

# PyOptiSLang cheat sheet



Version: 1.5.dev0

## Basic optiSLang Operations

### Create a new optiSLang instance

PyOptiSLang automatically detects the newest standard installation paths and by default spawns a new optiSLang instance locally. When used this way, optiSLang creates a new project in a temporary directory. A new instance can be created using either the context manager syntax (recommended):

```
# Create an Optislang instance using context
# manager
from ansys.optislang.core import Optislang
with Optislang() as osl:
    print(osl)
```

Or directly:

```
# Create an Optislang instance directly
osl = Optislang()
print(osl)
osl.dispose()
```

In this case, the instance must be disposed of when it is no longer needed.

### Connect to existing optiSLang

To connect to an already running optiSLang instance, use the `local_server_id` or `host` and `port` arguments.

```
# Connect to an existing optiSLang instance
from ansys.optislang.core import Optislang
osl = Optislang(host="127.0.0.1", port=5310)
print(osl)
osl.dispose()
```

If optiSLang was started with the `shutdown_on_finished` argument set to `False`, it won't shut down automatically. To shut down manually, the `shutdown()` command must be called prior to disposing the instance.

### Start optiSLang in GUI mode

By default, optiSLang is started in batch mode. To start it in GUI mode, set the `batch` argument to `False`.

```
# Start optiSLang in GUI mode
from ansys.optislang.core import Optislang
with Optislang(batch=False) as osl:
    print(osl)
```

### Find available optiSLang installations

To use a specific optiSLang installation, use the `executable` argument during initialization. Convenience functionality that provides an ordered dictionary sorted by version is also available:

```
# Find and use a specific optiSLang installation
from ansys.optislang.core import Optislang
from ansys.optislang.core.utils import
    find_all_osl_exec

osl_execs = find_all_osl_exec()
latest_exec = next(iter(osl_execs()))
with Optislang(executable=latest_exec) as osl:
    print(osl)
```

## Project Management

### Load a project

The path to the project can be specified either on initialization:

```
# Load project during initialization
from ansys.optislang.core import Optislang
from pathlib import Path

path = Path(r"C:\path\to\project.opf")
with Optislang(project_path=path) as osl:
    print(osl)
```

or after initialization:

```
# Load project after initialization
with Optislang() as osl:
    osl.application.open(path)
    print(osl)
```

### Start project execution

```
# Start project execution
from ansys.optislang.core import Optislang
from pathlib import Path

path = Path(r"C:\path\to\project.opf")
with Optislang(project_path=path) as osl:
    osl.application.project.start()
```

### Save projects

The project can be saved using the following commands: `save()`, `save_as()` and `save_copy()`. All these commands are provided by the Application class.

```
# Save project with a new name
from ansys.optislang.core import Optislang
from pathlib import Path

new_path = Path().cwd() / "project.opf"
with Optislang() as osl:
    osl.application.save_as(new_path)
```

## Obtain basic project information

Print the basic project information and nodes at the top level:

```
# Get basic project information
from ansys.optislang.core import Optislang
from pathlib import Path

path = Path(r"C:\path\to\project.opf")
with Optislang(project_path=path) as osl:
    project = osl.application.project
    print(project.get_name())
    print(project.get_status())
    rs = project.root_system
    for node in rs.get_nodes():
        print(
            f"Node: {node.get_name()}, " +
            f"Type: {node.type}"
        )
```

## Design Evaluation

### Evaluate design

This functionality is implemented for projects that have a parametric on top level.



Figure 1: Evaluate design

To evaluate a design, query the root system for the reference design, modify parameters as needed and evaluate them. Please note that only the last evaluated design is stored in the optiSLang database.

```
# Evaluate a design with modified parameters
from ansys.optislang.core import Optislang
from pathlib import Path

path = Path(r"C:\path\to\project.opf")
with Optislang(project_path=path) as osl:
    rs = osl.application.project.root_system
    design = rs.get_reference_design()
    design.set_parameter_value(
        name="Parameter1",
        value=10
    )
    out_design = rs.evaluate_design(design)
    print([resp.value for resp in
        out_design.responses])
```

Designs can also be created from scratch:

```
# Create a design from scratch
import ansys.optislang.core.project_parametric as pp

design = pp.Design(
    parameters={
        "Parameter1": 10,
        "Parameter2": 20
    }
)
```

## Workflow Creation

### Create a new node

```
# Create a new node
from ansys.optislang.core import Optislang
import ansys.optislang.core.node_types as nt

node_type = nt.Python2
with Optislang() as osl:
    rs = osl.application.project.root_system
    python_node = rs.create_node(node_type)
```

### Connect nodes

```
# Connect two nodes
from ansys.optislang.core import Optislang
import ansys.optislang.core.node_types as nt

with Optislang() as osl:
    rs = osl.application.project.root_system
    node1 = rs.find_nodes_by_name("Python_1")[0]
    node2 = rs.find_nodes_by_name("Python_2")[0]
    node1_oslot =
        node1.get_output_slots("ODesign")[0]
    node1_oslot.connect_to(
        node2.get_input_slots("IDesign")[0]
    )
```

