ska$_{p}act_{t}angoDocumentation$

Release 0.0.1-alpha

SARAO CAM Team

Aug 16, 2021

HOME

This repository contains a Python implementation for Pact, adapted to test the interactions between Tango devices. Pact is a specification for Consumer Driven Contracts Testing. For further information about Pact project, contracts testing, pros and cons and useful resources please refer to the Pact website.

CHAPTER

ONE

REQUIREMENTS

The system used for development needs to have Python 3 and `pip` installed.

CHAPTER

TWO

INSTALL

2.1 From source

- Clone the repo

```
git clone git@gitlab.com:ska-telescope/ska-pact-tango.git
```

- Install requirements

```
python3 -m pip install -r requirements.txt
```

- Install the package

```
python3 -m pip install .
```

2.2 From the Nexus PyPI

```
python3 -m pip install ska-pact-tango --extra-index-url https://artefact.skao.int/
↪repository/pypi-internal/simple
```

CHAPTER

THREE

TESTING

- Install the test requirements

```
python3 -m pip install -r requirements-test.txt
```

- Run the tests

```
tox
```

- Lint

```
tox -e lint
```

CHAPTER

FOUR

WRITING DOCUMENTATION

The documentation generator for this project is derived from SKA's SKA Developer Portal repository

The documentation can be edited under `./docs/src`

4.1 Build the documentation

- Install the test requirements

```
python3 -m pip install -r requirements-test.txt
```

- Build docs

```
tox -e docs
```

The documentation can then be consulted by opening the file `./docs/build/html/index.html`

4.1.1 Pact with Tango

This document explains the reasoning behind Pact testing and how it applies to Tango devices. The devices used are based on those defined in the *multi_device_proxy* folder examples. See *examples/multi_device_proxy* folder for device implementations as well as test examples.

Terminology

First, some Pact terms used below.

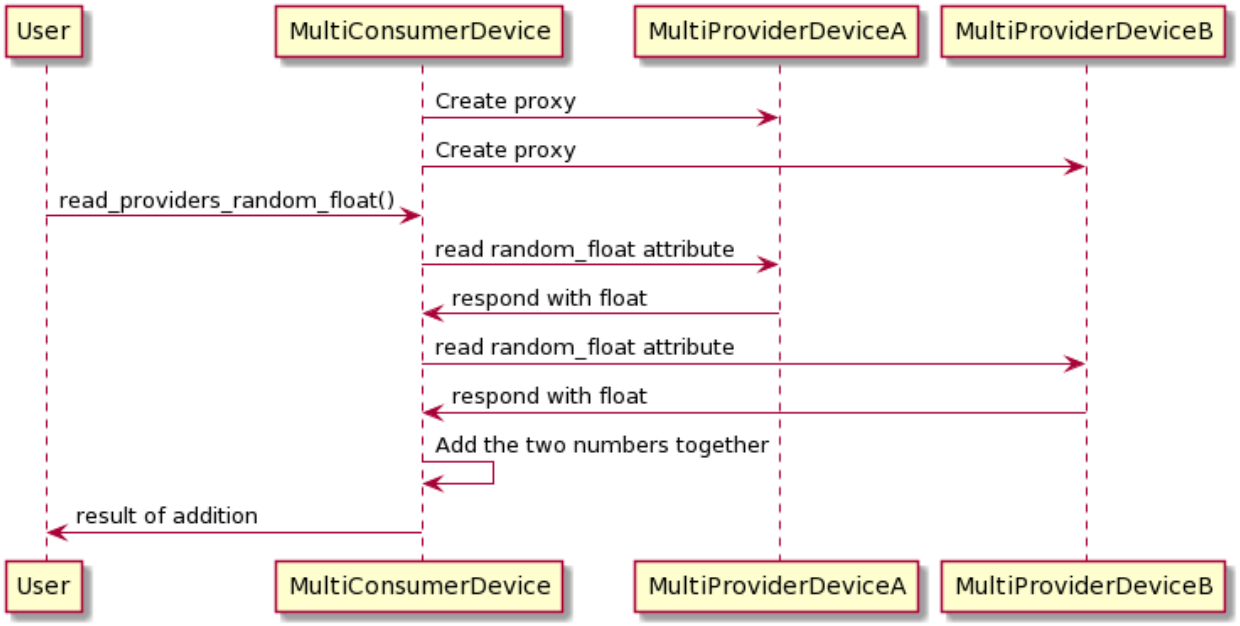
- **Pact** Defines the interactions between the consumers and provider(s).
- **Consumer** The Tango device under test that itself creates a proxy to another Tango device (provider) and interacts with it. I.e the consumer is a client of the provider. The consumer (client) could create multiple proxies to several providers (servers).
- **Provider** The Tango device that the consumer interacts with. There could be several providers that the consumer connects to.
- **Interaction** A request-response pair. The request from the consumer and the response from the provider.
- **Mocked Provider** The actual running provider is replaced with a mocked provider and it's behavior is defined by interactions.
- **Pact file (contract)** The serialized JSON file that describes a Pact between consumer and provider(s). This is the *contract* as defined in the Pact literature.

