PyDPF-Core cheat sheet

Version: 0.8.1 (stable) - API is subject to change

Import a model

Import a DPF model to see an overview of the content in its results file:

```
from ansys.dpf import core as dpf
model = dpf.Model(r"D:\Path\to\my\file.rst")
print(model)
```

Fetch metadata

Fetch all the results available in the results file:

```
metadata = model.metadata
print(metadata.result_info)
```

Print the time or frequency of the results:

```
print(metadata.time_freq_support)
```

Print available named selections:

```
print(metadata.available_named_selections)
```

Manage data sources

Manage results files by creating a DPF `DataSources` object from scratch or from a model:

```
from ansys.dpf import core as dpf
data_src = dpf.DataSources(r"D:\Path\to\my\file.rst")
# or
data_src = model.metadata.data_sources
```

Read results

Read displacement results by instantiating a displacement operator and then print the fields container:

```
disp_op = model.results.displacement()
disp = disp_op.outputs.fields_container()
print(disp)
```

Get field by Time ID = 1:

```
disp_1 = disp.get_field_by_time_id(1).data
print(disp_1)
```

Read the mesh

Get mesh information, the list of node IDs, and the list of element IDs:

```
mesh = metadata.meshed_region
print(mesh)
node_IDs = mesh.nodes.scoping.ids
element_IDs = mesh.elements.scoping.ids
```

Get a node and an element using their IDs:

```
# Node ID 1
node = mesh.nodes.node_by_id(1)
# Element ID 1
element = mesh.elements.element_by_id(1)
```

Scope results over time/space domains

Scope results over particular time IDs to make fetching data faster:

```
time_sets = [1, 3, 10]  # Data for time IDs
disp_op = model.results.displacement()
disp_op.inputs.time_scoping(time_sets)
```

Scope over particular elements:

```
eids = range(262, 272)  # Element scoping
element_scoping = dpf.Scoping(ids=eids)
element_scoping.location = dpf.locations.elemental

s_y = dpf.operators.result.stress_Y()
s_y.inputs.mesh_scoping.connect(element_scoping)
```

Retrieve a mesh from a scoping:

```
mesh_from_scoping_op = dpf.operators.mesh.from_scoping
    (mesh, element_scoping)
cropped_mesh = mesh_from_scoping_op.outputs.mesh()
```

Get a data field

Apply operators and extract the resulting data field:

```
norm_op2 = dpf.operators.math.norm_fc()
disp = model.results.displacement()
norm_op2.inputs.connect(disp.outputs)
field_norm_disp = norm_op2.outputs.fields_container()
    [0]
```

Plot contour results

Plot a field on its mesh and show the minimum and maximum values:

```
field_norm_disp.plot(show_max=True, show_min=True,
    title='Field', text='Field plot')
```

Take a screenshot from a specific camera position:

```
cpos = [(0.123, 0.095, 1.069), (-0.121, -0.149, 0.825)
    , (0.0, 0.0, 1.0)]
field.plot(off_screen=True, notebook=False, screenshot
    ='field_plot.png', cpos=cpos)
```

Workflows

Combine operators to create workflows:

```
disp_op = dpf.operators.result.displacement()
max_fc_op = dpf.operators.min_max.min_max_fc(disp_op)
workflow = dpf.Workflow()
workflow.add_operators([disp_op,max_fc_op])
workflow.set_input_name("data_sources", disp_op.inputs
    .data_sources)
workflow.set_output_name("min", max_fc_op.outputs.
    field_min)
workflow.set_output_name("max", max_fc_op.outputs.
    field_max)
```

Use the workflow:

```
data_src = dpf.data_sources.DataSources(r"D:\Path\to\
    my\file.rst")
workflow.connect("data_sources", data_src)
min = workflow.get_output("min", dpf.types.field)
max = workflow.get_output("max", dpf.types.field)
```

References from PyDPF-Core documentation

- Getting started
- Examples
- API reference
- Operators