Process Integration and Workflow Automation

WOST Workshop 2023

Overview

1. Python inside the optiSLang workflow (Python node)
   - Data Manipulation
   - (Customization)

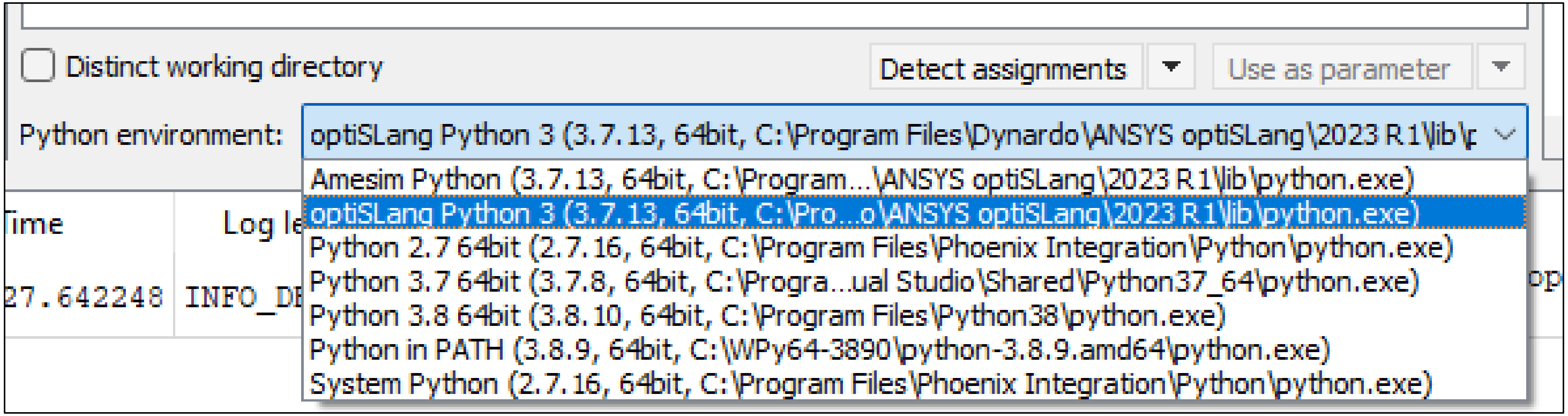
2. Python outside the optiSLang workflow (Python console)
   - Workflow manipulation
   - Batch execution

3. Python outside optiSLang (PyOptiSLang)
   - Workflow control

Python inside the optiSLang workflow

Data Manipulation – Python node

- Select Python version:



Python environment: optiSLang Python 3 (3.7.13, 64bit, C:\Program Files\Dynardo\ANSYS optiSLang\2023 R1\lib\p

Amesim Python (3.7.13, 64bit, C:\Program...\ANSYS optiSLang\2023 R1\lib\python.exe)
optiSLang Python 3 (3.7.13, 64bit, C:\Pro...o\ANSYS optiSLang\2023 R1\lib\python.exe)
Python 2.7 64bit (2.7.16, 64bit, C:\Program Files\Phoenix Integration\Python\python.exe)
Python 3.7 64bit (3.7.8, 64bit, C:\Progra...ual Studio\Shared\Python37_64\python.exe)
Python 3.8 64bit (3.8.10, 64bit, C:\Program Files\Python38\python.exe)
Python in PATH (3.8.9, 64bit, C:\WPy64-3890\python-3.8.9.amd64\python.exe)
System Python (2.7.16, 64bit, C:\Program Files\Phoenix Integration\Python\python.exe)

Data Manipulation – Python node

*may require admin privileges

- Add packages to optiSLang Python*
  - Windows:

```
cd „C:\Program Files\ANSYS Inc\v231\optiSLang“
optislang-python-cmd –m pip install –U matplotlib
```

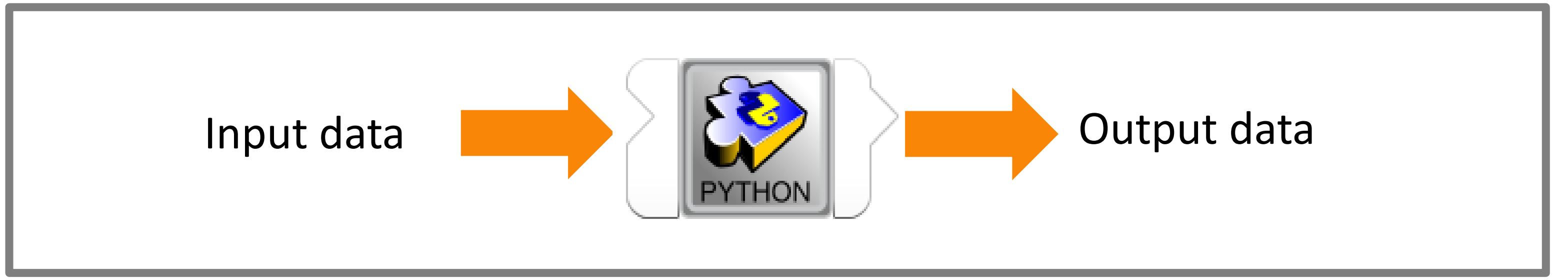
  - Linux:

```
cd [installation path]
optislang-python –m pip install –U matplotlib
```

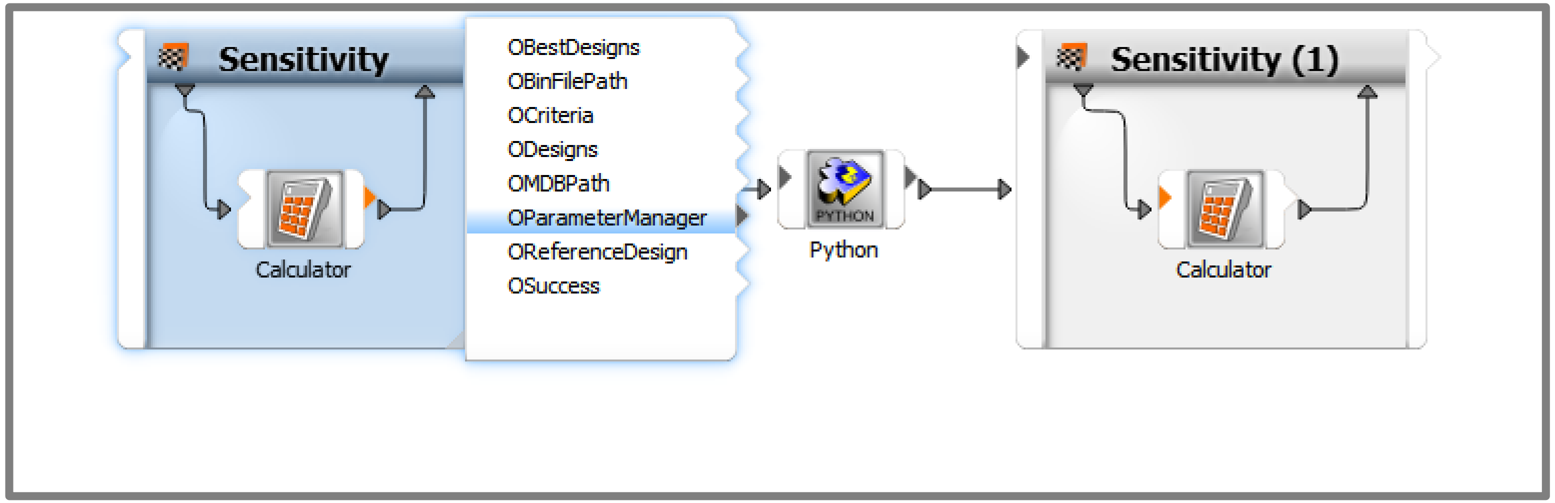
- For more information see optiSLang help:
  - https://ansyshelp.ansys.com/account/secured?returnurl=/Views/Secured/corp/v231/en/opti_inst_lic/opti_inst_lic_python_packages.html?q=pip

Data Manipulation – Python node

- For data manipulation, solving, pre- and postprocessing
- Read, create and manipulate data during run

