ["1", "2", "3", "4", "5", "6", "7", "8", "9"],
            "OutputPorts": ["IN 1", "IN 2", "IN 3", "IN 4"],
            "Wavelengths": ["1350 nm", "1550 nm"]
        }
    ]
} """

print(valid_json(json_str))

```

Building and Validating JSON from Data Structure

NOTE

Strings, variables, and data structures in your code can use single or double quotes as allowed by the language. In this script, the `json.dumps` method converts single quote characters to double quotes.

Validate JSON from Structure

```
1  import json
2
3  def valid_json(jsonstr):
4      try:
5          json.loads(jsonstr)
6      except ValueError as e:
7          return False
8      return True
9
10 sw1 = {'Name': 'SW1',
11        'InputPorts': ['1', '2', '3', '4', '5', '6', '7', '8'],
12        'OutputPorts': ['IN']}
13 sw2 = {'Name': 'SW2',
14        'InputPorts': ['1', '2', '3', '4', '5', '6', '7', '8', '9'],
15        'OutputPorts': ['IN 1', 'IN 2', 'IN 3', 'IN 4'],
16        'Wavelengths': ['1350 nm', '1550 nm']}
17 sw_list = [sw1, sw2]
18 data_structure = {'ModelNumber': 'My Switch', 'SerialNumber': '12455', 'Set-
   tlingTimeSeconds': '50e-3', 'Groups': sw_list}
19 json_str = json.dumps(data_structure)
20 print(valid_json(json_str))
21
```

Switch with One Internal Switch

This section describes a switch model that has only one internal switch. In the following two scripts, notice that the `Groups` element is *required* even though this switch model only has one internal switch. In this case, `Groups` is a list that contains a single item. Even though the internal switch does *not* have a name, the `Name` element is still *required* but is an empty string.

Building and Validating JSON from a String

Validate JSON from String

```
import json

def valid_json(jsonstr):
    try:
        json.loads(jsonstr)
    except ValueError as e:
        return False
    return True

json_str = """
{
    "ModelNumber": "My Switch",
    "SerialNumber": "12455",
    "SettlingTimeSeconds": "50e-3",
    "Groups": [
        {
            "Name": "",
            "InputPorts": ["1", "2", "3", "4", "5", "6", "7", "8"],
            "OutputPorts": ["IN"],
            "Wavelengths": ["1350 nm", "1550 nm"]
        }
    ]
} """

print(valid_json(json_str))

```

NOTE In JSON strings, string elements must be enclosed in double quotes. Single quoted string elements will invalidate the JSON.

Building and Validating JSON from Data Structure

NOTE Strings, variables, and data structures in your code can use single or double quotes as allowed by the language. In this script, the `json.dumps` method converts single quote characters to double quotes.

Validate JSON from Structure

```
1  import json
2
3  def valid_json(jsonstr):
4      try:
5          json.loads(jsonstr)
6      except ValueError as e:
7          return False
8      return True
9
10 switch = {'Name': '',
11           'InputPorts': ['1', '2', '3', '4', '5', '6', '7', '8'],
12           'OutputPorts': ['IN'],
13           'Wavelengths': ['1350 nm', '1550 nm']}
14
15 sw_list = [switch]
16 data_structure = {'ModelNumber': 'My Switch', 'SerialNumber': '12455', 'Set-
   tlingTimeSeconds': '50e-3', 'Groups': sw_list}
17 json_str = json.dumps(data_structure)
18 print(valid_json(json_str))
19
```

2 Writing a Switch Driver

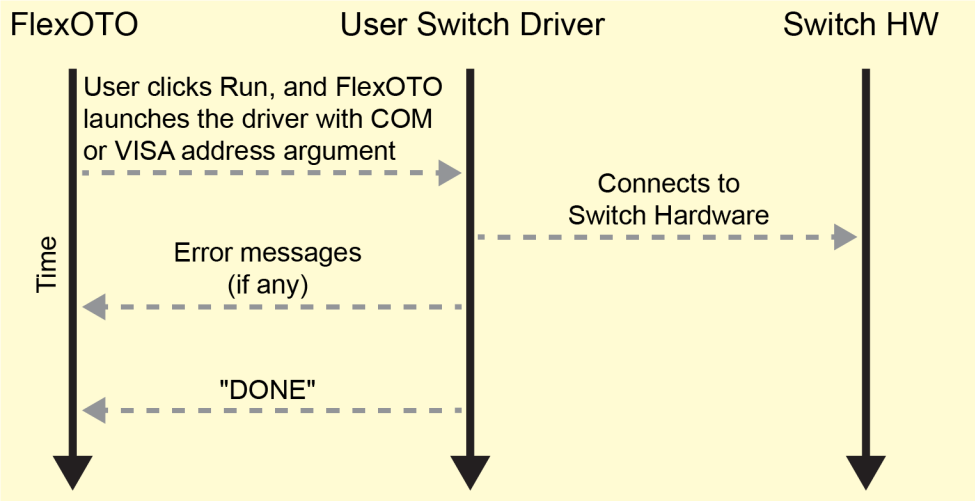
Your switch driver must respond to the following argument and four commands from FlexOTO:

All messages are read by the driver using `stdin`. For example, in Python you would use the `input()` statement. All messages are sent by the driver to FlexOTO using `stdout`. For example, in Python you would use the `print()` statement.

Command Line Arguments Sent to Driver

When FlexOTO runs the switch driver, FlexOTO sends any command line arguments to the driver. What arguments are expected depends on the driver. Usually command line arguments are used to pass the switch's COM or VISA address, but they can contain other configuration information as well. The arguments are sent from FlexOTO when the user clicks **Run** in the **Switch Connection Setup** dialog or sends the `:SWITch:RDRiver` command to FlexOTO. Your switch driver must parse any arguments, establish the connection with the switch, and send a response to FlexOTO.

Figure 6. Interaction when Switch Driver is Started



Returned Response to FlexOTO

The response should always return the `"DONE"` string. If the switch responds with an error, the error should be returned *before* the `"DONE"` string.

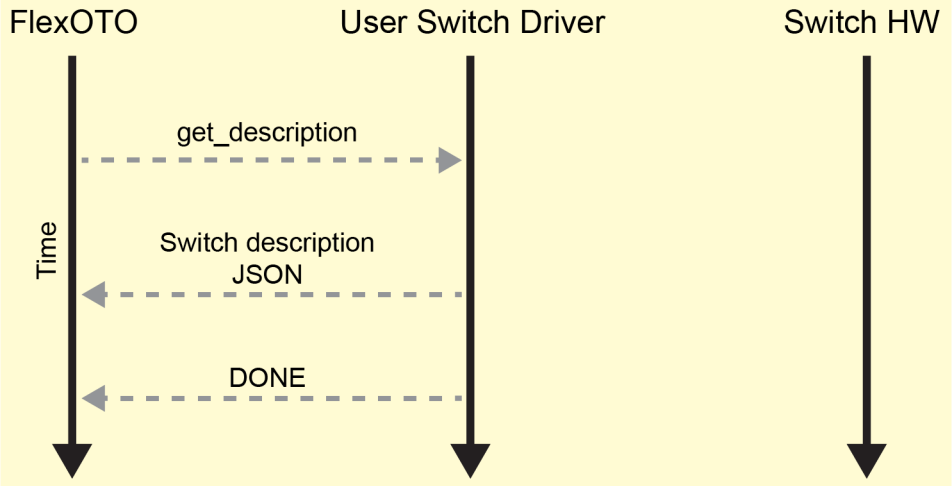
```
print(error-messages)  # if needed
print("DONE")
```

t

get_description Command

This command returns a description of the switch hardware in JSON format to FlexOTO. The following figure show the actions that occur with this command.

Figure 7. Interaction when the `get_description` query is sent to the switch driver



The returned JSON string provides the following information about the switch. FlexOTO uses this information when drawing one or more Switch blocks on FlexOTO's Hardware Diagram and to use the proper names when sending the `set_routes` command to the switch driver.

- Switch model number. *(shown on switch block)*
- Switch serial number. *(shown on switch block)*
- Switch settling time in seconds.
- For each of the switch's internal switches (there may be only one):
  - Name of internal switch. If the switch model only has a single internal switch, the name should be an empty string. *(shown on switch block)*
  - SupportsDisconnected *(optional)*
  - Names of input ports. *(shown on switch block)*
  - Names of output ports. *(shown on switch block)*
  - Wavelengths *(optional)*

Command from FlexOTO

