Embedding a debugger in your PyTango device

Anton Joubert (SARAO)
Matteo Di Carlo (INAF)

Is it possible to debug a device?

Yes and we have tested the following options:

- The pdb debugger
- Remote debugging with debugpy (vscode)

Note that there are other options.

Pdb debugger

Pdb is a python module which defines an interactive source code debugger

It does not have the ability to attach to a running program

So to use it we must go into the container, kill the running device server and restart it with the debugger module

Pdb debugger

Note that by default the pdb debugger works only with the main thread.

```
tango@centralnode-01-0:/app$ python3 -m pdb /usr/local/bin/CentralNodeDS 01
> /usr/local/bin/CentralNodeDS(3)<module>()
-> import re
(Pdb) break /usr/local/lib/python3.7/dist-packages/centralnode/central_node.py:127
Breakpoint 1 at /usr/local/lib/python3.7/dist-packages/centralnode/central_node.py:127
(Pdb) continue
Downloading https://hpiers.obspm.fr/iers/bul/bulc/Leap_Second.dat
|==========================================================================================
1|2021-03-04T10:53:03.600Z|INFO|MainThread|write_loggingLevel|base_device.py#1001|tango-device:
1|2021-03-04T10:53:03.601Z|INFO|MainThread|update_logging_handlers|base_device.py#305|tango-dev
1|2021-03-04T10:53:03.602Z|DEBUG|MainThread|_init_logging|base_device.py#691|tango-device:ska_m
1|2021-03-04T10:53:03.669Z|INFO|MainThread|exit|core.py#131||Finished processing state UNINITIA
1|2021-03-04T10:53:03.669Z|INFO|MainThread|enter|core.py#125||Finished processing state INIT en
1|2021-03-04T10:53:03.669Z|INFO|MainThread|_update_state|base_device.py#821|tango-device:ska_mi
1|2021-03-04T10:53:03.670Z|INFO|MainThread|callbacks|core.py#1084||Executed callback '<bound me
1|2021-03-04T10:53:03.670Z|DEBUG|MainThread|do|base_device.py#616|tango-device:ska_mid/tm_centr
1|2021-03-04T10:53:03.671Z|DEBUG|MainThread|do|base_device.py#630|tango-device:ska_mid/tm_centr
1|2021-03-04T10:53:03.671Z|INFO|MainThread|do|base_device.py#635|tango-device:ska_mid/tm_centra
> /usr/local/lib/python3.7/dist-packages/centralnode/central_node.py(127)do()
-> self.logger.info("Device initialisating...")
(Pdb) continue
1|2021-03-04T10:53:09.141Z|INFO|MainThread|do|central_node.py#127|tango-device:ska_mid/tm_centr
1|2021-03-04T10:53:09.142Z|DEBUG|MainThread|do|central_node.py#139|tango-device:ska_mid/tm_cent
1|2021-03-04T10:53:09.143Z|INFO|MainThread|do|central_node.py#172|tango-device:ska_mid/tm_centr
1|2021-03-04T10:53:09.143Z|INFO|MainThread|_call_do|commands.py#212|tango-device:ska_mid/tm_cen
1|2021-03-04T10:53:09.145Z|INFO|MainThread|exit|core.py#131||Finished processing state INIT exi
1|2021-03-04T10:53:09.146Z|INFO|MainThread|enter|core.py#125||Finished processing state OFF ent
1|2021-03-04T10:53:09.146Z|INFO|MainThread|_update_state|base_device.py#821|tango-device:ska_mi
1|2021-03-04T10:53:09.147Z|INFO|MainThread|callbacks|core.py#1084||Executed callback '<bound me
Ready to accept request
```