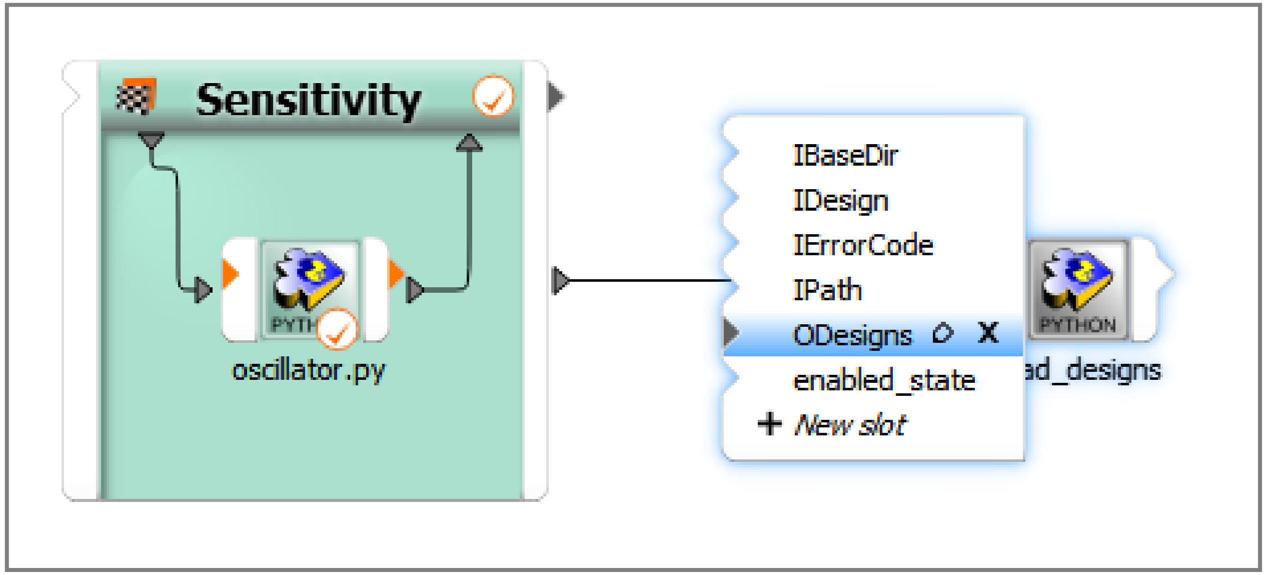


- Transfer any existing data from the workflow to Python and vice versa



Data Manipulation – Python node

- Example: Convert “ODesigns” to Python

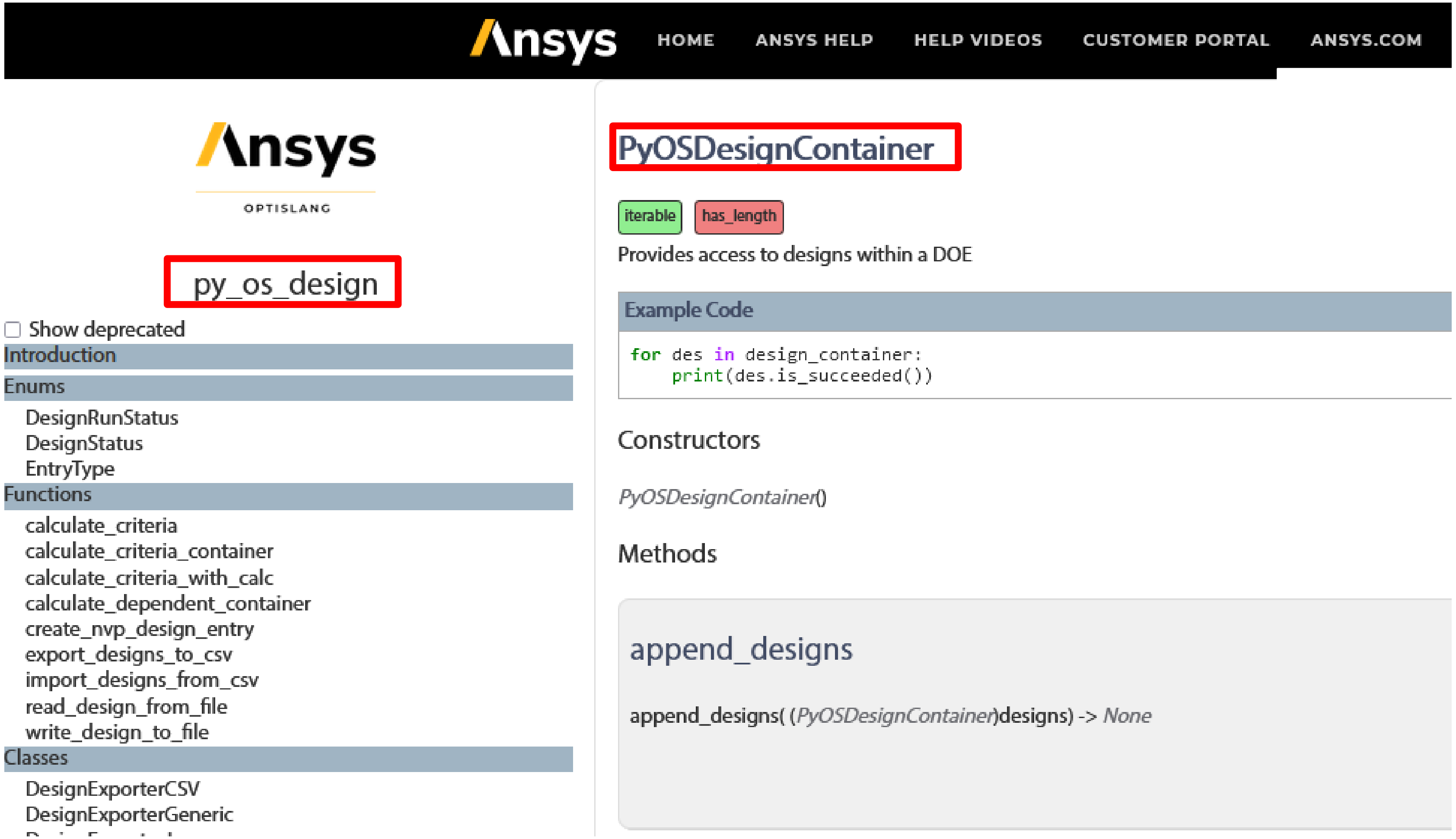


- Get type of “ODesigns”

```
print(type(ODesigns))
<class ‘py_os_design.PyOSDesignContainer‘>
```

Python API documentation

https://ansysapi.ansys.com/account/secured?returnurl=/Views/Secured/published_content/opti_python_api_v231/index.html



py_os_design

□ Show deprecated

Introduction

Enums

- DesignRunStatus
- DesignStatus
- EntryType

Functions

- calculate_criteria
- calculate_criteria_container
- calculate_criteria_with_calc
- calculate_dependent_container
- create_nvp_design_entry
- export_designs_to_csv
- import_designs_from_csv
- read_design_from_file
- write_design_to_file

Classes

- DesignExporterCSV
- DesignExporterGeneric

PyOSDesignContainer

iterable has_length

Provides access to designs within a DOE

Example Code

```python
for des in design_container:
    print(des.is_succeeded())
```

Constructors

PyOSDesignContainer()

Methods

append_designs

append_designs((*PyOSDesignContainer*)designs) -> *None*

iterable

```python
for thing in my_obj:
    pass
```

haslength

```python
len(my_obj)
```

itemgetter

```python
value = my_obj[key]
```

itemsetter

```python
my_obj[key] = value
```

Data Manipulation – Python node

```python
import py_os_design

# print(type(ODesigns))  # <class 'py_os_design.PyOSDesignContainer'>

try:
    dc = ODesigns
except NameError:
    dc = py_os_design.PyOSDesignContainer()

# dc is iterable
for d in dc:
    # print(type(d))  # <class 'py_os_design.PyOSDesign'>
    # print(type(d.get_parameters()))  # <class 'py_os_design.PyOSDesignPoint'>
    # print(type(d.get_responses()))  # <class 'py_os_design.PyOSDesignPoint'>
    # print(type(value))  # <class 'py_os_design.PyOSDesignEntry'>

    id_ = d.get_id().ToString()
    print(f'design id={id_}'.format())
    for name, value in d.get_parameters():
        print(f' parameter {name}={value.get()}')
    for name, value in d.get_responses():
        print(f' response {name}={value.get()}')
```