```
get_description
```

Returned Response to FlexOTO

A JSON string describing the switch, followed by DONE, on separate lines.

```
print(error_messages)  # if needed
print(json_string)
print("DONE")
```

The following example JSON string creates two switch blocks (SW1 and SW2) that will be available for placing on FlexOTO's Hardware Diagram.

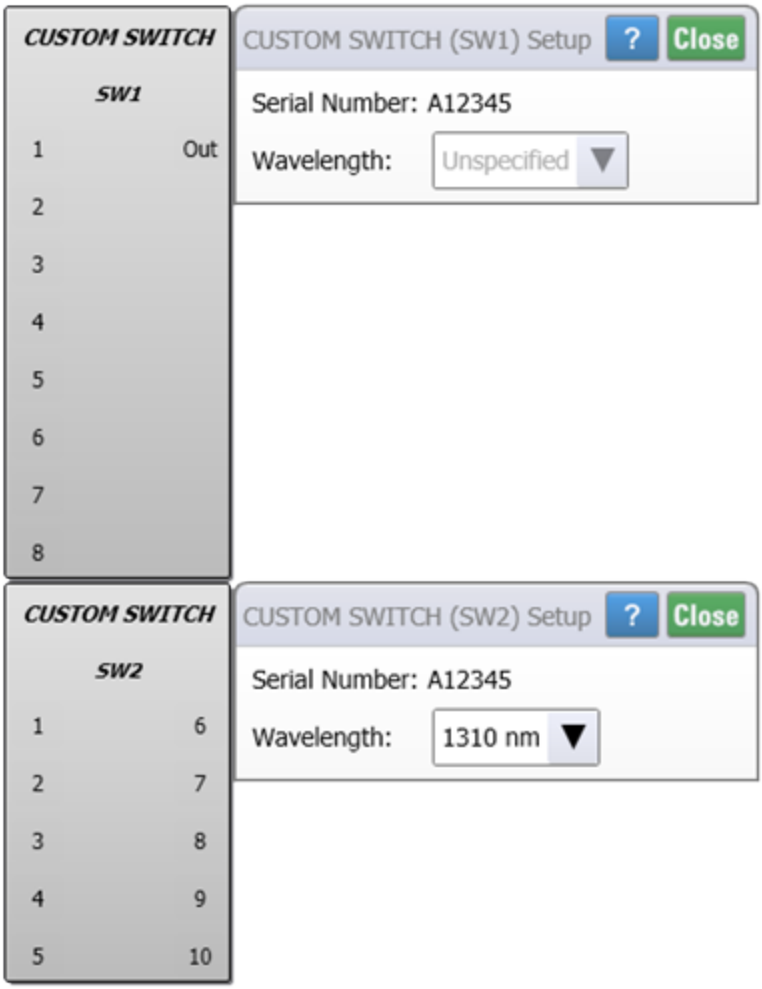
Example of returning a JSON string

```
json_string = """
{
    "ModelNumber": "CUSTOM SWITCH",
    "SerialNumber": "A12345",
    "SettlingTimeSeconds": 50e-3,
    "Groups": [
        {
            "Name": "SW1",
            "InputPorts": ["1","2","3","4","5","6","7","8"],
            "OutputPorts": ["OUT"]
        },
        {
            "Name": "SW2",
            "InOutPorts": ["1","2","3","4","5","6","7","8","9","10"],
            "Wavelengths": ["1350 nm", "1550 nm"]
        }
    ]
} """
print(json_string)
print("DONE")

```

The two switch blocks that this JSON string creates are shown placed on the Hardware Diagram in the following figure. The model number, group name, and port labels appear on the block. Also notice in the switch Setup dialogs show the switch's serial number and that switch SW1's wavelength selection is grayed out while switch SW2's wavelength selection is available.

Figure 8. CUSTOM SWITCH Blocks on Hardware Diagram



JSON Elements Returned to FlexOTO

FlexOTO expects to find the following elements in the imported JSON string. See the above example.

`ModelNumber` Element

The `ModelNumber` element is a string that names the optical switch on FlexOTO's Hardware Diagram. The name that you give is entirely up to you and need not be related to the actual switch's model number.

```
"ModelNumber": "CUSTOM SWITCH",
```

`SerialNumber` Element

The `SerialNumber` element is a string that is the optical switch's serial number. You can query this value from the switch and then insert the name into the JSON string.

```
"SerialNumber": "A1234",
```

`SettlingTimeSeconds` Element

The value of the `SettlingTimeSeconds` element is a real number that represents the time in seconds that the switch requires to stabilize after a switch route has been selected. For example, 0.05 for 50 ms. When setting the switch route, FlexOTO will wait for this time to pass before acquiring or analyzing data through the route.

```
"SettlingTimeSeconds": 50e-3,
```

If you don't know your switch's settling time, you can enter zero or any other time delay that you want.

```
"SettlingTimeSeconds": 0.0,
```

`Groups` Element

The `Groups` element is a list of one or more internal switch modules using the following elements. For each switch group, include the following elements:

- `Name`
- `SupportsDisconnected`
- `InputPorts`
- `OutputPorts`
- `InOutPorts`
- `Wavelengths` *(optional)*

FlexOTO displays the switch name, input ports, and output ports. Each group will be displayed on the resulting switch block that can be installed on FlexOTO's Hardware Diagram.

