JSON Wrapper

`JsonWrapper` is a convenience wrapper that transparently encodes/decodes JSON strings on their way to/from a `tango.DeviceProxy`.

Usage

The intent is that operation is transparent and things work as per a normal `DeviceProxy`, except you're now free of the drudgery of typing `json.loads` and `json.dumps` all the time.

Instantiation

You can instantiate via an existing `DeviceProxy` object, or just pass a device name and one will be created for you:

```
# Using a DeviceProxy
cnic_proxy = tango.DeviceProxy("low-cbf/cnic/1")
cnic = JsonWrapper(cnic_proxy)

# Or, straight from the name
cnic = JsonWrapper("low-cbf/cnic/1")
```

Attributes & Commands using JSON are automatically detected if their description includes "JSON". Optional parameters can add Attributes & Commands that are not picked up by the automatic detection logic:

```
cnic = JsonWrapper("low-cbf/cnic/1", json_commands=["CallMethod"], json_attributes=
["vd__configuration"])
```

You can also exclude Attributes & Commands that are incorrectly detected:

```
# Perhaps MyCommand's doc_in says "Use CSV, never JSON!"
# and notJsonAttr's description is "This is a colon-separated list, NOT JSON!"
# These would normally be automatically detected as using JSON by JsonWrapper!
JsonWrapper("my/device/name", not_json_commands=["MyCommand"], not_json_attributes=
["notJsonAttr"])
```

These parameters can be combined as you see fit. If you try to include & exclude the same thing, exclusion wins.

Reading a JSON Attribute

```
cnic = JsonWrapper("low-cbf/cnic/1", json_attributes=["vd__configuration"])
# We can loop over a list of dicts
for cfg in cnic.vd__configuration:   # look Ma, no json.loads !
    print("Beam:", cfg["beam"], "Station:", cfg["station"], cfg["substation"])
```

```
Beam: 1 Station: 345 1
Beam: 2 Station: 346 1
Beam: 3 Station: 347 1
Beam: 4 Station: 348 1
Beam: 5 Station: 349 1
```

Sending a JSON Command

If the Command is expecting a `dict`, you can use either keyword arguments, or pass a `dict` directly, your choice. Note that **`JsonWrapper`** has no knowledge of the underlying Tango Device's JSON schema, so you're on your own in terms of what data structure or keywords to supply.

```
cnic = JsonWrapper(cnic_proxy, json_commands=["CallMethod"])

# Keyword Arguments
cnic.CallMethod(method="stop_receive")   # Command explicitly configured as JSON above

# Any data type that can be processed by json.dumps can be used directly
request = {"version": "0.1.9", "source": "nexus"}
cnic.SelectPersonality(request)   # Command auto-detected, as its doc_in contains "JSON"
```

API

class **ska_low_cbf_integration.tango.JsonWrapper**(*target, json_commands=None, not_json_commands=None, json_attributes=None, not_json_attributes=None*)

Transparently translate JSON to/from a Tango DeviceProxy.

__getattr__(*attr_name*)

Get a Python attribute from our target (note: may be a Tango Command).

__init__(*target, json_commands=None, not_json_commands=None, json_attributes=None, not_json_attributes=None*)

Create a transparent JSON wrapper.

JSON conversion is automatically applied to Tango Attributes or Commands that contain the word "JSON" (case-insensitive) in their description (*doc_in* aka *in_type_desc* for Commands). Parameters are available to override detection behaviour.

Attributes detected/configured as JSON will have their return value decoded.

Commands detected/configured as JSON will have their input argument encoded.

Parameters:

- **target** (`Union`[`DeviceProxy`, `str`]) – Tango DeviceProxy or device name
- **json_commands** (`Union`[`Iterable`, `str`, `None`]) – Commands that take JSON input (overrides auto-detection)
- **not_json_commands** (`Union`[`Iterable`, `str`, `None`]) – Commands that do not take JSON input (overrides auto-detection)
- **json_attributes** (`Union`[`Iterable`, `str`, `None`]) – Attributes that return JSON strings (overrides auto-detection)
- **not_json_attributes** (`Union`[`Iterable`, `str`, `None`]) – Attributes that do not return JSON strings (overrides auto-detection)

__setattr__(*key, value*)

Delegate to target, once we are initialised.

***static* jsonify_input(*func*)**

Convert non-JSON input into JSON before passing to func.

Return type: `Callable`