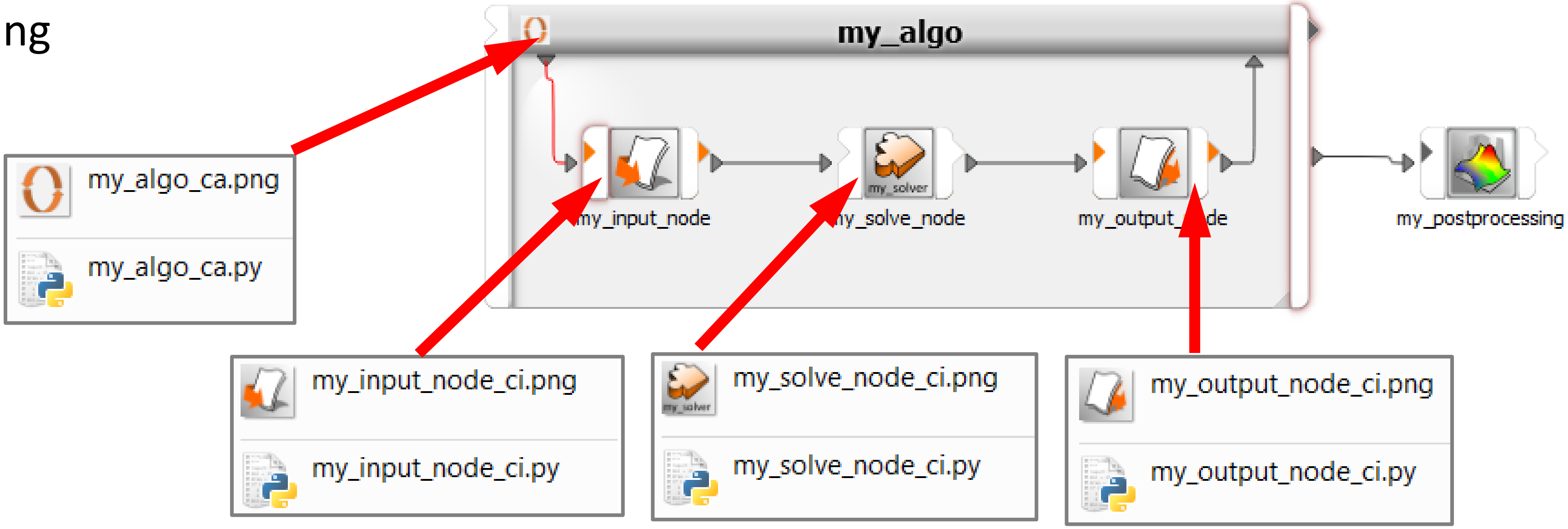
Data Manipulation – Python node

- **Example**: Filter DesignContainer (without violated designs)

```
import py_os_design

try:
   dc = ODesigns
except NameError:
   dc = py_os_design.PyOSDesignContainer()

dc_without_violated = py_os_design.PyOSDesignContainer()
for d in dc:
   pass
   # ??
```

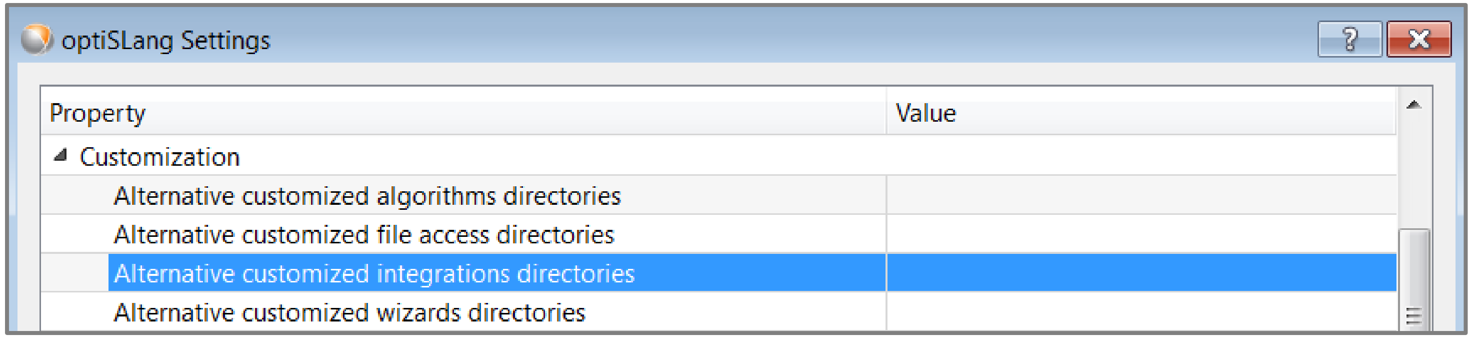
Customization

- optiSLang provides plugin mechanisms via Python/C++ for
  - Integration nodes (input, solve, output)
  - ETK
  - Algorithms/Systems
  - Solver Wizard
  - Surrogates
  - Postprocessing



Customization

- Naming conventions
  - Integration: *_ci.py
  - ETK: *_etk.py
  - Algorithm: *_ca.py
  - Surrogate: *_surr.py
  - Solver Wizard: *_cw.py
- Default search directories:
  - [optislang install path]\scripting
- Add further search directories via optiSLang settings



optiSLang Settings

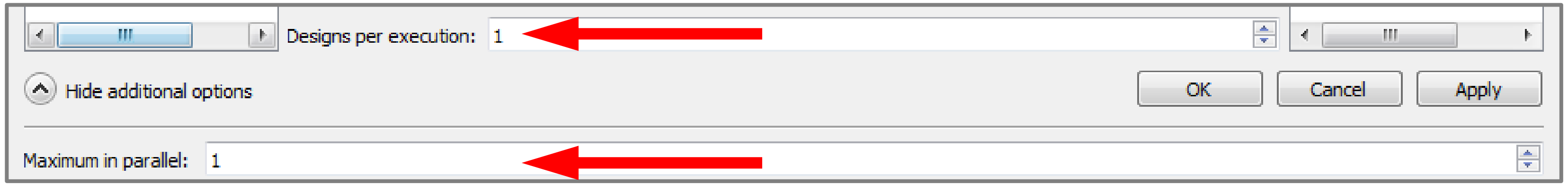
Property	Value
Customization	
Alternative customized algorithms directories	
Alternative customized file access directories	
Alternative customized integrations directories	
Alternative customized wizards directories	

Customization

- Add further search directories via environment variable
  - Integration: OSL_ALT_CI_SEARCH_DIRS
  - ETK: OSL_ALT_CI_SEARCH_DIRS
  - Algorithm: OSL_ALT_CA_SEARCH_DIRS
  - Surrogate: OSL_ALT_CA_SEARCH_DIRS
  - Solver Wizard: OSL_ALT_CW_SEARCH_DIRS
- Custom scripts are loaded on application startup
- Optional additional files:
  - script_name.icon – icon used in optiSLang
  - script_name.html – help
  - script_name_ci_ui.py – ettings GUI
  - script_name.cfg – config file

Integration config file

	Version1	Version2	Config file entry
set GUI name	x	x	Name *gui_name*
enable parallel computing	x	x	EnableParallel true
Enable multiple designs	-	x	ScriptInterfaceVersion 2 EnableMultiDesignMode true
Select Python	x	x	Python environment: "Python 3.7 64bit"



Designs per execution: 1

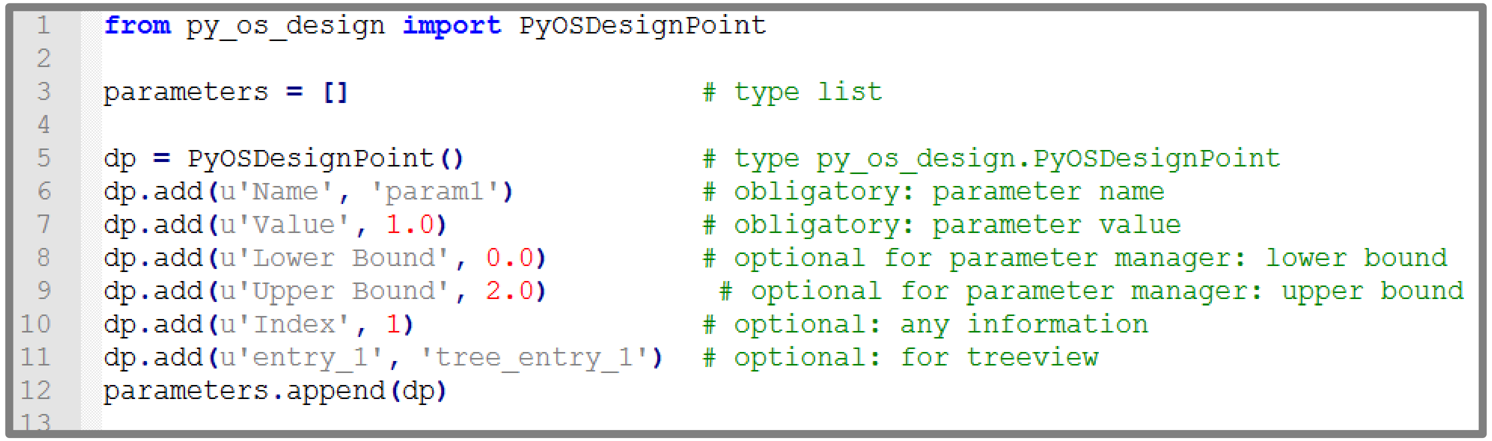
Maximum in parallel: 1

Integration (Version 1)