Pdb debugger: Threading

```
tango@centralnode-01-0:/app$ python3 -m pdb /usr/local/bin/CentralNodeDS 01
> /usr/local/bin/CentralNodeDS(3)<module>()
-> import re
(Pdb) break /usr/local/lib/python3.7/dist-packages/centralnode/central_node.py:314
Breakpoint 1 at /usr/local/lib/python3.7/dist-packages/centralnode/central_node.py:314
(Pdb) continue
> /usr/local/bin/CentralNodeDS(9)<module>()
```

```
# -*- coding: utf-8 -*-
import re
import sys
from centralnode.central_node import main
if __name__ == '__main__':
    sys.argv[0] = re.sub(r'(-script\.pyw|\.exe)?$', '', sys.argv[0])
    import pdb; pdb.Pdb(nosigint=True).set_trace()
    sys.exit(main())
```

```
def StartUpTelescope(self):
    """
    This command invokes SetOperateMode() command on DishL
    SdpMasterLeafNode and SubarrayNode and sets the Centra
    """
    import pdb; pdb.Pdb(nosigint=True).set_trace()
    handler = self.get_command_object("StartUpTelescope")
    (result_code, message) = handler()
    return [[result_code], [message]]
```

```
7Z|INFO|MainThread|write_loggingLevel|base_device.py#1001|tango-dev
7Z|INFO|MainThread|update_logging_handlers|base_device.py#305|tango
7Z|DEBUG|MainThread|_init_logging|base_device.py#691|tango-device:s
5Z|INFO|MainThread|exit|core.py#131||Finished processing state UNIN
5Z|INFO|MainThread|enter|core.py#125||Finished processing state INI
5Z|INFO|MainThread|_update_state|base_device.py#821|tango-device:sk
5Z|INFO|MainThread|callbacks|core.py#1084||Executed callback '<boun
1|2021-03-04T11:11:08.077Z|DEBUG|MainThread|do|base_device.py#616|tango-device:ska_mid/tm_c
4T11:11:08.077Z|DEBUG|MainThread|do|base_device.py#630|tango-device:ska_mid/tm_c
4T11:11:08.077Z|INFO|MainThread|do|base_device.py#635|tango-device:ska_mid/tm_ce
4T11:11:08.077Z|INFO|MainThread|do|central_node.py#127|tango-device:ska_mid/tm_c
4T11:11:08.077Z|DEBUG|MainThread|do|central_node.py#139|tango-device:ska_mid/tm_
4T11:11:08.078Z|INFO|MainThread|do|central_node.py#172|tango-device:ska_mid/tm_c
4T11:11:08.078Z|INFO|MainThread|_call_do|commands.py#212|tango-device:ska_mid/tm
4T11:11:08.078Z|INFO|MainThread|exit|core.py#131||Finished processing state INIT
4T11:11:08.078Z|INFO|MainThread|enter|core.py#125||Finished processing state OFF
4T11:11:08.078Z|INFO|MainThread|_update_state|base_device.py#821|tango-device:sk
1|2021-03-04T11:11:08.078Z|INFO|MainThread|callbacks|core.py#1084||Executed callback '<boun
Ready to accept request
1|2021-03-04T11:11:13.042Z|DEBUG|Dummy-1|log_stream|log4tango.py#138|tango-device:ska_mid/t
> /usr/local/lib/python3.7/dist-packages/centralnode/central_node.py(312)StartUpTelescope()
-> handler = self.get_command_object("StartUpTelescope")
(Pdb)
```