- **Pact verification** The process of ensuring the validity of a Pact file by running and verifying the interactions against the actual provider.
- **Provider state** The state that the provider should be in prior to doing the pact verification. This is not limited to the State attribute of the provider, but could include attribute values.

Pact test process

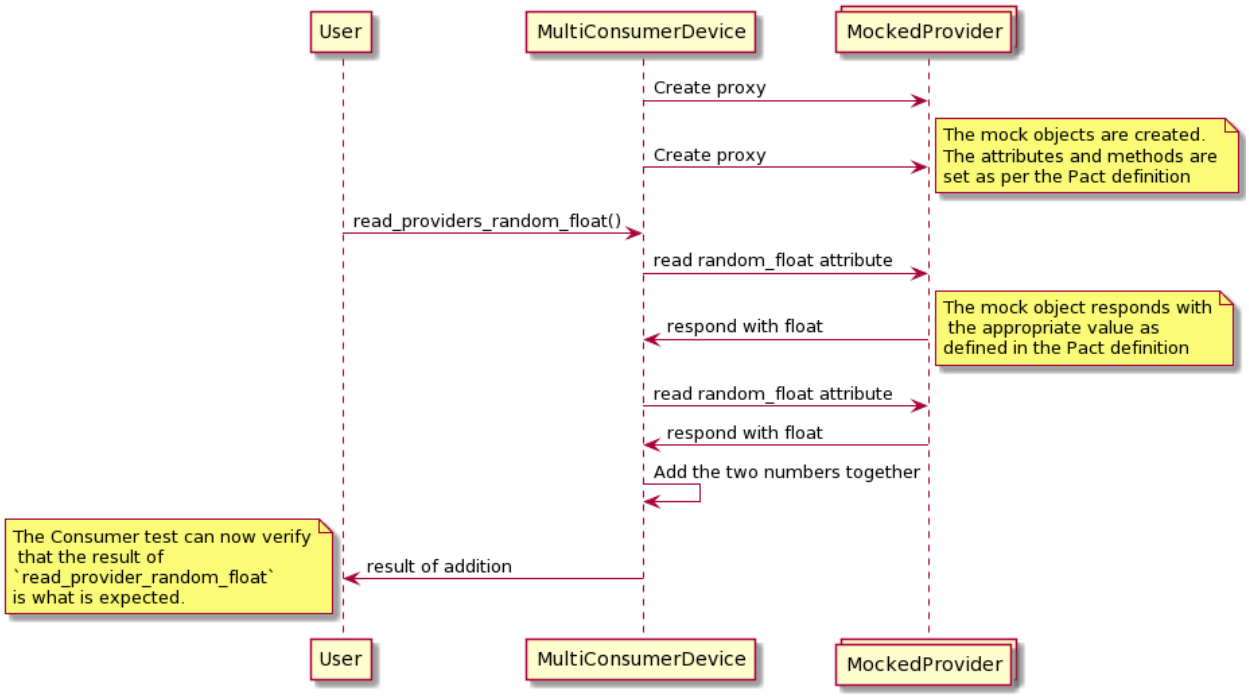
Normal operation

In our example below the MultiConsumerDevice creates two proxies to MultiProviderDeviceA and MultiProviderDeviceB, respectively. The consumer then gets a command *read_providers_random_float*. The Consumer then reads the attribute *random_float* from each proxy. The numbers are added together and returned to the user as the result of *read_providers_random_float*.