- Input node, e.g. my_input_node_ci.py expects:
  - def ExtractInputContainer(args)
  - def SetParameters(args)
- Solve node, e.g. my_solve_node_ci.py expects:
  - def RunSolver(args)
- Output node, e.g. my_output_node_ci.py expects:
  - def ExtractOutputContainer(args)
- Full integration node, e.g. my_integration_node_ci.py expects:
  - def ExtractInputContainer(args)
  - def SetParameters(args)
  - def RunSolver(args)
  - def ExtractOutputContainer(args)
- „args“ and expected return value depends on the function and is documented here:
  https://ansyshelp.ansys.com/account/secured?returnurl=/Views/Secured/corp/v231/en/opti_api/opti_api_python_based_custom_int_api.html

Integration - ExtractInputContainer

- Create list of py_os_design.PyOSDesignPoint:
  - Obligatory:
    - Name and value
  - Optional with optiSLang usage:
    - Lower and upper bound
    - Mean, Stddev
    - Treeview (e.g. for xml)
  - Optional without optiSLang usage
    - Any other information

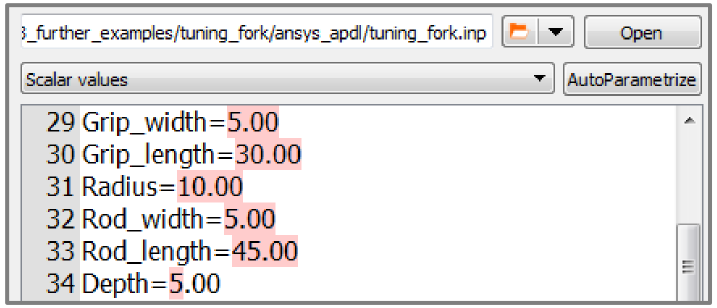


```python
from py_os_design import PyOSDesignPoint

parameters = []                    # type list

dp = PyOSDesignPoint()             # type py_os_design.PyOSDesignPoint
dp.add(u'Name', 'param1')          # obligatory: parameter name
dp.add(u'Value', 1.0)              # obligatory: parameter value
dp.add(u'Lower Bound', 0.0)        # optional for parameter manager: lower bound
dp.add(u'Upper Bound', 2.0)         # optional for parameter manager: upper bound
dp.add(u'Index', 1)                # optional: any information
dp.add(u'entry_1', 'tree_entry_1')  # optional: for treeview
parameters.append(dp)
```

Integration – Input node example

- Steps to parametrize input file with standard text input node
  - AutoParametrize locations for scalar values
  - Separately for every location: set name and add as parameter

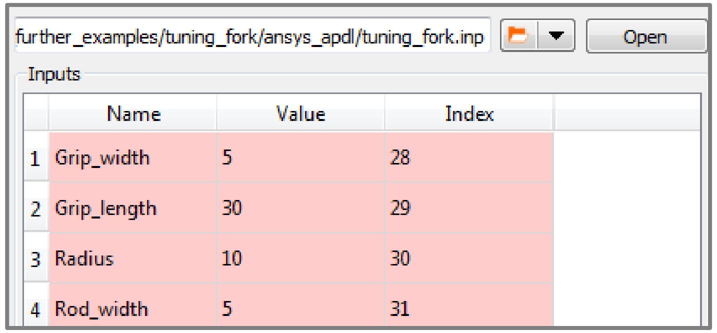
text_input



```
29 Grip_width=5.00
30 Grip_length=30.00
31 Radius=10.00
32 Rod_width=5.00
33 Rod_length=45.00
34 Depth=5.00
```

- Steps to parametrize with custom input node
  - Add wanted parameters using drag and drop

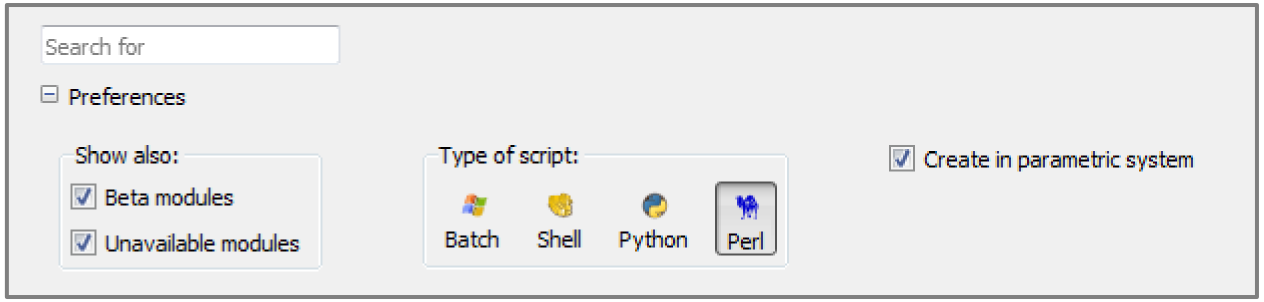
custom_input



	Name	Value	Index
1	Grip_width	5	28
2	Grip_length	30	29
3	Radius	10	30
4	Rod_width	5	31

Solver Wizard

- Solver Wizard settings



Search for

⊟ Preferences

Show also:

☑ Beta modules

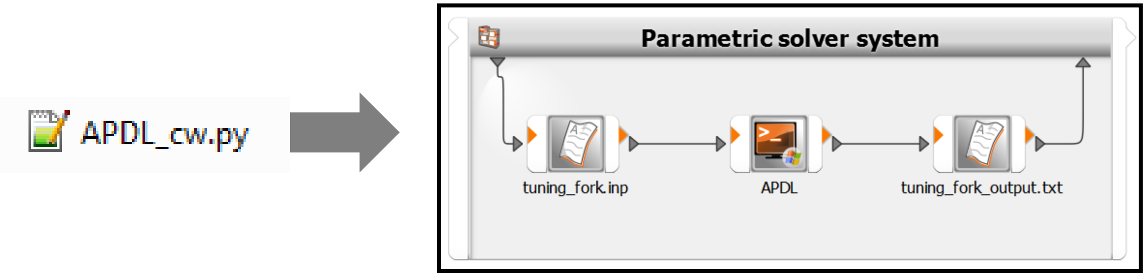
☑ Unavailable modules

Type of script:

☑ Create in parametric system

- Arguments passed into the *_cw.py script:
  - sys.argv[1] : str
    - Selected file
  - wizard_system_name : str
    - Selected/Created parametric system name
  - type_of_script : str
    - Selected script type
  - create_in_parametric_system : bool
    - Existence of a parametric system

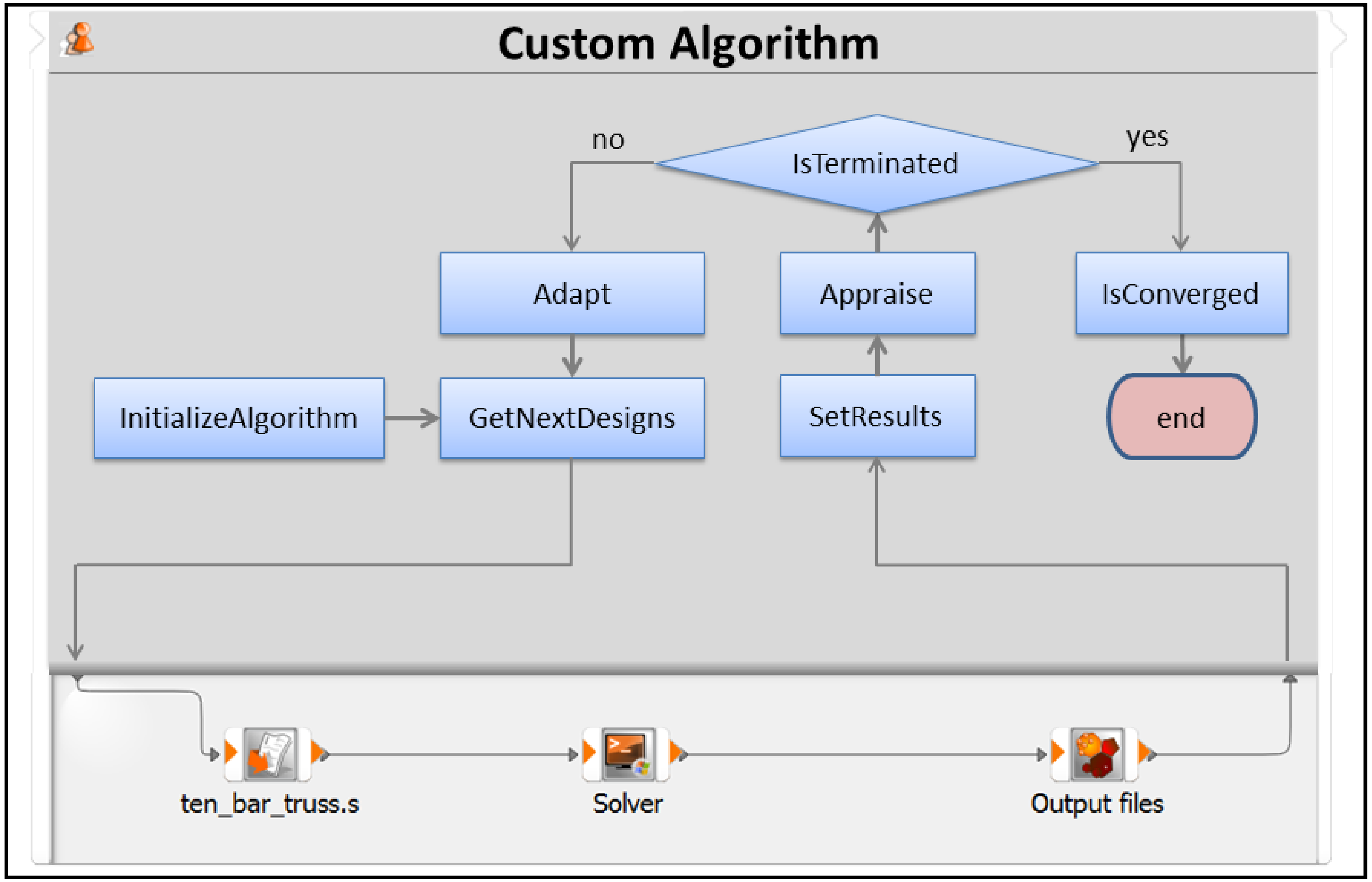
Solver Wizard



```
selected_file = sys.argv[1]

parametric_system = find_actor(wizard_system_name)

input_node = actors.CustomIntegrationActor('APDL_input')
input_node.name = 'tuning_fork.inp'

solve_node = actors.BatchScriptActor('APDL')

output_node = actors.CustomIntegrationActor('APDL_output')
output_node.name = 'tuning_fork_output.txt'

connect(parametric_system, 'IODesign', input_node, 'IDesign')
connect(input_node, 'ODesign', solve_node, 'IDesign')
connect(solve_node, 'ODesign', output_node, 'IDesign')
connect(output_node, 'ODesign', parametric_system, 'IIDesign')
```