Remote debugging

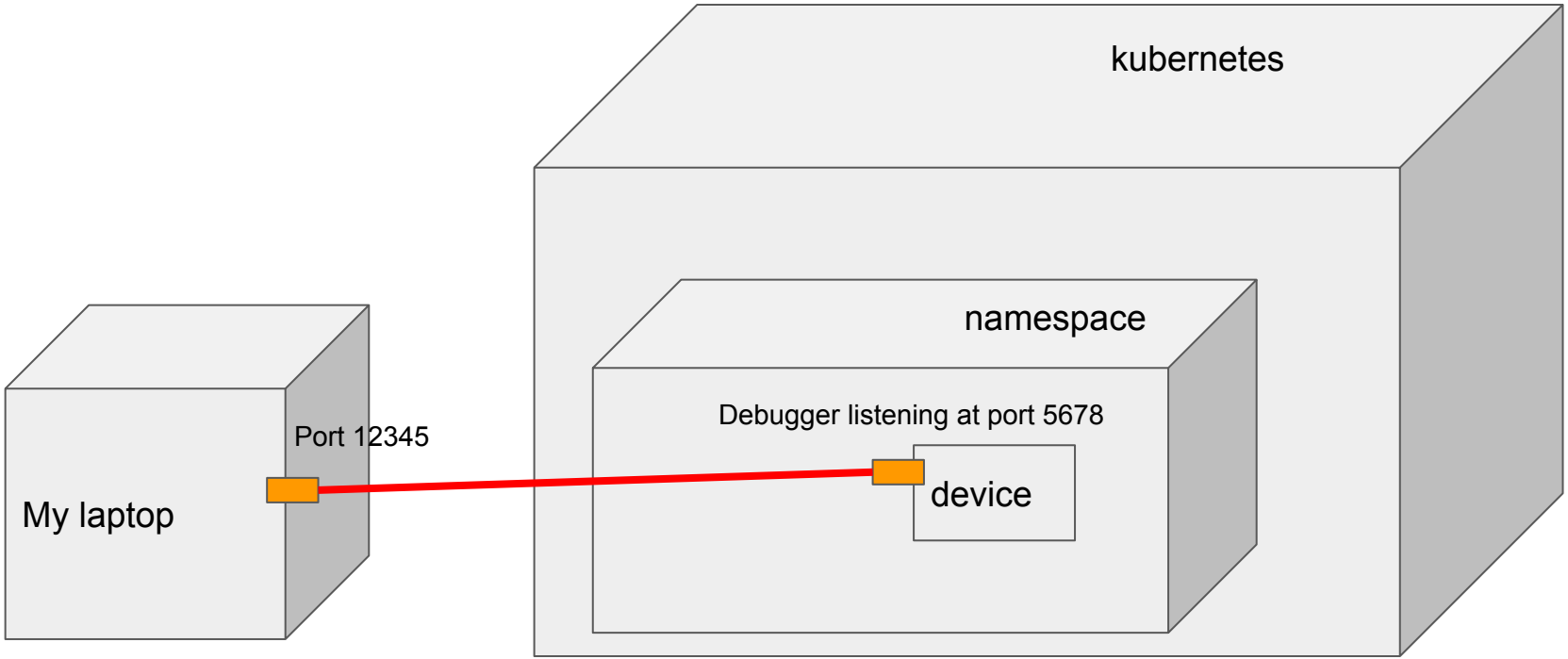
The remote debugging requires a debugger listening from a port in the remote machine (in our case a container)

If the debugger is listening to a port, we need to port-forward it to a local port (feature of kubernetes)

```
$ kubectl port-forward centralnode-01-0 12345:5678 -n
ci-debug-matt
```

With non-kubernetes machines, ssh could do something similar.

As if the debugger listen to a local port of my laptop



debugpy library

vscode extensions uses this python library for remote debugging:
https://github.com/microsoft/debugpy

It is an adapter of the pydevd used in PyCharm
https://github.com/fabioz/PyDev.Debugger/

How it works:

- CLI: `python -m debugpy --listen localhost:5678 mydevice.py`
- From code:
  - `import debugpy`
  - `debugpy.listen(5678)`

Debug this thread method

A standard TANGO Device server does not use Python threads so most method calls are not debuggable unless we make them aware of the debugger.

Threads per device server? 8 + *num polling* + *num connected clients*

Debug this thread method

Same situation of the pdb debugger (debugpy uses the standard pdb library).