```
"Groups": []
```

`Name` Element

The `Name` element labels the switch group on FlexOTO's Hardware Diagram and should match the switch's front panel. If the switch model does *not* include multiple internal switches, `Name` should be assigned an empty string. When requesting that a switch route be created, FlexOTO sends the switch group name with the `set_routes` command to the driver. Refer to *set_routes Command* on page 26. Your driver will need to translate these strings to the correct strings for the switch. Consult the switch manual to find the exact strings to use.

```
"Name": "SW1",
```

`SupportsDisconnected` Element (optional)

This optional element indicates if the optical switch allows the output state to be disconnected. The value of this element can be set to `true` or `false`. A `true` setting indicates that all output ports can be disconnected from the input ports. FlexOTO's Instrument AutoCal is disabled when this element is `false`, because the calibration requires that DCA-M modules be disconnected from all input signals.

```
"SupportsDisconnected": true,
```

NOTE

If the switch does *not* allow the output ports to be disconnected from the input ports, set `SupportsDisconnected` to `false`. You cannot run FlexOTO's Instrument AutoCal. You can, however, disconnect the fiber-optic cables from the DCA-M module's inputs and perform a module calibration from FlexDCA.

`InputPorts` Element

Identifies and labels switch input ports. When requesting that a switch route be created, FlexOTO sends the switch port names with the `set_routes` command to the driver. The element requires the associated `OutputPorts` element. For any-to-any switches, use the `InOutPorts` instead.

```
"InputPorts": ["1","2","3","4"],
```

`OutputPorts` Element

Identifies and labels switch output ports. When requesting that a switch route be created, FlexOTO sends the switch port names with the `set_routes` command to the driver. The element requires the associated `InputPorts` and `OutputPorts` elements. For any-to-any switches, use the `InOutPorts` instead.

```
"OutputPorts": ["OUT A","OUT B"],
```

`InOutPorts` Element

This element describes an any-to-any switch matrix and is used in place of the `InputPorts` and `OutputPorts` element. Any port can be an input or an output port. When requesting that a switch route be created, FlexOTO sends the switch port names with the `set_routes` command to the driver.

```
"InOutPorts": ["1","2","3","4","5","6","7","8"]
```

`Wavelengths` Element (optional)

Use this optional element to indicate all possible switch wavelength settings. Not all optical switches support this setting. If your switch does *not* support the wavelengths settings, your driver must still process the `set_wavelength` driver command but the response should do nothing except to return the `"DONE"` string). Refer to *set_wavelength Command* on page 28.

```
"Wavelengths": ["1330 nm","1550 nm"]
```

NOTE

In FlexOTO's GUI, the switch wavelength setting is located by clicking on the Switch block on the Hardware Diagram. Don't confuse this wavelength setting with the setting that is used to change a DCA-M modules wavelength setting. The location of the DCA-M setting is found by clicking the Stations Setup button which is located above the FlexOTO's Hardware Diagram.

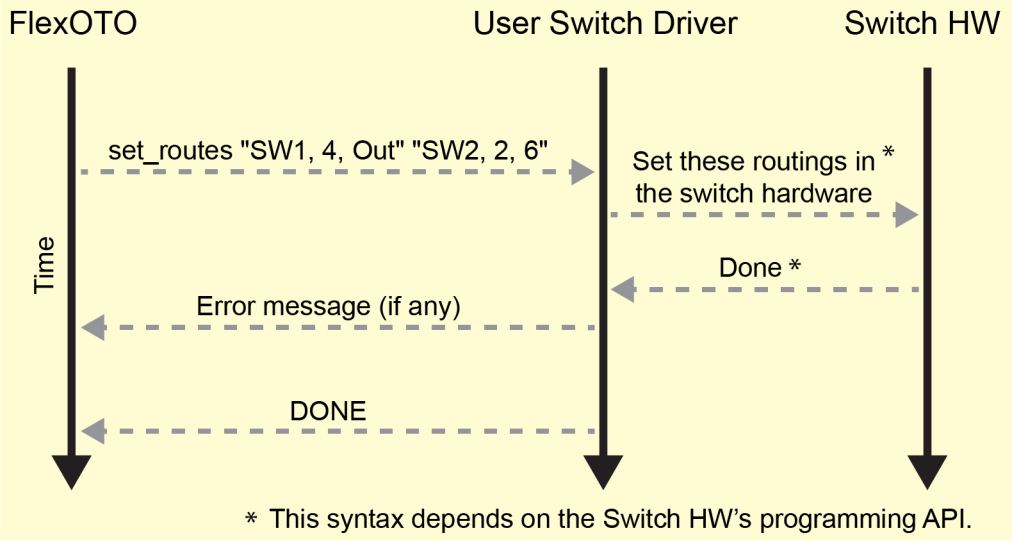
You can enter any wavelength values that you want. In fact there are no rules on the strings except that their length is limited. FlexOTO uses these strings to populate the wavelength selections in the Switch Setup dialog that appears when you click on a Switch block. When you make a selection in this dialog, FlexOTO simply returns the wavelength string back to your driver as an argument to the `set_wavelength` command.

set_routes Command

This command instructs the switch driver to create one or more routes in the switch hardware. A single route description is formed by the following three comma-separated values enclosed in double quote characters: group name, input port name, and output port name.

The two ports must belong to the same group. If multiple routes are described, each route description must be separated by a space character as shown in the example below.

Figure 9. Interaction when the `set_routes` command is sent to the switch driver



Command from FlexOTO

```
set_routes "<group-name>, <input-port-name>, <output-port-name>" "<group-name>, <input-port-name>, <output-port-name>" ...
```

Examples

This example creates two switch routes in a switch model that has multiple internal switches. Notice that the string delimiter between routes is a space (" ") character. The delimiter between a route's arguments is the comma character.

```
set_routes "SW1, 4, Out" "SW2, 2, 6"
```

Two switch routes in a switch module that has a single internal switch. Notice that because there is only one internal switch, the <*group-name*> is *not* included:

```
set_routes ", 4, Out" ", 2, 6"
```

Returned Response to FlexOTO

This command should always return a value of **DONE**. If the switch responds with an error, the error should be returned *before* the **DONE** value.