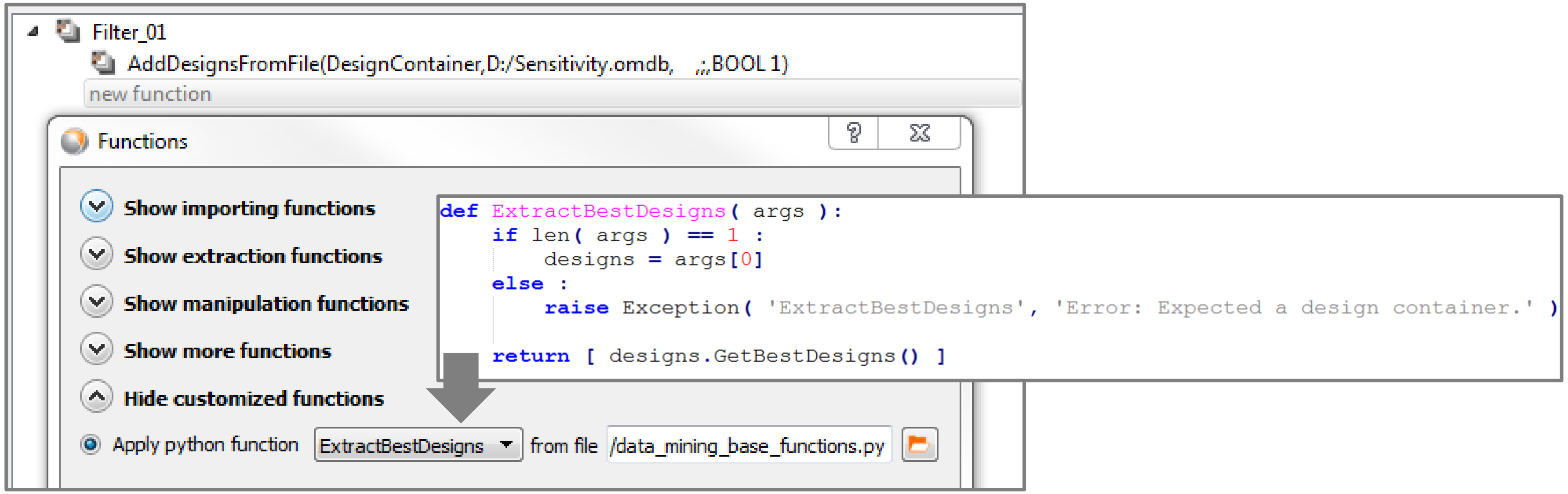
Algorithms

- Use own algorithms to sample designs or to find optimal/robust solutions



Customized Data Mining

- Use own python functions to extract or modify data
- Function argument is the resulting object of the previous function call with type list (currently with only one entry)
- Outputs must be a list of output arguments of type of PyOSDesignEntry or PyOSDesignContainer (currently the first entry will be used only)


Filter_01
AddDesignsFromFile(DesignContainer,D:/Sensitivity.omdb, ,;,BOOL 1)
new function
Functions
Show importing functions
Show extraction functions
Show manipulation functions
Show more functions
Hide customized functions

```
def ExtractBestDesigns( args ):
    if len( args ) == 1 :
        designs = args[0]
    else :
        raise Exception( 'ExtractBestDesigns', 'Error: Expected a design container.' )

    return [ designs.GetBestDesigns() ]
```

● Apply python function ExtractBestDesigns from file /data_mining_base_functions.py

Customized Postprocessing

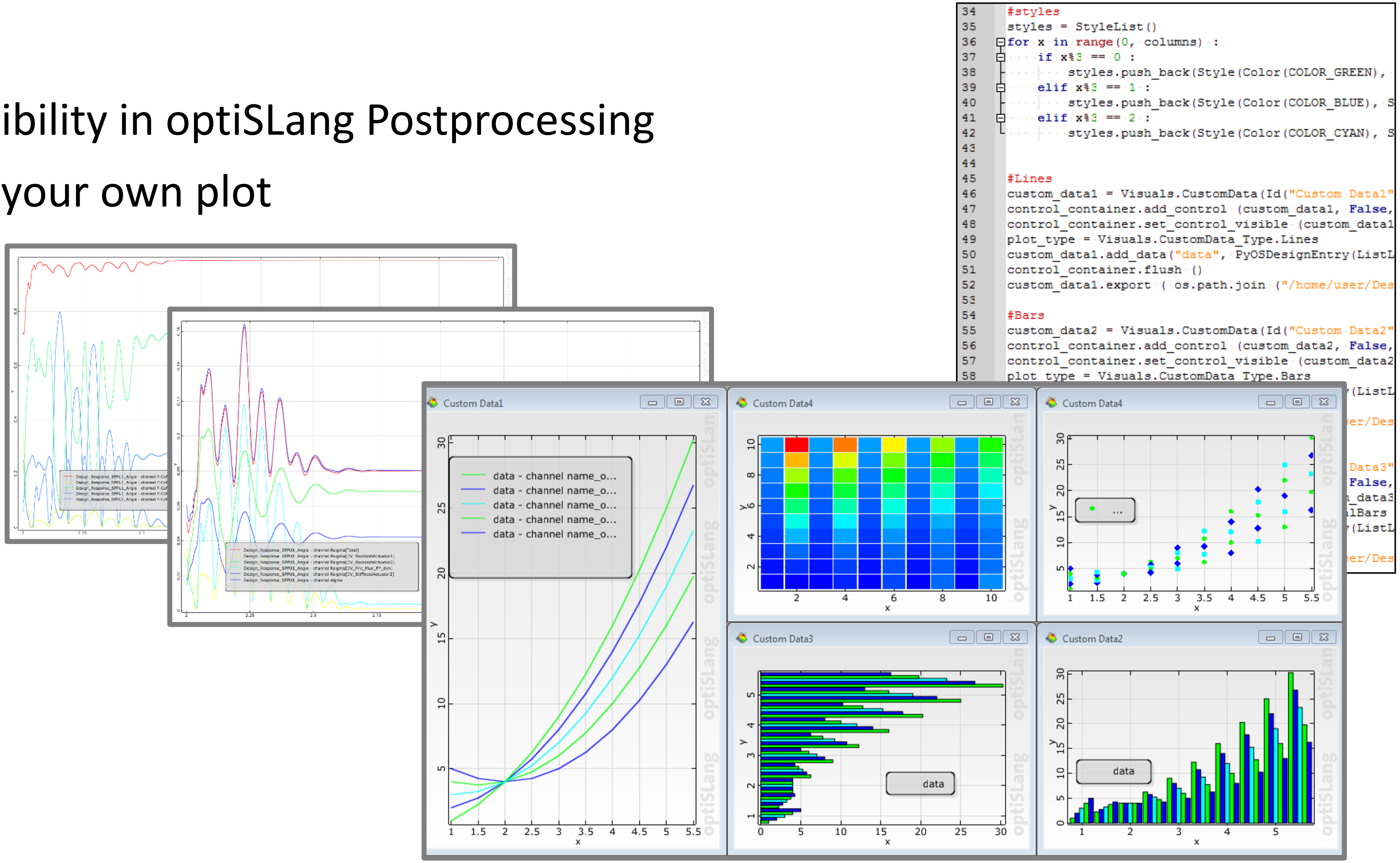
- optiSLang Postprocessing incorporates several default modes