In every method we want to debug we must add the following line of code:

```
debugpy.debug_this_thread()
```

https://github.com/microsoft/debugpy/wiki/API-Reference#debug_this_thread

Makes the debugger aware of the current thread, and start tracing it. Must be called on any background thread that is started by means other than the usual Python APIs (i.e. the threading module), in order for breakpoints to work on that thread.

Using vscode: `.vscode/launch.json`

We must configure the vscode to work with the remote debugger

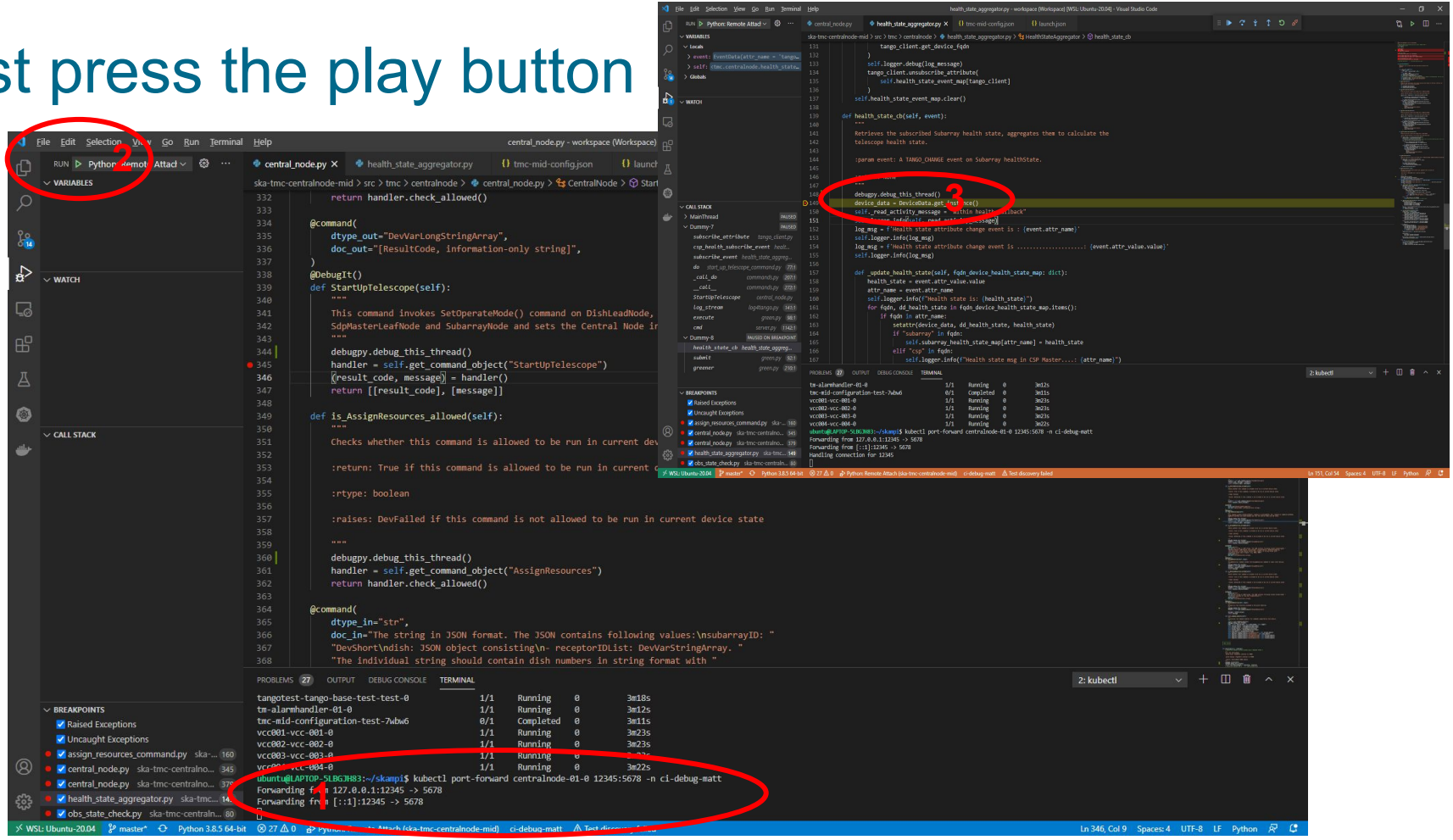
```
{
      "configurations": [
      {
            "name": "Python: Remote Attach",
            "type": "python",
            "request": "attach",
            "connect": {
            "host": "127.0.0.1",
            "port": 12345,
            "justMyCode": false,
            },
            "pathMappings": [
            {
                  "localRoot": "${workspaceFolder}",
                  "remoteRoot": "/app"
            }
            ]
      }
      ]
}
```