Consumer side test

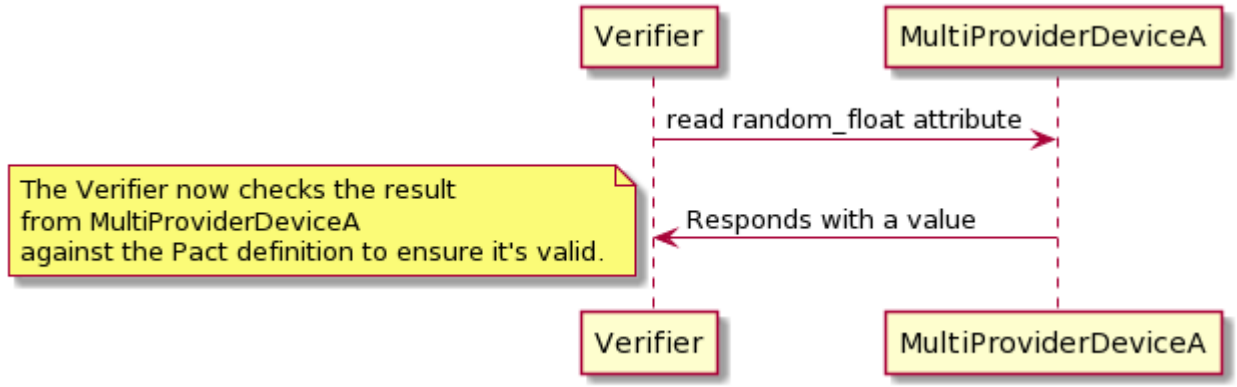
Here we see that both providers have now been replaced by the mocked provider. The result of reading *random_float* from the providers are now determined by the Pact file. The Consumer is oblivious to the fact that the real providers don't exist. Now the result of command *read_providers_random_float* on the consumer can be verified since the response from the providers are known.



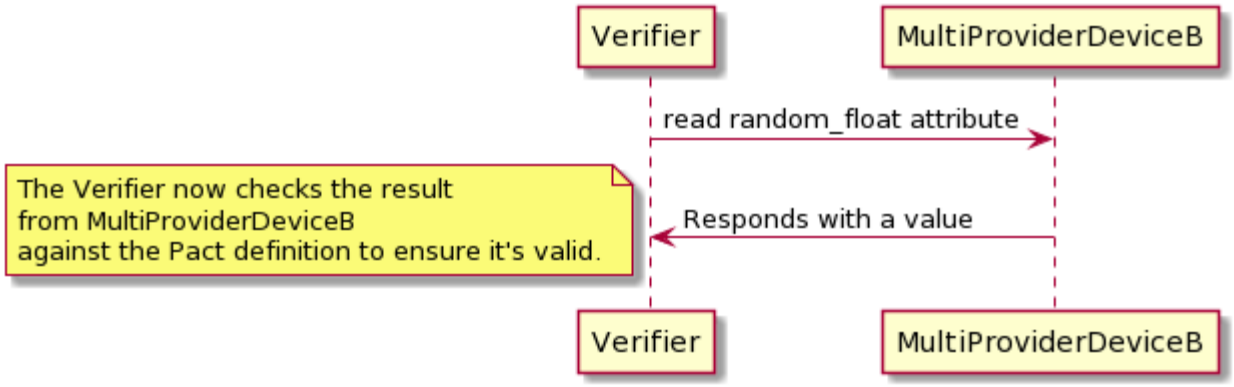
Provider side test

To ensure that the provider responds as the pact expects we need to verify the interactions.

ProviderA



ProviderB



4.1.2 Examples

The examples are split into three directories:

- `charts`
  - Contains the helm charts to deploy all the sample devices
- `single_device_proxy`
  - The Tango device under test (Consumer) has a single `tango.DeviceProxy` to another Tango device (Provider)
  - Includes devices and sample tests
- `multi_device_proxy`
  - The Tango device under test (Consumer) has multiple `tango.DeviceProxy` to Tango devices (Providers)
  - Includes devices and sample tests

Pact testing Tango devices (Consumer side)

A simple test sample:

```
from tango.test_context import MultiDeviceTestContext, DeviceTestContext
from tango import DeviceAttribute


from consumer_device import ConsumerDevice
from ska_pact_tango import Consumer, Provider, Interaction

devices_info = [
    {"class": ConsumerDevice, "devices": ({"name": "test/consumer/2", "properties": {}
↪},)}
]


def test_consumer_to_provider_attribute_read():
    """Test the attribute read"""
    device_attribute_response = DeviceAttribute()
    device_attribute_response.value = 99.99
```

(continues on next page)

(continued from previous page)

```
    pact = Consumer("test/consumer/1", consumer_cls=ConsumerDevice).has_pact_with(
        providers=[
            Provider("test/provider/1").add_interaction(
                Interaction()
                .given("The provider is in Init State", "Init")
                .upon_receiving("A read attribute request for the attribute random_
↪float")
                .with_request("read_attribute", "random_float")
                .will_respond_with(DeviceAttribute, device_attribute_response)
            )
        ]
    )

    with pact:
        with MultiDeviceTestContext(devices_info, process=True) as context:
            consumer = context.get_device("test/consumer/2")
            assert consumer.read_provider_random_float() == device_attribute_response.
↪value
```

For more examples refer to `ska-pact-tango/examples`.