- Find following modes in [install_path]/scripting/postprocessing
  - Approximation monitoring
  - Optimization monitoring
  - Reliability monitoring
  - Statistics monitoring

- Customize existing modes or create completely user-defined scripts using Python interface

Custom Plots

- Full flexibility in optiSLang Postprocessing
- Define your own plot



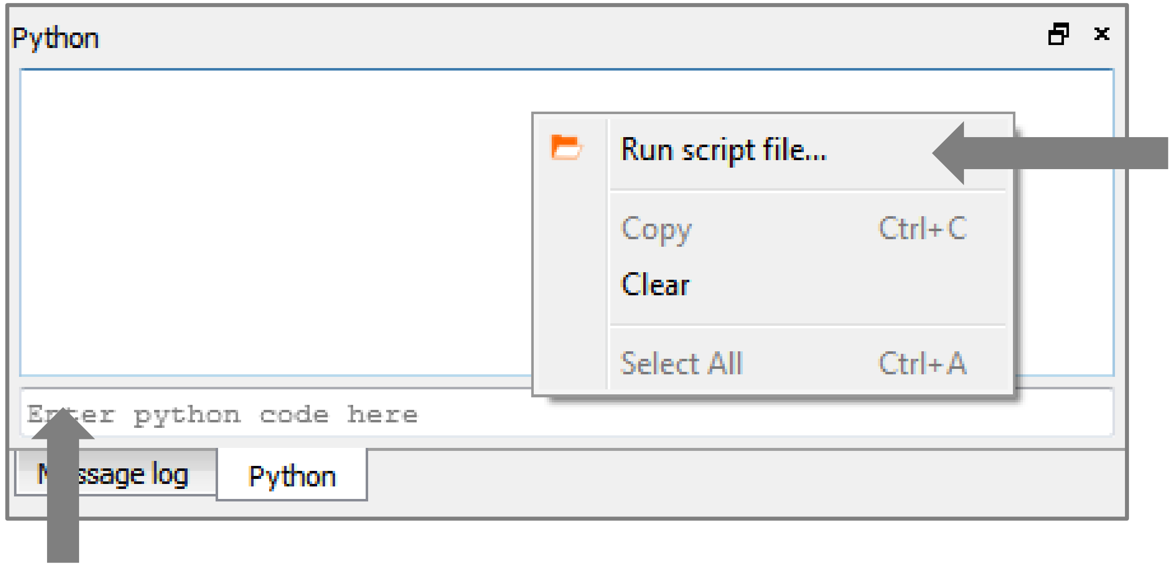
```
#styles
styles = StyleList()
for x in range(0, columns) :
    if x%3 == 0 :
        styles.push_back(Style(Color(COLOR_GREEN),
    elif x%3 == 1 :
        styles.push_back(Style(Color(COLOR_BLUE), S
    elif x%3 == 2 :
        styles.push_back(Style(Color(COLOR_CYAN), S


#Lines
custom_data1 = Visuals.CustomData(Id("Custom Data1"
control_container.add_control (custom_data1, False,
control_container.set_control_visible (custom_data1
plot_type = Visuals.CustomData_Type.Lines
custom_data1.add_data("data", PyOSDesignEntry(ListL
control_container.flush ()
custom_data1.export ( os.path.join ("/home/user/Des

#Bars
custom_data2 = Visuals.CustomData(Id("Custom Data2"
control_container.add_control (custom_data2, False,
control_container.set_control_visible (custom_data2
plot_type = Visuals.CustomData_Type.Bars
```

Python outside the optiSLang workflow

Ansys

Workflow Manipulation – Python console

- Build or modify a project
- Access to all data which influences the workflow behavior before or after run (e.g. systems, nodes, connections, settings)

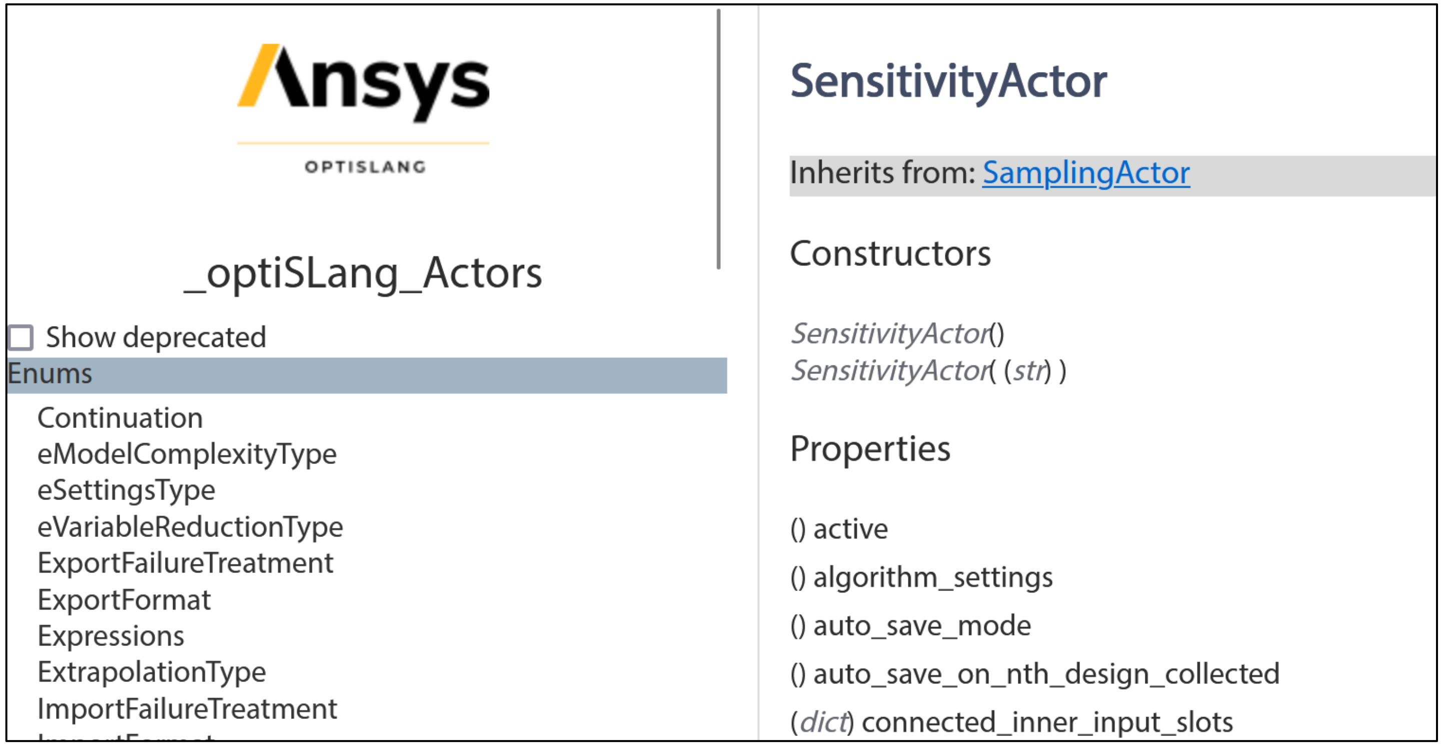


Multiple line commands (including indendation)

Multilple line commands (without indendation)

Workflow Manipulation – Python console

- Create new actor (see Python API)



Ansys OPTISLANG

_optiSLang_Actors

☐ Show deprecated

Enums

Continuation
eModelComplexityType
eSettingsType
eVariableReductionType
ExportFailureTreatment
ExportFormat
Expressions
ExtrapolationType
ImportFailureTreatment

SensitivityActor

Inherits from: SamplingActor

Constructors

SensitivityActor()
SensitivityActor((*str*))

Properties

() active

() algorithm_settings

() auto_save_mode

() auto_save_on_nth_design_collected

(*dict*) connected_inner_input_slots

```
my_sensitivity = actors.SensitivityActor('Sensitivity')
my_python = actors.Python2Actor('Python')
```