PyCharm?

PyCharm debugger probably won’t work for Kubernetes, as PyCharm hosts the debugger service.

It could work in environments that allow a connection to be established from the device server to the PyCharm host.

Just press the play button



Adding to the `SKABaseDevice`?

src > ska_tango_base > base_device.py > SKABaseDevice > DebugDevice

```
1725
1726        @command(
1727            dtype_out="DevUShort",
1728            doc_out="The TCP port the debugger is listening on."
1729        )
1730        def DebugDevice(self):
1731            """
1732            Enables remote debugging of this device.
1733
1734            Starts the ``debugpy`` debugger listening for remote connections
1735            (via Debugger Adaptor Protocol), and patches all methods so that
1736            they can be debugged.
1737
1738            If the debugger is already listening, additional e
1739            command will trigger a breakpoint.
1740            """
1741            command = self.get_command_object("DebugDevice")
1742            return command()
```

1. Add new command: `DebugDevice`

2. Patches (almost) all the methods

src > ska_tango_base > base_device.py > SKABaseDevice > DebugDeviceCommand > patch_metho

```
1702        def patch_method_for_debugger(self, name, method):
1703            """Ensure method calls trigger the debugger.
1704
1705            Most methods in a device are executed by calls from threads spawned
1706            by the cppTango layer.  These threads are not known to Python, so
1707            we have to explicitly inform the debugger about them.
1708            """
1709
1710            def debug_thread_wrapper(orig_method, *args, **kwargs):
1711                debugpy.debug_this_thread()
1712                return orig_method(*args, **kwargs)
1713
1714            patched_method = partial(debug_thread_wrapper, method)
1715            setattr(self.target, name, patched_method)
```

See source code on gitlab

Adding to the `SKABaseDevice`?

src > ska_tango_base > base_device.py > SKABaseDevice > DebugDevice

```
1725
1726        @command(
1727            dtype_out="DevUShort",
1728            doc_out="The TCP port the debugger is listening on."
1729        )
1730        def DebugDevice(self):
1731            """
1732            Enables remote debugging of this device.
1733
1734            Starts the ``debugpy`` debugger listening for remote connections
1735            (via Debugger Adaptor Protocol), and patches all methods so that
1736            they can be debugged.
1737
1738            If the debugger is already listening, additional e
1739            command will trigger a breakpoint.
1740            """
1741            command = self.get_command_object("DebugDevice")
1742            return command()
```

1. Add new command: `DebugDevice`

2. Patches (almost) all the methods

src > ska_tango_base > base_device.py > SKABaseDevice > DebugDeviceCommand > patch_metho

```
1702        def patch_method_for_debugger(self, name, method):
1703            """Ensure method calls trigger the debugger.
1704
1705            Most methods in a device are executed by calls from threads spawned
1706            by the cppTango layer.  These threads are not known to Python, so
1707            we have to explicitly inform the debugger about them.
1708            """
1709
1710            def debug_thread_wrapper(orig_method, *args, **kwargs):
1711                debugpy.debug_this_thread()
1712                return orig_method(*args, **kwargs)
1713
1714            patched_method = partial(debug_thread_wrapper, method)
1715            setattr(self.target, name, patched_method)
```