### Modify node properties

```
# Modify node properties
from ansys.optislang.core import Optislang
from ansys.optislang.core.examples import get_files

path = get_files("omdb_files")[1][0]
with Optislang(project_path=path) as osl:
    rs = osl.application.project.root_system
    node = rs.find_nodes_by_name("Sensitivity")[0]
    prop = node.get_property("AlgorithmSettings")
    prop["num_discretization"] = 100
    node.set_property("AlgorithmSettings", prop)
```

## Manage parameters, responses, criteria and designs

Parameters, responses and criteria can be modified using the corresponding manager instance. These managers are provided by each parametric system (such as Sensitivity node): ParameterManager, CriteriaManager, and ResponseManager. To work with evaluated designs, use the DesignManager.

```
# Work with parameters, responses, and designs
from ansys.optislang.core import Optislang
from ansys.optislang.core.examples import get_files

path = get_files("omdb_files")[1][0]
with Optislang(project_path=path) as osl:
    rs = osl.application.project.root_system
    node = rs.find_nodes_by_name("Sensitivity")[0]

    # Get parameter names
    pm = node.parameter_manager
    print(pm.get_parameters_names())

    # Access design and responses
    dm = node.design_manager
    design = dm.get_design("0.1")
    print([r.value for r in design.responses])
```

## Full workflow creation example

This snippet creates a simple workflow consisting of a Sensitivity node and a Calculator node that calculates the distance from the origin in 3D space. First, node types and parameters are defined:

```
# Define node types and parameters for workflow
import ansys.optislang.core.node_types as nt
import ansys.optislang.core.project_parametric as pp

sensi_type = nt.Sensitivity
calc_type = nt.CalculatorSet
parameters = [
    pp.OptimizationParameter(name="X1"),
    pp.OptimizationParameter(name="X2"),
    pp.OptimizationParameter(name="X3")
]
```

Then, create the Sensitivity node and add parameters:

```
# Create Sensitivity node and add parameters
from ansys.optislang.core import Optislang
from ansys.optislang.core.nodes import DesignFlow

with Optislang() as osl:
    rs = osl.application.project.root_system
    sensi = rs.create_node(sensi_type)
    for par in parameters:
        sensi.parameter_manager.add_parameter(par)
```

Create and connect the Calculator node:

```
# Create Calculator node with automatic connection
calc = sensi.create_node(
    type_=calc_type,
    design_flow=DesignFlow.RECEIVE_SEND
)
```

Alternatively, connections can be specified explicitly:

```
# Manual connection of nodes
iod_slot =
    sensi.get_inner_output_slots("IDesign")[0]
iod_slot.connect_to(
    calc.get_input_slots("IDesign")[0]
)
iid_slot =
    sensi.get_inner_input_slots("IDesign")[0]
iid_slot.connect_from(
    calc.get_output_slots("ODesign")[0]
)
```

Create variables in the calculator node and register them as responses:

```
# Register calculator variable as response
location = {
    "id": "distance",
    "expression": "(X1**2 + X2**2 +
    X3**2)**0.5"
}
calc.register_location_as_response(
    location=location,
    name="distance",
    reference_value=0.0
)
```

## Advanced Features

### Working with placeholders

Placeholders are variables that can be assigned to workflow component properties, allowing you to easily update properties in multiple components at once.

### Create and assign placeholders

```
# Create a standalone placeholder
from ansys.optislang.core import Optislang
from ansys.optislang.core.placeholder_types import PlaceholderType, UserLevel

with Optislang() as osl:
    # Create a placeholder
    thickness_id = osl.project.create_placeholder(
        value=5.0,
        placeholder_id="thickness",
        type_=PlaceholderType.REAL,
        user_level=UserLevel.COMPUTATION_ENGINEER,
        description="Plate thickness in mm",
    )

    # Assign to a node property
    node = osl.project.root_system.get_nodes()[0]
```

```
node.assign_placeholder(  
    property_name="PropertyName",  
    placeholder_id="thickness"  
)
```

## Create placeholders from node properties

```
# Create placeholder directly from node property  
with Optislang() as osl:  
    node = osl.project.root_system.get_nodes()[0]  
    placeholder_id =  
        node.create_placeholder_from_property(
```

```
        property_name="MaxParallel",  
        placeholder_id="max_parallel")
```

## Query and modify placeholders

```
# Get all placeholders and modify values  
with Optislang() as osl:  
    # List all placeholders  
    placeholder_ids =  
        osl.project.get_placeholder_ids()
```

```
print(f"Found {len(placeholder_ids)}  
placeholders")  
  
# Get placeholder information  
for pid in placeholder_ids:  
    info = osl.project.get_placeholder(pid)  
    print(f"ID: {info.placeholder_id}, Value:  
{info.value}")  
  
# Set new value  
osl.project.set_placeholder_value("thickness",  
    7.5)
```