Workflow Manipulation – Python console

- Add sensitivity to scenery

```
add_actor(my_sensitivity)
```

- Add python node to sensitivity

```
my_sensitivity.add_actor(my_python)
```

- connect slots

```
connect(my_sensitivity, 'IODesign', my_python, 'IDesign')
connect(my_python, 'ODesign', my_sensitivity, 'IIDesign')
```

- setup sampling method to e.g. FULLFACTORIAL

```
# ??
```

Workflow Manipulation – Python console

- optiSLang Examples “damped oscillator”
  - Parametric System using Python integration (oscillator_system_python.py)
  - ARSM with following Robustness Analysis using Python integration (oscillator_robustness_arsm.py)
  - Sensitivity Analysis using Python integration (oscillator_sensitivity_mop.py)
  - Append Optimization on MOP (oscillator_optimization_on_mop.py)
  - > Requires: oscillator_sensitivity_mop.py

- optiSLang Examples “oscillator calibration”
  - Parametric System using Python integration (oscillatorcalibration_system_python.py)
  - Parametric System using Text-based integration (oscillatorcalibration_system_ascii.py)

optiSLang Project – Command Line Interface

- optiSLang can be started via command line as graphical user interface (GUI) process

  [installation_path]/optislang [options]

- or as a batch job

  [installation_path]/optislang --batch (-b) [options]

Informational options	Description
--help (-h)	Overview of all command options
--version (-v)	Show optiSLang version information
--support-info	Display support information

Mode options	Description
--new [path]/project.opf	Create a new project in path with name „project.opf“

optiSLang Project – Command Line Interface

Operation options	Description
--no-run	No run of specified project (default: --run)
--force	Force processing of project
--restore	Restore project from last save point
--reset	Reset project before running
--no-save	Don‘t save the specified project at the end
--autorelocate	Automatically relocate external filepaths

Script options	Description
--python [path]/script.py	Run a Python script before processing project
--script-arg <arg>	Provide an argument for the script
--script-args <args>	Provide multiple arguments (space separated)

optiSLang Project – Command Line Interface

- **Example:** Create optiSLang project using batch mode (no run!)

```
set optislang_home=C:\Program Files\ANSYS Inc\v231\optiSLang

„%optislang_home%\optislang“ -b --new my_project.opf --python
                    „%examples_home%\oscillator_sensitivity_mop.py“ --no-run
```

- **Example:** Run optiSLang project using batch mode

```
„%optislang_home%\optislang“ -b my_project.opf
```

optiSLang Project – Command Line Interface

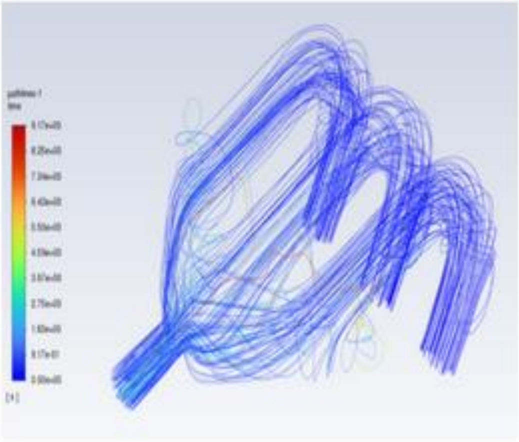
- Example: Update project with Optimization on MOP (oscillator_optimization_on_mop.py)

```
rem ??
```

Python outside optiSLang

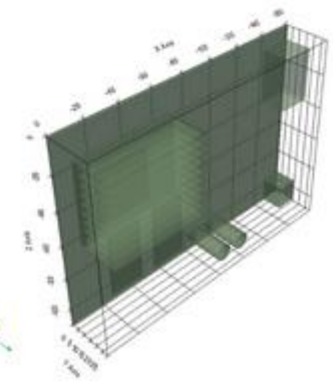
PyAnsys

PyAnsys a collection of many Python packages for using Ansys products through Python.



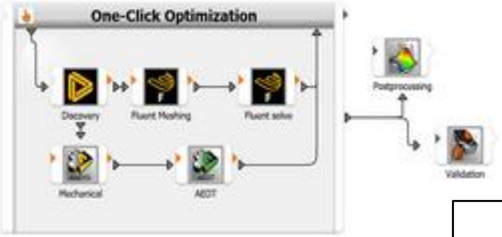
PyFluent

Pythonic interface to Ansys Fluent



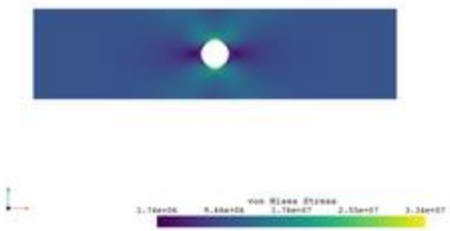
PyAEDT

Pythonic interface to AEDT (Ansys Electronic Desktop)



PyOptislang

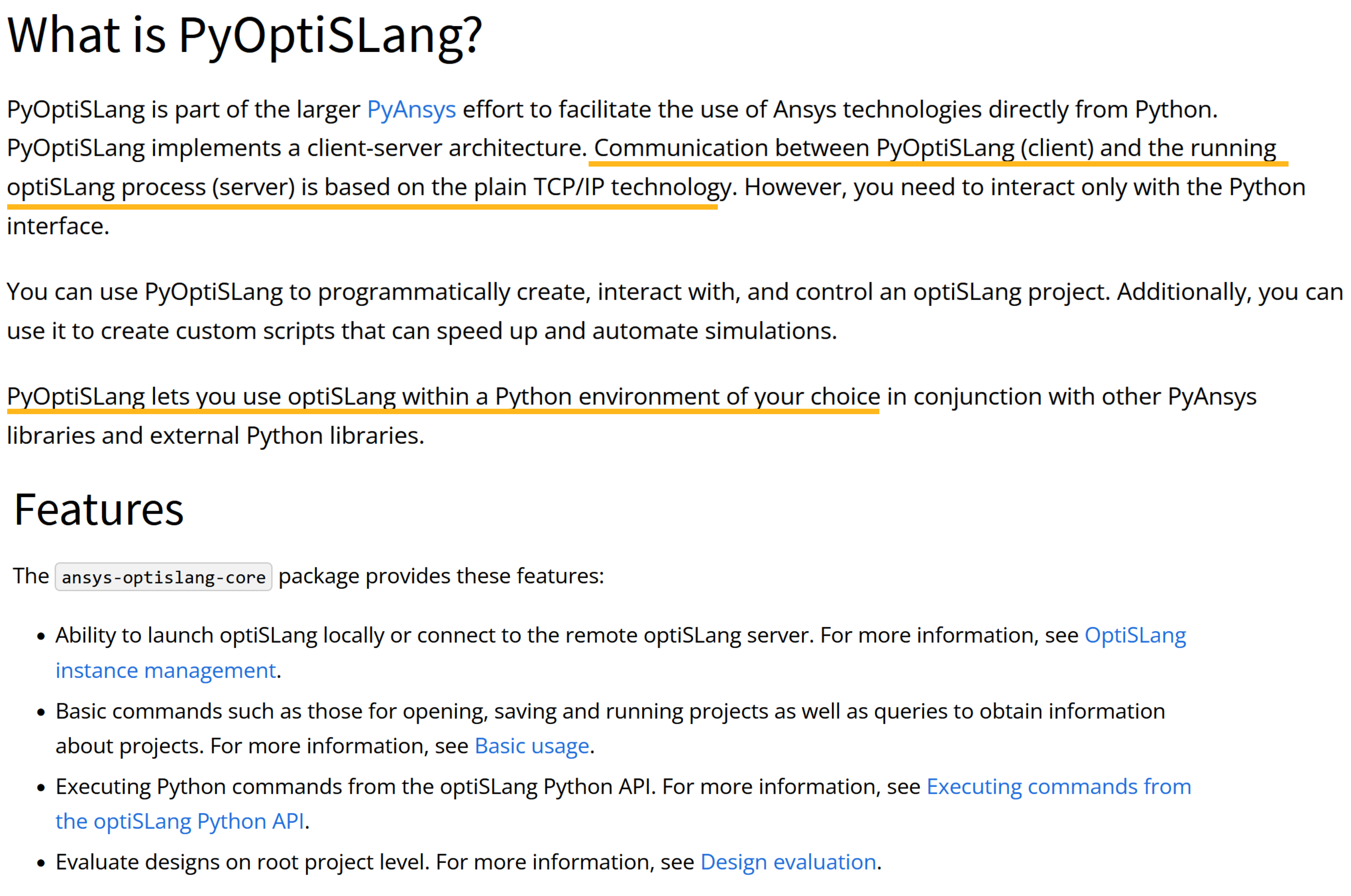
Pythonic interface to Ansys Optislang



PyMAPDL

Pythonic interface to Ansys MAPDL (Mechanical APDL)

....