```
print('error_message')  # if any
print("DONE")
```

set_wavelength Command

This command instructs the driver to set the wavelength setting of a given group. The <*group-name*> argument should be whatever the writer specified in the `get_description` JSON and it is possible to be an empty string. This command is sent after the user makes a wavelength selection in FlexOTO's Switch Setup dialog. The wavelength-setting string will match one of the strings provided in the `wavelength` JSON element sent in response to the `get_description` driver command.

Figure 10. Wavelength Setting in FlexOTO's Switch Setup dialog

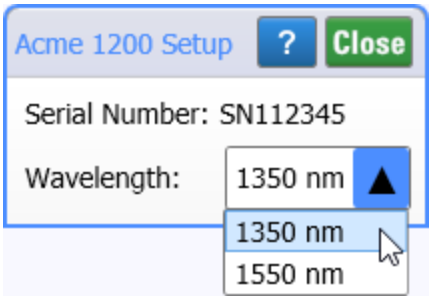
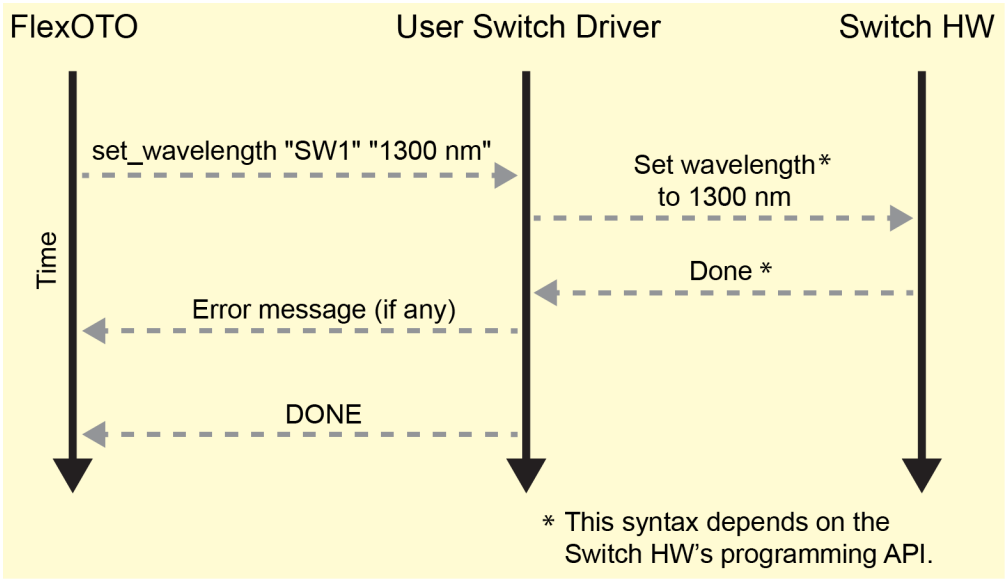


Figure 11. Interaction when the `set_wavelength` command is sent to the switch driver



Command from FlexOTO

`set_wavelength "<group-name>" "<wavelength-setting>"`

Examples

This example enters a wavelength setting in a switch model that has multiple internal switches. Notice that the string delimiter between <*group-name*> and <*wavelength-setting*> is a space (`" "`) character.

```
set_wavelength "SW1" "1330 nm"
```

The <*group-name*> argument should be whatever the writer specified in the `get_description` JSON and it is possible to be an empty string:

```
set_wavelength "" "1330 nm"
```

Returned Response to FlexOTO

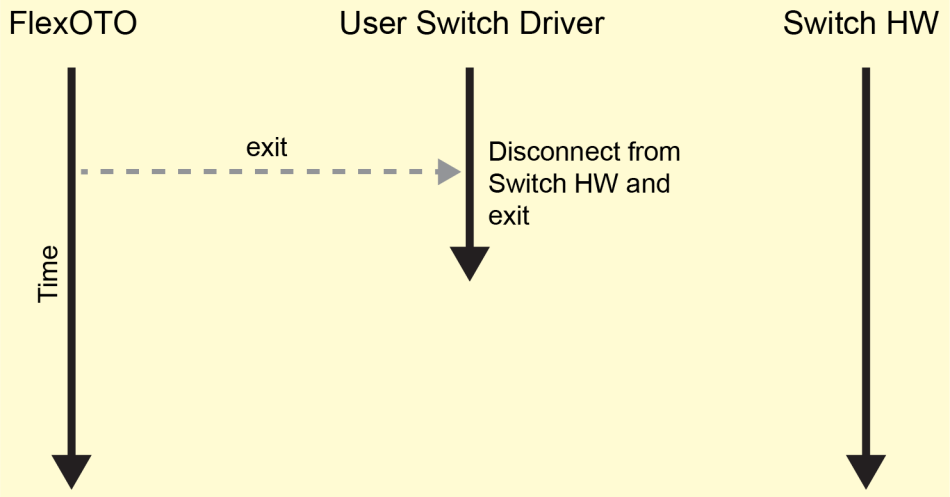
This command should always return the `"DONE"` string. If the switch responds with an error, the error should be returned *before* the `"DONE"` string.

```
print('error_message')  # if any
print("DONE")
```

exit Command

This command instructs the switch driver to disconnect from the switch hardware and end the driver process. The command is sent from FlexOTO when the user clicks **Exit** in the **Switch Connection Setup** dialog or the `:SWITch:DISConnect` command is sent to FlexOTO. This command does not expect a return value.

Figure 12. Interaction when the `exit` command is sent to the Switch Driver



Command

```
exit
```

Returned Response to FlexOTO

This command does not provide a return value.

Example Switch Driver

This is an example of a switch driver written in Python. This example driver allows FlexOTO to use a DiCon GP600 which FlexOTO already supports! But, writing a driver for a supported switch that you might be familiar with is good technique for learning how to create and test your script. This driver connects the switch using the PC's RS-232 port (USB) port, so you would pass the COM port to the driver as explained in Switch Connection Setup dialog. Refer to *To run your Switch driver* on page 10. For example, `COM4`.

Custom GP600 as Generic Switch

```
#*****************************************************************************
#    MIT License
#    Copyright(c) 2023 Keysight Technologies
#    Permission is hereby granted, free of charge, to any person obtaining a copy
#    of this software and associated documentation files (the "Software"), to deal
#    in the Software without restriction, including without limitation the rights
#    to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
#    copies of the Software, and to permit persons to whom the Software is
#    furnished to do so, subject to the following conditions:
#    The above copyright notice and this permission notice shall be included in all
#    copies or substantial portions of the Software.
#    THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
#    IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
#    FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
#    AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
#    LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
#    OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
#    SOFTWARE.
#*****************************************************************************

import pyvisa as visa
import pyvisa.constants
DONE = 'DONE'


def get_command_line_argument():
```

```
    """ Reads command-line argument COM port for serial connection sent from FlexOTO's Switch Connection Setup dialog.
    argv[0] will be this script's name. Subsequent argv values will be the command line arguments from FlexOTO.

    Returns
        sys.argv[1]: str
    """
    import sys
    if len(sys.argv) != 2:
        print('Invalid: Only one command line argument allowed, for example "COM4".')
        print(DONE)
        return None
    if 'com' not in sys.argv[1].lower():
        print('Invalid: argument must be a COM port, for example "COM4".')
        print(DONE)
        return None
    return sys.argv[1]


def connect_to_switch(comport):
    """ Opens a connection to the optical switch.