Tango to `with_request` mapping

Tango syntax	with_request
provider_device.random_float = 5.0	with_request("attribute", "random_float", 5.0)
provider_device.write_attribute("random_float", 5.0)	with_request("method", "write_attribute", "random_float", 5.0)
	with_request("write_attribute", "random_float", 5.0)
provider_device.random_float	with_request("attribute", "random_float")
provider_device.read_attribute("random_float")	with_request("read_attribute", "random_float")
provider_device.SomeCommand()	with_request("command", "SomeCommand")
provider_device.SomeCommand(1.0)	with_request("command", "SomeCommand", 1.0)
provider_device.command_inout("SomeCommand")	with_request("method", "command_inout", "SomeCommand")
	with_request("command_inout", "SomeCommand")
provider_device.command_inout("SomeCommand", 1.0)	with_request("method", "command_inout", "SomeCommand", 1.0)
	with_request("command_inout", "SomeCommand", 1.0)

Verifying the Pact contract with the provider

Every Pact contract should be verified against a Tango device (Provider) to ensure it's valid.

There's a helper decorator (`verifier`) that exposes the interaction to verify as well as the device name. The verifier decorator takes to location of the pact file and the interaction description as parameters. The description is used to get the interaction that you want to verify as there could be several in the Pact file.

Use the command line tool

Once the package is installed there should be a script named `pact-tango-verify` on the `PATH`. This can be used to verify the Pact contract file.

```
root@ska-pact-tango# pact-tango-verify  -h
usage: pact-tango-verify [-h] [-v] pact_file_path

Verify a Pact file against the provider

positional arguments:
  pact_file_path

optional arguments:
  -h, --help      show this help message and exit
  -v, --verbose   Print test output
```

Example

```
root@ska-pact-tango# pact-tango-verify ./examples/multi_device_proxy/verifier_test.
↪json  -v
Loading Pact from ./examples/multi_device_proxy/verifier_test.json
Done
Checking against provider [test/multiprovider/1]
Checking interaction [A request to run add_int_to_five]
```

Write custom tests

Put the Provider in a desired state

You can add commands (and parameters) to execute as part of `given` and `and_given` in the pact definition. During the `setup_provider` step in your test they will be executed in order against the Provider. This will update the Provider so that it's in the correct state prior to running `verify_interaction`.

See comments in the code sample below.

Example

```
import pytest

import tango

from ska_pact_tango.verifier import verifier
from ska_pact_tango.provider import Provider, Interaction
from ska_pact_tango.consumer import Consumer


pact = Consumer("test/consumer/1", consumer_cls=ProviderDevice).has_pact_with(
        providers=[
            Provider("test/nodb/providerdevice").add_interaction(
                Interaction()
```

(continues on next page)

(continued from previous page)

```
                .given("The provider is in Init State", "Init")
                .and_given("with_arg_command has ran", "with_arg_command", 5)
                .and_given("no_arg_command has ran", "no_arg_command")
                .and_given("Only description")
                .upon_receiving("A request to run add_int_to_five")
                .with_request("command", "add_int_to_five", 5)
                .will_respond_with(int, 10)
            )
        ]
    )

@verifier("/path/to/pact_file.json", "A request to run add_int_to_five")
def verify_provider(*args, **kwargs):
    interaction = kwargs["interaction"]
    device_name = kwargs["device_name"]

    proxy = tango.DeviceProxy(device_name)

    # Here you can run commands on proxy that was not defined in the `given`s
    proxy.Standby()

    # setup_provider runs through the provider_states and executes the commands
    # In our case it will be:
    # - proxy.Init()
    # - proxy.with_arg_command(5)
    # - proxy.no_arg_command()
    interaction.setup_provider(proxy)

    # verify_interaction will execute the `with_request` and compare it to `will_
↪respond_with`
    interaction.verify_interaction(proxy)
```

4.1.3 Device test playground

For setting up an environment where you can run tests and experiment with them, follow the steps below:

Setup

Introduction

Deploy the sample devices by means of a local helm chart in a Kubernetes namespace.