What is PyOptiSLang?

PyOptiSLang is part of the larger PyAnsys effort to facilitate the use of Ansys technologies directly from Python. PyOptiSLang implements a client-server architecture. Communication between PyOptiSLang (client) and the running optiSLang process (server) is based on the plain TCP/IP technology. However, you need to interact only with the Python interface.

You can use PyOptiSLang to programmatically create, interact with, and control an optiSLang project. Additionally, you can use it to create custom scripts that can speed up and automate simulations.

PyOptiSLang lets you use optiSLang within a Python environment of your choice in conjunction with other PyAnsys libraries and external Python libraries.

Features

The `ansys-optislang-core` package provides these features:

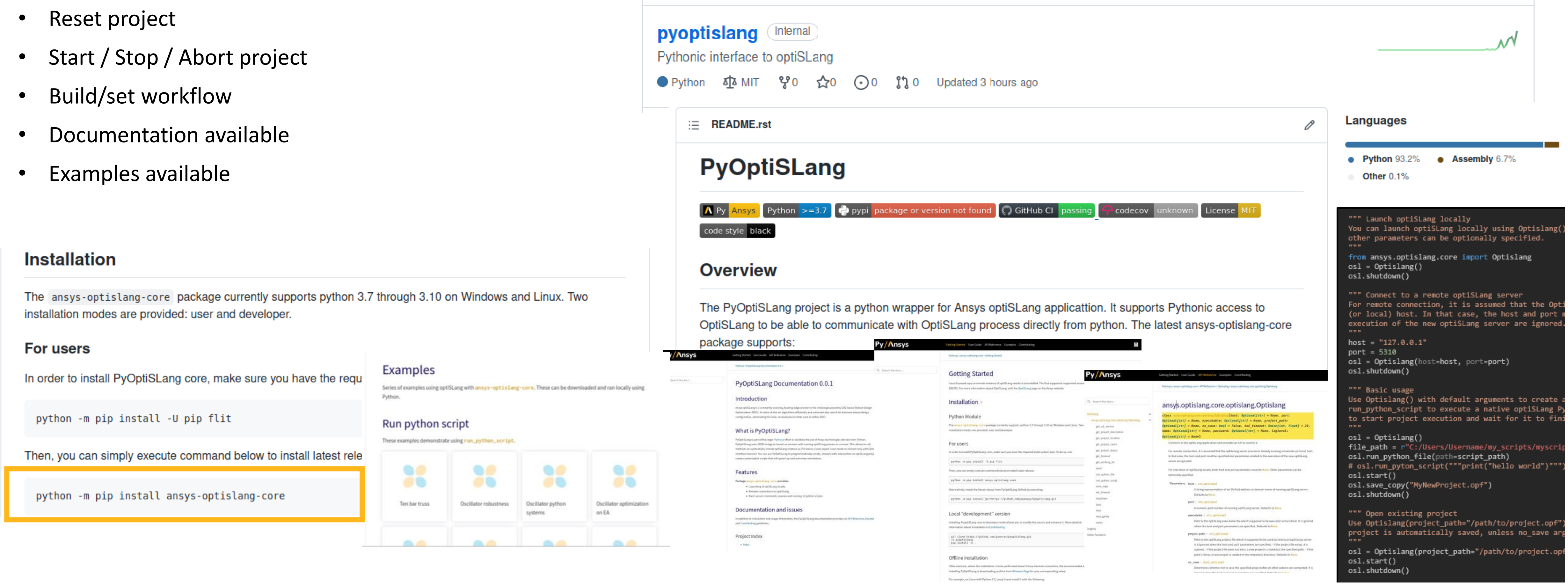
- Ability to launch optiSLang locally or connect to the remote optiSLang server. For more information, see OptiSLang instance management.
- Basic commands such as those for opening, saving and running projects as well as queries to obtain information about projects. For more information, see Basic usage.
- Executing Python commands from the optiSLang Python API. For more information, see Executing commands from the optiSLang Python API.
- Evaluate designs on root project level. For more information, see Design evaluation.

PyOptislang

Now Available for public

Py/Ansys

- Start new local optiSLang instance or connect to a running (local or remote) optiSLang instance (batch- mode)
- Open / Create New / Save project
- Reset project
- Start / Stop / Abort project
- Build/set workflow
- Documentation available
- Examples available



pyoptislang Internal

Pythonic interface to optiSLang

Python · MIT · 0 · 0 · 0 · 0 · Updated 3 hours ago

README.rst

PyOptiSLang

Py Ansys | Python >=3.7 | pypi package or version not found | GitHub CI passing | codecov unknown | License MIT | code style black

Overview

The PyOptiSLang project is a python wrapper for Ansys optiSLang applicattion. It supports Pythonic access to OptiSLang to be able to communicate with OptiSLang process directly from python. The latest ansys-optislang-core package supports:

Languages

Python	93.2%
Assembly	6.7%
Other	0.1%

Installation

The `ansys-optislang-core` package currently supports python 3.7 through 3.10 on Windows and Linux. Two installation modes are provided: user and developer.

For users

In order to install PyOptiSLang core, make sure you have the requ

```
python -m pip install -U pip flit
```

Then, you can simply execute command below to install latest rele

```
python -m pip install ansys-optislang-core
```

Examples

Series of examples using optiSLang with `ansys-optislang-core`. These can be downloaded and ran locally using Python.

Run python script

These examples demonstrate using `run_python_script`.

PyOptiSLang Documentation 0.0.1

Introduction

What is PyOptiSLang?

Features

Documentation and issues

Project Index

Getting Started

Installation

Python Module

For users

Local "development" version

Offline installation

ansys.optislang.core.optislang.Optislang

```
""" Launch optiSLang locally
You can launch optiSLang locally using Optislang(
other parameters can be optionally specified.
"""
from ansys.optislang.core import Optislang
osl = Optislang()
osl.shutdown()

""" Connect to a remote optiSLang server
For remote connection, it is assumed that the Opti
(or local) host. In that case, the host and port m
execution of the new optiSLang server are ignored.
"""
host = "127.0.0.1"
port = 5310
osl = Optislang(host=host, port=port)
osl.shutdown()

""" Basic usage
Use Optislang() with default arguments to create a
run_python_script to execute a native optiSLang Py
to start project execution and wait for it to fini
"""
osl = Optislang()
file_path = r"C:/Users/Username/my_scripts/myscrip
osl.run_python_file(path=script_path)
# osl.run_pyton_script("""print("hello world")""")
osl.start()
osl.save_copy("MyNewProject.opf")
osl.shutdown()

""" Open existing project
Use Optislang(project_path="/path/to/project.opf")
project is automatically saved, unless no_save arg
"""
osl = Optislang(project_path="/path/to/project.opf
osl.start()
osl.shutdown()
```

Examples

- **Launch optiSLang locally**

```
from ansys.optislang.core import Optislang

osl = Optislang()
print(osl)
osl.dispose()
```

- **Open and start existing project**

```
from ansys.optislang.core import Optislang

project_path = r'C:\MyDirectory\Sensitivity.opf'

osl = Optislang(ini_timeout=60, project_path=project_path)
osl.start()
osl.dispose()
```

Create new project

```
from ansys.optislang.core import Optislang

project_path = r'C:\MyDirectory\NewProject.opf'
python_file = r'C:\MyDirectory\oscillator_sensitivity_mop.py'

osl = Optislang(ini_timeout=60)
osl.new()
osl.save_as(project_path)
osl.run_python_file(python_file)

osl.dispose()
```

Check process state during run

```python
import time
from ansys.optislang.core import Optislang
from ansys.optislang.core.nodes import System


def print_node_info(node):
    name = node.get_name()
    type_ = node.get_type()
    status = node.get_status()
    print(name, type_, status)


def process_nodes(nodes):
    for node in nodes:
        print_node_info(node)
        if isinstance(node, System):
            process_nodes(node.get_nodes())
```

```python
project_path = r'C:\MyDirectory\Sensitivity.opf'

osl = Optislang(ini_timeout=60, project_path=project_path)
osl.reset()
osl.start(wait_for_finished=False)

project = osl.project
root_system = project.root_system
while root_system.get_status() == 'Running':
    nodes = root_system.get_nodes()
    process_nodes(nodes)
    time.sleep(1)

osl.dispose()
```

Evaluate designs

```
from ansys.optislang.core import Optislang

project_path = r'C:\MyDirectory\parametric_project.opf‘

osl = Optislang(ini_timeout=60, project_path=project_path)

root_system = osl.project.root_system
reference_design = root_system.get_reference_design()

reference_design.set_parameter_by_name(name="a", value=13)

result_design = root_system.evaluate_design(design=reference_design)
for p in result_design.responses:
    print(p.name, p.value)

osl.dispose()
```

End of presentation