        Args
            comport: str
        Returns
            ser: visa.ResourceManager object of switch
    """
    visa_library = r'C:\WINDOWS\system32\visa64.dll'
    com_address = {'COM1': 'ASRL1::INSTR',
                   'COM2': 'ASRL2::INSTR',
                   'COM3': 'ASRL3::INSTR',
                   'COM4': 'ASRL4::INSTR',
                   'COM5': 'ASRL5::INSTR',
                   'COM6': 'ASRL6::INSTR',
                   'COM7': 'ASRL7::INSTR',
```

```
61                      'COM8': 'ASRL8::INSTR',
62                      }
63      try:
64          rm = visa.ResourceManager()
65          ser = rm.open_resource(com_address[comport])
66          ser.timeout = 1000  # (seconds)
67          ser.read_termination = '\r'
68          ser.write_termination = '\r'
69          ser.set_visa_attribute(visa.constants.VI_ATTR_ASRL_BAUD, 115200)
70          ser.set_visa_attribute(visa.constants.VI_ATTR_ASRL_DATA_BITS, 8)
71          ser.set_visa_attribute(visa.constants.VI_ATTR_ASRL_STOP_BITS, visa.-
constants.VI_ASRL_STOP_ONE)
72          ser.set_visa_attribute(visa.constants.VI_ATTR_ASRL_PARITY, visa.-
constants.VI_ASRL_PAR_NONE)
73      except:
74          print('Unable to open port: ' + comport)
75          print(DONE)
76          return None
77      # Do a query to make sure the connection is set up correctly.
78      sw_name = ser.query('*IDN?')
79      if not sw_name:
80          print("Couldn't connect to switch.")
81          print(DONE)
82          return None
83      # Successfully connected
84      print(DONE)
85      return ser
86
87
88  def get_description(oswitch):
89      """ Responds to FlexOTO get_description query by returning a JSON formatted
string that describes the switch.
90      Groups represent one or more internal switch modules.
91      Args
92          oswitch: visa.ResourceManager object of switch
93      stdout
94          description: JSON str
```

```
95          DONE: str
96      Returns
97          None
98      """
99
100     s = oswitch.query('*IDN?')
101     id = s.split(',')
102     serial_number = id[2].strip()
103     description = """
104 {
105     "ModelNumber": "DiCon GP600",
106     "SerialNumber": "%s",
107     "SettlingTimeSeconds": 50e-3,
108     "Groups": [
109         {
110             "Name": "M1",
111             "InputPorts": ["1","2","3","4","5","6","7","8"],
112             "OutputPorts": ["IN"]
113         },
114         {
115             "Name": "X1",
116             "InputPorts":
["1","2","3","4","5","6","7","8","9","10","11","12","13","14","15","16","17","18","1-
9",
117                              "20","21","22","23","24","25","26","27","28","29","30","-
31","32"],
118             "OutputPorts": ["IN 1","IN 2","IN 3","IN 4","IN 5","IN 6","IN 7","IN 8"]
119         }
120     ]
121 } """ % serial_number  # Insert the queried serial number
122     print(description)
123     print(DONE)
124
125
126 def convert_port_names_to_arguments(route):
127     """ The set routes commands uses the name of the ports as configured in the get_
description JSON string.
128     However, the arguments sent to the hardware switch have different order and
```

```
    strings. In FlexOTO, the
129     GP600 input ports are used as output and the GP600 output ports are used as
    inputs.
130     Args:
131         route: str (comma delimited)
132     Returns
133         cmd_parts: str
134     """
135     s = route.replace('"', '')
136     cmd_parts = s.split(', ')
137     group = cmd_parts[0]
138     flexoto_in_port = cmd_parts[1]
139     flexoto_out_port = cmd_parts[2]
140     if 'X1' in group:
141         gp600_in_port = flexoto_out_port.replace('IN ', '')
142     elif 'M1' in cmd_parts[0]:
143         gp600_in_port = flexoto_out_port.replace('IN', '1')
144     else:
145         print('Unknown group name: ' + group)
146         return
147     gp600_out_port = flexoto_in_port
148     cmd_parts[1] = gp600_in_port
149     cmd_parts[2] = gp600_out_port
150     return cmd_parts
151
152
153 def set_routes(oswitch, args):
154     """ While the Test Plans are running, FlexOTO's Sessions configure one or more
    switch routes (paths) through
155     the optical switch FlexOTO with this command. Each route is defined by the fol-
    lowing three fields:
156     name, switch block input port, and switch block output port.
157     Example of a routes string for GP600:
158         "X1, 4, IN 1" "X1, 5, IN 2"
159     resulting in these commands to switch hardware:
160         oswitch.write('X1 CH 1 4')
161         oswitch.write('X1 CH 2 5')
```

```
162
163     Args
164         oswitch: visa.ResourceManager object of switch
165         args: str (comma delimited)
166     stdout
167         DONE: str
168     Returns
169         None
170     """
171     routes = args.split('" "')
172     for route in routes:
173         cmd_parts = convert_port_names_to_arguments(route)
174         group = cmd_parts[0]
175         gp600_in_port = cmd_parts[1]
176         gp600_out_port = cmd_parts[2]
177         cmd = group + ' CH ' + gp600_in_port + ', ' + gp600_out_port
178         oswitch.write(cmd)
179         errcode = oswitch.query("SYST:ERR?")
180         if '+0' in errcode:
181             continue
182         else:
183             print('Error code: {} ("{}")'.format(errcode, cmd))
184     print(DONE)
185
186
187 def set_wavelength(oswitch, grp_and_wavelength):
188     """ Specifies the wavelength setting for a switch group. If switch does not sup-
    port wavelength settings
189     do nothing and still send DONE to stdout.
190     Args
191         oswitch: visa.ResourceManager object of switch
192         grp_and_wavelength: str (comma delimited)
193     stdout
194         DONE: str
195     Returns
196         None
```

```
    """
    # Does nothing.
    print(DONE)