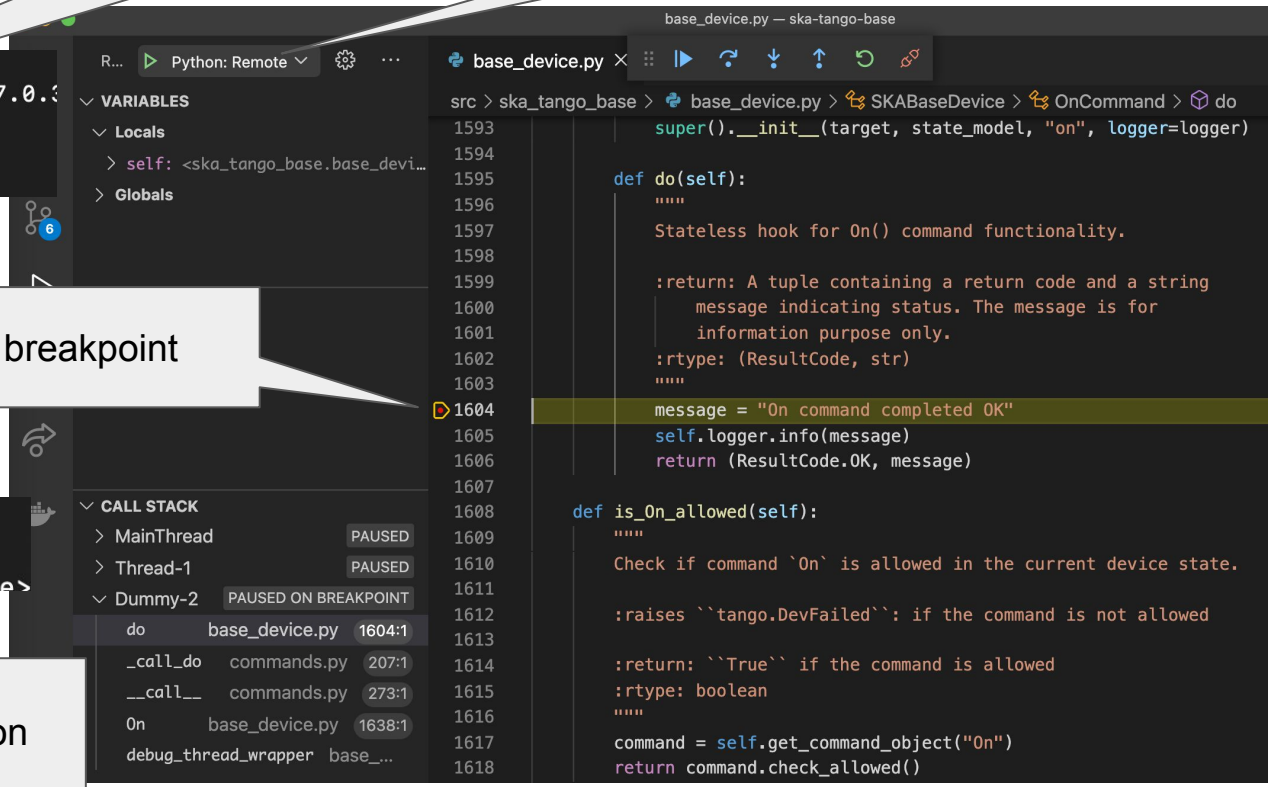
See source code on gitlab

Example

1. Use `DeviceProxy` to enable

2. Press play

```
>>> import tango
>>> dp = tango.DeviceProxy("tango://172.17.0.3
>>> dp.DebugDevice()
5678
>>>
```



3. Set your breakpoint

```
>>> dp.On()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
```

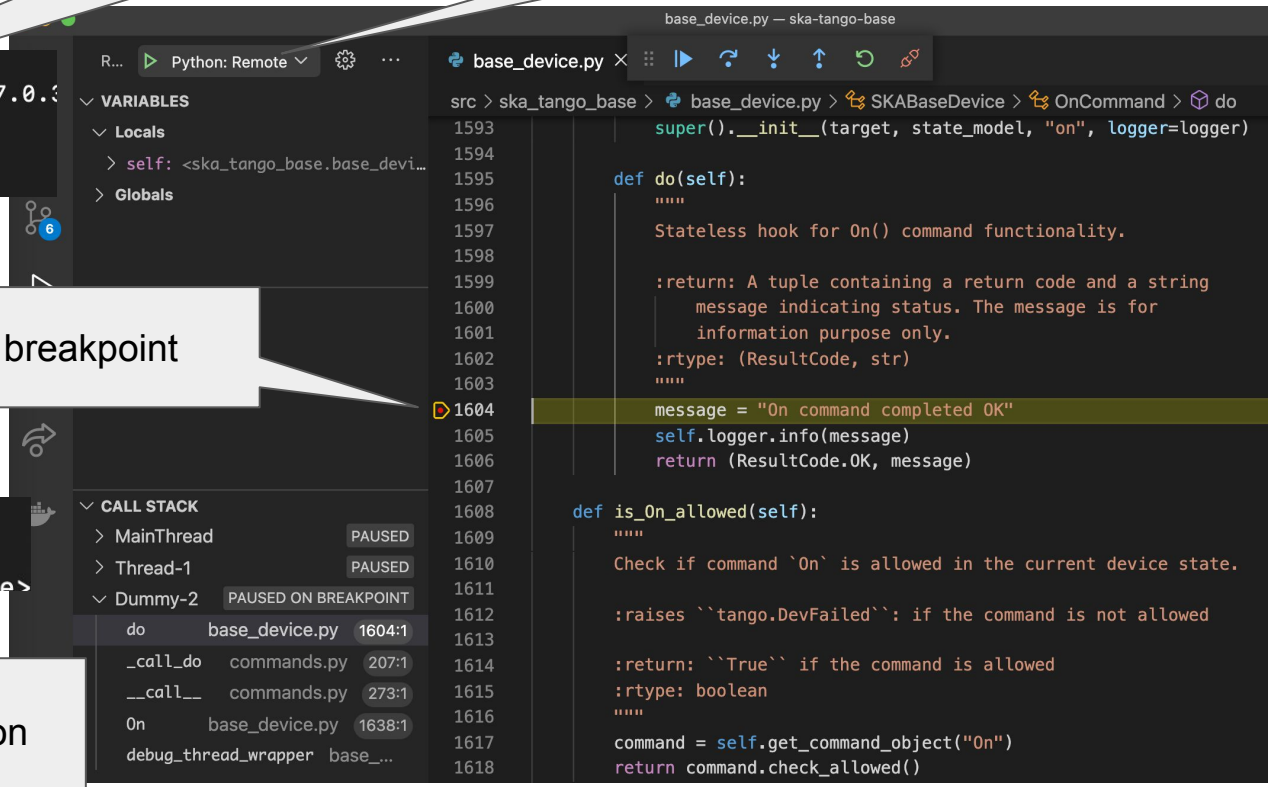
4. Cause execution (command may timeout on client side)

Example

1. Use `DeviceProxy` to enable

2. Press play

```
>>> import tango
>>> dp = tango.DeviceProxy("tango://172.17.0.3
>>> dp.DebugDevice()
5678
>>>
```



3. Set your breakpoint

```
>>> dp.On()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
```

4. Cause execution (command may timeout on client side)

Conclusion

- pdb debugger from console (no attach option)
- debugpy and vscode
- Listening debugpy session:
  - May affect performance
  - Is a security hole
  - Cannot be turned off again

Both pdb and debugpy require a trace for non-Python threads.

SKABaseDevice source code on gitlab