A development container where tests can be edited and executed is also included.

Each container mounts the `ska-pact-tango` directory in `/app/`

Install

1. Ensure you are in the `ska-pact-tango` directory.
2. Ensure that `helm` and `kubectl` has been set up.
3. Deploy the chart of sample devices into a namespace (using `pact` below).

3.1. If you want to use the Jupyter Notebook, set your Ingress hostname

```
kubectl create namespace pact
helm install test ./examples/charts/pact-example   -n pact --dependency-update  --set␣
↪pact_example.hostpath=$(pwd) --set ingress.hostname=<Your hostname>
```

3.2. If you don't have Ingress configured you can omit the hostname. You will not be able to access Jupyter.

```
kubectl create namespace pact
helm install test ./examples/charts/pact-example   -n pact --dependency-update  --set␣
↪pact_example.hostpath=$(pwd)
```

Open a shell to run commands in the container

```
kubectl exec --stdin --tty $(kubectl get pods -o name  | grep pact-example) -c dev-
↪test -n pact -- /bin/bash
```

Run the sample tests:

```
python3 -m pytest /app/ska-pact-tango/examples/sample_tests.py
```

Access a Jupyter Notebook to run arbitrary code

Navigate to:

`http://<your-hostname>/jupyter/`

Password: `pact`

Clean up

```
helm delete test  -n pact
kubectl delete namespaces pact
```

4.1.4 Python Modules

This section details the public API for using the Pact testing package.

Public API Documentation

class ska_pact_tango.pact.**Pact**(*providers*, *consumer_name*, *consumer_class=None*, *pact_dir=None*, *pact_file_name=''*)

Bases: object

Represents a contract between a consumer and provider.

Provides Python context handlers to configure the Pact mock service to perform tests with a Tango device (consumer).

Also generates and writes a pact file.

classmethod from_dict(*pact_dict: dict*)

Generate an instance of Pact from a dictionary

Parameters **pact_dict** (*dict*) – Pact in a dictionary format

Returns an instance of Pact

Return type *Pact*

classmethod from_file(*pact_file_path: str*)

Parses a pact file and returns an instance of Pact

Parameters **pact_file_path** (*str*) – Path to a Pact file

Returns an instance of Pact

Return type *Pact*

get_interactions_for_provider(*provider_name: str*) → list

Fetch all the interactions associated with the device name passed in.

Parameters **provider_name** (*str*) – The provider name

Returns Interactions for the provider

Return type list

to_dict()

Construct a pact dictionary

Returns A dictionary of the Pact

Return type dict

write_pact()

Write the pact to disk

class ska_pact_tango.consumer.**Consumer**(*name*, *consumer_cls=None*)

Bases: object

A Pact consumer.

Use this class to describe the consumer executing commands on the provider and then use *has_pact_with* to create a contract with a specific provider.

has_pact_with(*providers=[]*, *pact_dir=''*, *pact_file_name=''*)

Create a contract between the provider and this consumer.

Parameters

- **providers** – A list of providers that this contract has
- **pact_dir** (*str*) – Directory where the resulting pact files will be written. Defaults to the current directory.

- **pact_file_name** (*str*) – The name of the pact file for this interaction. Defaults to <consumer_name>-pact.json

Return type *pact.Pact*

class ska_pact_tango.provider.**Interaction**

Bases: object

Define an interaction between a consumer and provider

and_given(*provider_state_description: str*, **args*)

Add a description of what state the provider should be in

Parameters

- **provider_state** (*str*) – Description of the provider state, defaults to “”
- **params** (*List[str], optional*) – A list of string paramters, defaults to []

Returns self

Return type *Provider*

execute_request(*request: dict*, *device_proxy: <Mock name='mock.DeviceProxy' id='140704821930384'>*)

Execute a request against a Tango device

Parameters

- **request** (*dict*) – The request to execute
- **device_proxy** (*tango.DeviceProxy*) – The proxy to the provider device

Returns The result of the request

Return type Any

classmethod from_dict(*interaction_dict: dict*)

Returns an instance of Interaction

Parameters **interaction_dict** (*dict*) – Interaction in a dictionary format

Returns instance of Interaction

Return type *Interaction*

given(*provider_state_description: str*, **args*)