# Main loop

com_port = get_command_line_argument()
switch = connect_to_switch(com_port)
switch.write('*RST')
if switch:
    while True:
        # Loop until FlexOTO sends 'exit'.
        fromFlexOTO = input()  # stdin from FlexOTO
        if 'get_description' in fromFlexOTO:
            get_description(switch)
        elif 'set_routes' in fromFlexOTO:
            set_routes(switch, fromFlexOTO.replace('set_routes ', ''))
        elif 'set_wavelength' in fromFlexOTO:
            set_wavelength(switch, fromFlexOTO.replace('set_wavelength ', ''))
        elif 'exit' in fromFlexOTO:
            break

```

3 Writing an Instrument Driver

Your instrument driver must respond to the following argument and three driver commands from FlexOTO:

All messages are read by the driver using `stdin`. For example, in Python you would use the `input()` statement. All messages are sent by the driver to FlexOTO using `stdout`. For example, in Python you would use the `print()` statement.

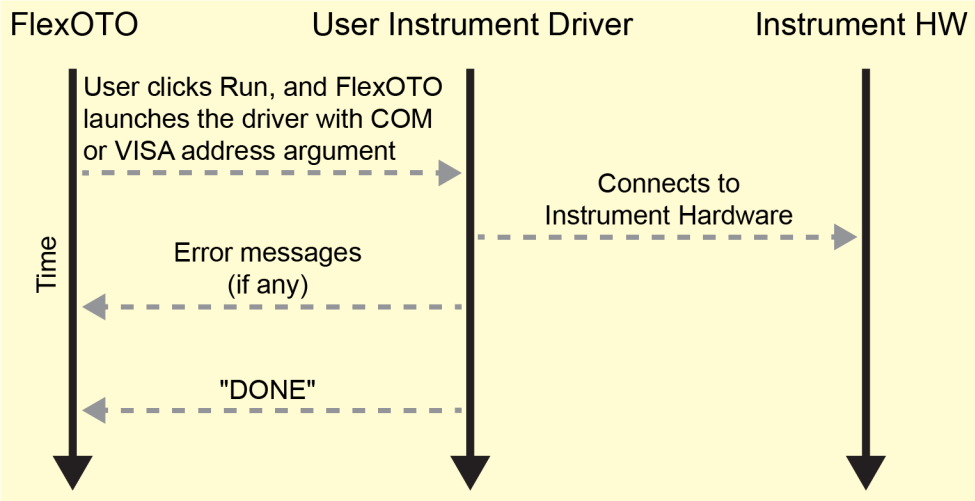
FlexOTO sends the `get_description` driver command and the driver returns the instrument's model and serial numbers along with front panel input connectors.

FlexOTO sends the `measure` driver command with a list of Instrument input connectors on which to perform measurements. One or more available measurements are defined in the driver. All measurements are performed on each specified input connector. The names of the measurements are returned to FlexOTO along with the measurement results.

Command Line Arguments Sent to Driver

When FlexOTO runs the instrument driver, FlexOTO sends any command line arguments to the driver. What arguments are expected depends on the driver. Usually command line arguments are used to pass the instrument's COM or VISA address, but they can contain other configuration information as well. The command line arguments are sent from FlexOTO when the user clicks **Run** in the **Instrument Connection Setup** dialog or sends the `:INSTrument:RDRiver` command to FlexOTO. Your instrument driver must parse any arguments, establish the connection with the instrument, and send a response to FlexOTO.

Figure 13. Interaction when Instrument Driver is Started



Returned Response to FlexOTO

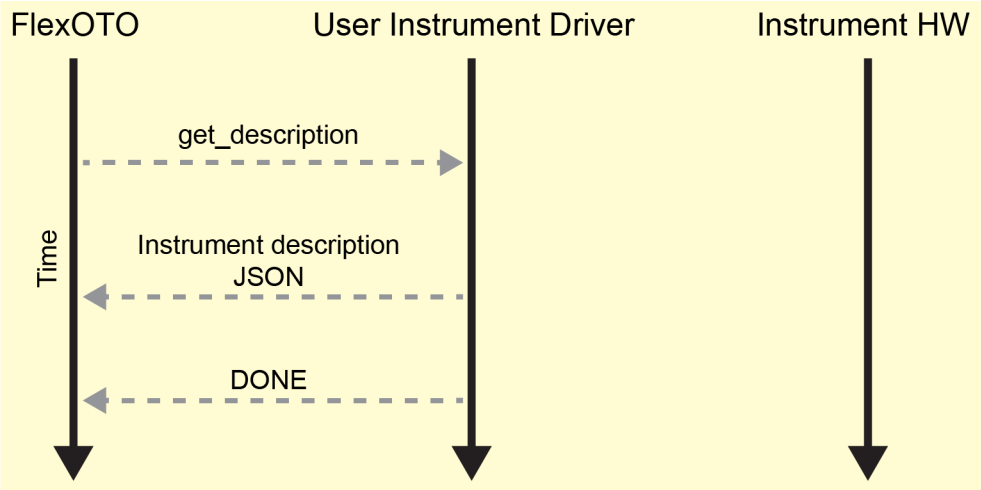
The response should always return the `"DONE"` string. If the instrument responds with an error, the error should be returned *before* the `"DONE"` string.

```
print(error-messages)  # if needed
print("DONE")
```

get_description Command

This command returns a description of the instrument hardware in JSON format to FlexOTO. The following figure show the actions that occur with this command.

Figure 14. Interaction when the `get_description` query is sent to the Instrument Driver



The returned JSON string provides the following information about the instrument. FlexOTO uses this information when drawing one or more Instrument blocks on FlexOTO's Hardware Diagram. FlexOTO also passes input connector names as arguments to the measure command. The instrument performs all of its assigned measurements to each listed connector. Refer to *measure Command* on page 44.

- Instrument's model number. *(shown on switch block)*
- Instrument's serial number. *(shown on switch block)*
- List of the names of the instrument's measurement input connectors. *(shown on switch block)*

Command from FlexOTO

`get_description`

Returned Response to FlexOTO

Returns a JSON string that describes the Instrument, followed by `"DONE"`, on separate lines.

```
print(error_messages)  # if needed
print(json_string)
print("DONE")
```

The following example JSON string creates an Instrument block *(My Instrument SN12345)* that will be available for placing on FlexOTO's Hardware Diagram.

Returning a JSON string

```
1  json_string = """
2  {
3      "ModelNumber": "My Instrument",
4      "SerialNumber": "SN12345",
5      "Inputs": ["IN A", "IN B"]
6  } """
7
8  print(json_string)
9  print("DONE")
10
```

Or, you could make Python variables and a list and convert them to JSON using `json.dumps` method:

Returning a JSON string from Conversion

```
1  mn = 'My Instrument'
2  sn = 'SN12345'
3  InputNames = ['IN A', 'IN B']
4  description = json.dumps({'ModelNumber':mn, 'SerialNumber':sn, 'Inputs':InputNames})
5  print(description)
6  print("DONE")
7
```

The instrument block that this JSON string creates is shown placed on the Hardware Diagram in the following figure. The model number, serial number, and port labels appear on the block.

Figure 15. Instrument Switch Block on the Hardware Diagram

JSON Elements Returned to FlexOTO

FlexOTO expects to find the following elements in the imported JSON string. See the above JSON example.

`ModelNumber` Element

The `ModelNumber` element is a string that names the instrument on FlexOTO's Hardware Diagram. The name that you give is entirely up to you and need not be related to the actual instrument.

```
"ModelNumber": "My Instrument",
```

`SerialNumber` Element

The `SerialNumber` element is a string that is the instrument 's serial number. You can query this value from the instrument and then insert the name into the JSON string.

```
"SerialNumber": "Z1234",
```

`Inputs` Element

The `Inputs` element lists the names of the instrument's input connectors on which measurements will be performed. These strings label Instrument block connectors on FlexOTO's Hardware Diagram. While not required, the labels specified should match those on the instrument's front panel. FlexOTO sends these input connector names as arguments to the driver's measure command. Refer to *measure Command* on page 44. Your driver will need to translate these strings to the correct SCPI strings for the instrument. Consult the instrument's manual to find the exact strings to use.

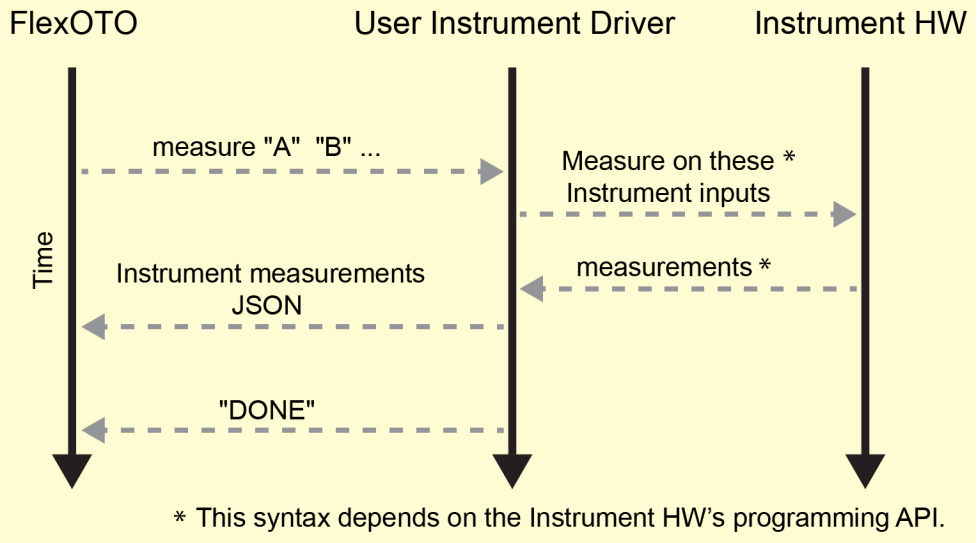
```
"Inputs": ["IN A", "IN B"]
```

measure Command

This command passes to the driver the names of instrument input connectors on which to perform measurements. The driver performs all measurement on each input and returns a JSON results string for all inputs listed.

The list of measurements to be performed is specified within your driver. The driver should translate from the input connector name to the instrument's SCPI commands that are required to select the instrument's input connector and perform the measurements.

Figure 16. Interaction when the `measure` command is sent to the Instrument Driver



Command from FlexOTO

```
measure "input connector" "input connector" ...
```

Command Example

This is an example of a typical argument string:

```
'measure "IN A" "IN B"'
```

Your driver will need to strip "measure " from the string, and create list of connectors without the double quote characters ("). For example:

```
['IN A','IN B']
```

Returned Response to FlexOTO

Returns a JSON string with a list of measurement results. The JSON string is followed by `"DONE"`, on separate lines.

JSON Measure String Returned to FlexOTO

Example JSON Measure String in Python

```python
measurements = """
[
    {
        "Name": "User Meas1",
        "Input": "IN A",
        "Result": 0.000001234,
        "FormattedResult": "1.23 uW"
    },
    {
        "Name": "User Meas2",
        "Input": "IN B",
        "Result": 0.000001526,
        "FormattedResult": "1.53 uW"
    }
] """

print(error_messages)  # if any
print(measurements)
print("DONE")

```

JSON Elements Returned to FlexOTO

For each measurement, FlexOTO expects to find the following elements returned in the imported JSON string.

***Name** Element*

The value of the `Name` element is a string that names the measurement. The name will be displayed on FlexOTO's **Job Results** panel.

```
"Name": "User Meas1",
```

`Input` Element

The value of the `Input` element is a string that names the instrument's measurement input connector.

```
"Input": "IN A",
```

`Result` Element

The `Result` element returns the measurement result (floating-point number). The measurement will be displayed on FlexOTO's Job Results panel.

```
"Result": 0.000001234,
```

`FormattedResult` Element

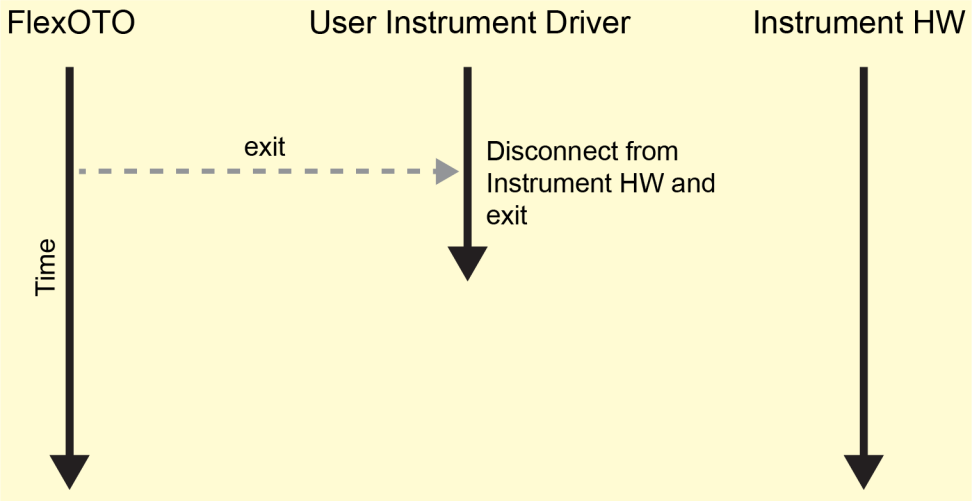
The `FormattedResult` element returns the measurement result with the value formatted to include units of measure. The measurement (formatted) will be displayed on FlexOTO's Job Results panel. The formatting should be performed by the driver.

```
"FormattedResult": '{0:.2f} uW'.format(0.000001234)
```

exit Command

This command instructs the instrument driver to disconnect from the instrument hardware and end the driver process. The command is sent from FlexOTO when the user clicks **Exit** in the **Instrument Connection Setup** dialog or the `:INSTrument:DISConnect` command is sent to FlexOTO. This command does not expect a return value.

Figure 17. Interaction when the `exit` command is sent to the Instrument Driver



Command from FlexOTO

```
exit
```

Returned Response to FlexOTO

This command does not provide a return value.

Example Instrument Driver

This is an example of a instrument driver written in Python. The driver connects to a Keysight 8163/4/6-series mainframe that has an 81634A optical power meter module installed. The driver connects to the instrument using the LAN port with a VISA address. The VISA address must be passed to the driver by entering the address using the Instrument Connection Setup dialog. Refer to *To run your Instrument driver* on page 11. For example, `'TCPIP0::MYINST::inst2::INSTR'`.

DriverInstrument.py