Description of what state the provider should be in

Parameters

- **provider_state** (*str*) – Description of the provider state, defaults to “”
- **params** (*List[str], optional*) – A list of string paramters, defaults to []

Returns self

Return type *Provider*

setup_provider(*device_proxy: <Mock name='mock.DeviceProxy' id='140704821930384'>*)

Run through the provider states as defined in the Pact and execute them on the provider

Parameters **device_proxy** (*tango.DeviceProxy*) – The proxy to the provider

to_dict() → dict

Return a dictionary of the Interaction

Returns A dictionary of the Interaction

Return type dict

upon_receiving (*scenario: str = ''*)
Describe the interaction

Parameters **scenario** (*str, optional*) – Description of the interaction, defaults to “”

verify_interaction (*device_proxy: <Mock name='mock.DeviceProxy' id='140704821930384'>*)
Verify the request against the Provider

Parameters **device_proxy** (*tango.DeviceProxy*) – proxy to the provider device

Raises **AssertionError** – If the request and response from the proxy fails

will_respond_with (*response_type*, *response=None*)
Define what the provider should return with.

Parameters

- **response** – The type of the response from the provider
- **response** – The provider response

Return type *Interaction*

with_request (**args*)
Define the request from the consumer

Reference examples

- **Tango attribute write**
  - **Shortform**

    E.g

    ```
    >>> provider_device.random_float = 5.0
    ```

    Use

    ```
    >>> with_request("attribute", "random_float", 5.0)
    ```

  - **Longform**

    E.g

    ```
    >>> provider_device.write_attribute("random_float", 5.0)
    ```

    Use

    ```
    >>> with_request("method", "write_attribute", "random_float", 5.0)
    OR
    >>> with_request("write_attribute", "random_float", 5.0)
    ```

- **Tango attribute read**
  - **Shortform**

    E.g

    ```
    >>> provider_device.random_float
    ```

    Use

```
>>> with_request("attribute", "random_float")
```

- **Longform**

  E.g

  ```
  >>> provider_device.read_attribute("random_float")
  ```

  Use

  ```
  >>> with_request("read_attribute", "random_float")
  ```

- **Tango commands**
  - **Shortform**

    E.g

    ```
    >>> provider_device.SomeCommand()
    ```

    Use

    ```
    >>> with_request("command", "SomeCommand")
    ```

    E.g

    ```
    >>> provider_device.SomeCommand(1.0)
    ```

    Use

    ```
    >>> with_request("command", "SomeCommand", 1.0)
    ```

  - **Longform**

    E.g

    ```
    >>> provider_device.command_inout("SomeCommand")
    ```

    Use

    ```
    >>> with_request("method", "command_inout", "SomeCommand")
    OR
    >>> with_request("command_inout", "SomeCommand")
    ```

    E.g

    ```
    >>> provider_device.command_inout("SomeCommand", 1.0)
    ```

    Use

    ```
    >>> with_request("method", "command_inout", "SomeCommand", 1.0)
    OR
    >>> with_request("command_inout", "SomeCommand", 1.0)
    ```

Parameters

- **req_type** (*string*) – read_attribute or command
- **name** (*string*) – The name of the command or attribute

- **arg** (*Any*) – Any argument to be used in the command

Return type *Interaction*

class ska_pact_tango.provider.**Provider**(*device_name: str*)

Bases: object

A Pact provider.

add_interaction(*interaction:* ska_pact_tango.provider.Interaction)

Add an Interaction

Parameters **interaction** (*Interaction*) – An interaction between the consumer and provider

Returns self

Return type *Provider*

classmethod from_dict(*provider_dict: dict*)

Return an instance of Provider

Parameters **provider_dict** (*dict*) – Provider in a dictionary format

Returns instance of Provider

Return type *Provider*

get_interaction_from_description(*description*) → Optional[*ska_pact_tango.provider.Interaction*]

Returns an interaction that matches the description

Parameters **description** (*[type]*) – [description]

Returns [description]

Return type Optional[ska_pact_tango.Interaction]

to_dict() → dict

Return a dictionary of the Provider

Returns A dictionary of the Provider and it's interactions

Return type dict

ska_pact_tango.verifier.**verifier**(*pact_file_path*, *description*)

A convenience decorator that adds the interaction and device_name to kwargs. The relevant interaction is looked up from the description.

Parameters

- **pact_file_path** (*str*) – The path to the Pact file
- **description** (*str*) – The interaction description, as defined in the Pact file

PYTHON MODULE INDEX

S

INDEX