```
#****************************************************************************
#    MIT License
#    Copyright(c) 2023 Keysight Technologies
#    Permission is hereby granted, free of charge, to any person obtaining a copy
#    of this software and associated documentation files (the "Software"), to deal
#    in the Software without restriction, including without limitation the rights
#    to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
#    copies of the Software, and to permit persons to whom the Software is
#    furnished to do so, subject to the following conditions:
#    The above copyright notice and this permission notice shall be included in all
#    copies or substantial portions of the Software.
#    THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
#    IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
#    FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
#    AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
#    LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
#    OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
#    SOFTWARE.
#****************************************************************************

import sys
import json
import pyvisa

DONE = 'DONE'
InputNames = ['In']
```

```
27  Model = 'UNKNOWN'
28  Serial = 'UNKNOWN'
29
30  # FlexOTO is listening to the standard output
31  def send_to_FlexOTO(message: str):
32    print(message)
33
34
35  # Check the error queue
36  def check_error(inst: pyvisa.Resource) -> tuple[int, str]:
37     error = inst.query('SYST:ERR?').strip()
38     sections = error.split(',', 1)
39     return (int(sections[0]), sections[1].strip('"'))
40
41
42  # Open connection to instrument
43  def connect(visaAddress: str) -> pyvisa.Resource:
44     try:
45        rm = pyvisa.ResourceManager()
46        return rm.open_resource(visaAddress)
47     except:
48        send_to_FlexOTO(f'Failed to connect to "{visaAddress}"')
49        return None
50
51
52  # Check if this is a supported instrument.
53  def validate(inst: pyvisa.Resource) -> bool:
54
55     if inst is None: return False
56
57     # Make sure we are connected to a supported Lightwave Mainframe.
58     mainframeIdn = inst.query('*IDN?')
59     sections = mainframeIdn.split(',')
60     if len(sections) < 3:
61        send_to_FlexOTO('Please connect to a 816x mainframe with a power meter in Slot
   1.')
```

```
62          return False
63
64      mfManufacturer = sections[0].upper().strip()
65      if not (mfManufacturer.startswith('KEYSIGHT') or
66              mfManufacturer.startswith('AGILENT') or
67              mfManufacturer.startswith('HEWLETT') or
68              mfManufacturer.startswith('HP')):
69          send_to_FlexOTO('Please connect to a 816x mainframe with a power meter in Slot
    1.')
70          return False
71
72      mfModel = sections[1].strip()
73      if not mfModel.startswith('816'):
74          send_to_FlexOTO('Please connect to a 816x mainframe with a power meter in Slot
    1.')
75          return False
76
77      # Found an 816x mainframe (e.g. 8163B)
78      # Make sure a supported power meter is in Slot 1.
79      moduleIdn = inst.query('SLOT1:IDN?')
80      sections = moduleIdn.split(',')
81      if len(sections) < 3:
82          send_to_FlexOTO('Please install the power meter module in Slot 1')
83          return False
84
85      manufacturer = sections[0].upper().strip()
86      if manufacturer.startswith('KEYSIGHT'):
87          manufacturer = 'Keysight'
88      elif manufacturer.startswith('AGILENT'):
89          manufacturer = 'Agilent'
90      elif manufacturer.startswith('HEWLETT') or manufacturer.startswith('HP'):
91          manufacturer = 'HP'
92      else:
93          send_to_FlexOTO('Unsupported manufacturer: ' + manufacturer)
94          return False
95
96      model = sections[1].strip()
```

```
97      if not model.startswith('8163'):
98          send_to_FlexOTO('Unrecognized power meter model: ' + model)
99          return False
100
101     # Found an 8163x power meter (e.g. 81634A)
102     # Save the model and serial numbers for later.
103     global Model, Serial
104     Model = manufacturer + " " + model
105     Serial = sections[2].strip()[-5:] # Get last 5 of serial number
106     return True
107
108
109 # Do the initial setup of the instrument
110 def initialize(inst: pyvisa.Resource):
111
112     # Set timeout to 10 sec. This should work for all commands except zeroing.
113     inst.timeout = 10000
114
115     # Make sure that the reference is not used.
116     inst.write('SENS1:CHAN1:POW:REF:STATE 0')
117
118     # Turn auto range on.
119     inst.write('SENS1:CHAN1:POW:RANGE:AUTO 1')
120
121     # Change the power unit to Watt.
122     inst.write('SENS1:CHAN1:POW:UNIT W')
123
124     # Set the averaging time for measuring to 0.5s.
125     inst.write('SENS1:CHAN1:POW:ATIME 0.5')
126
127     # Turn continuous measuring off.
128     inst.write('INIT1:CHAN1:CONT 0')
129
130
131 # Blocks until a command comes in from FlexOTO,
```

```
# and then extracts the command and arguments strings.
def wait_for_input() -> tuple[str, list[str]]:

   # Commands come from the standard input.
   rawInput = input()
   sections = rawInput.split('"')

   items = []
   for s in sections:
      s = s.strip()
      if s: items.append(s)

   command = items.pop(0)
   args = items

   return (command, args)


# Sends the JSON description of the instrument to FlexOTO.
def get_description():

   # Use the model and serial numbers determined earlier
   desc = json.dumps({ 'ModelNumber': Model,
                       'SerialNumber': Serial,
                       'Inputs': InputNames,
                       'MeasurementTimeoutSeconds': 10 })

   send_to_FlexOTO(desc)
   send_to_FlexOTO(DONE)


# Measure the active inputs and send the results to FlexOTO.
def measure(inst: pyvisa.Resource, activeInputs: list[str]):

   # Clear error queue
   inst.write('*CLS')
```

```
168
169     measList = []
170     for inputName in activeInputs:
171         inputNum = InputNames.index(inputName) + 1
172
173         # Make an average power measurement on this channel.
174         avgPower = float(inst.query('READ1:CHAN{0}:POW?'.format(inputNum)))
175
176         # Start a dictionary to describe a measurement of Average Power on this input.
177         measurement = { 'Name': 'Average Power', 'Input': inputName }
178
179         (errorCode, errorMsg) = check_error(inst)
180
181         if errorCode == 0:
182             measurement['Result'] = avgPower
183             measurement['FormattedResult'] = '{0:.2f} \u03BCW'.format(avgPower * 1e6) #
    Format in uW
184         else:
185             # Report the error
186             measurement['Result'] = float("NaN") # NaN ("not a number") indicates an
    invalid result
187             measurement['FormattedResult'] = errorMsg
188
189         # Add the result to the measurements list.
190         measList.append(measurement)
191
192     measJson = json.dumps({'Measurements': measList}, allow_nan=True)
193
194     send_to_FlexOTO(measJson)
195     send_to_FlexOTO(DONE)
196
197
198
199 # Program begins here
200 if len(sys.argv) < 2:
201     send_to_FlexOTO("Please provide the instrument's VISA address
    (TCPIP0::HOSTNAME::inst0::INSTR) in the Command Line Arguments.")
```

```
202 else:
203
204    # Get the instrument's VISA address from the command line args.
205    visaAddress = sys.argv[1]
206    inst = connect(visaAddress)
207
208    # Check if this is a valid instrument.
209    if validate(inst):
210
211       # Do initial configuration of instrument setup.
212       initialize(inst)
213
214 # Connection and initial setup is done.
215 send_to_FlexOTO(DONE)
216
217 ########## Main loop ##########
218
219 # Loop until FlexOTO sends us 'exit'.
220 exit = False
221 while not exit:
222    (command, args) = wait_for_input()
223
224    if command == 'exit':
225       exit = True
226
227    elif command == 'get_description':
228       get_description()
229
230    elif command == 'measure':
231       measure(inst, args)
232
233
234 # Exiting...
235
```