# PyFluent cheat sheet

Version: 0.35.1

### Launch and exit a meshing session

```
import ansys.fluent.core as pyfluent
meshing = pyfluent.launch_fluent(
    mode=pyfluent.FluentMode.MESHING)
meshing.exit()
```

### Launch and exit a solver session

```
solver = pyfluent.launch_fluent(
    mode=pyfluent.FluentMode.SOLVER)
solver.exit()
```

### Dimension, Precision, Processor count, Product version

```
solver = pyfluent.launch_fluent(
    dimension=pyfluent.Dimension.THREE,
    precision=pyfluent.Precision.DOUBLE,
    processor_count=2,
    product_version=pyfluent.FluentVersion.v251
    )
```

### Connect to an existing instance of Fluent

```
fluent = pyfluent.connect_to_fluent(
    ip="127.0.0.1",
    port=50000,
    password="abcdefg")
```

### Watertight geometry meshing workflow

```
import ansys.fluent.core as pyfluent
from ansys.fluent.core import examples
import_file_name =
    examples.download_file('mixing_elbow.pmdb',
    'pyfluent/mixing_elbow')
meshing = pyfluent.launch_fluent(
    mode="meshing",
        precision=pyfluent.Precision.DOUBLE,
        processor_count=2
)
wt = meshing.watertight()
wt.import_geometry.file_name.set_state(
    import_file_name)
wt.import_geometry.length_unit.set_state('in')
wt.import_geometry()
```

### Add local sizing

```
wt.add_local_sizing.add_child_to_task()
wt.add_local_sizing()
```

### Generate surface mesh

```
csm = wt.create_surface_mesh
csmc = csm.cfd_surface_mesh_controls
csmc.max_size.set_state(0.3)
wt.create_surface_mesh()
```

### Describe geometry

```
wt.describe_geometry.update_child_tasks(
    setup_type_changed=False)
wt.describe_geometry.setup_type.set_state("The
    geometry consists of only fluid regions with
    no voids")
wt.describe_geometry.update_child_tasks(
    setup_type_changed=True)
wt.describe_geometry()
```

### Update boundaries

```
ub = wt.update_boundaries
ub.boundary_label_list.set_state(["wall-inlet"])
ub.boundary_label_type_list.set_state(["wall"])
ub.old_boundary_label_list.set_state(
    ["wall-inlet"])
ub.old_boundary_label_type_list.set_state(
    ["velocity-inlet"])
ub()
```

### Update regions

```
wt.update_regions()
```

### Add boundary layers

```
wt.add_boundary_layer.add_child_to_task()
wt.add_boundary_layer.bl_control_name.set_state("smooth-transition_1")
wt.add_boundary_layer.insert_compound_child_task()
wt.add_boundary_layer_child_1()
```

### Generate volume mesh

```
wt.create_volume_mesh.volume_fill.set_state(
    "poly-hexcore")
vfc = wt.create_volume_mesh.volume_fill_controls
vfc.hex_max_cell_length.set_state(0.3)
wt.create_volume_mesh()
```

### Switch to solution mode

```
solver = meshing.switch_to_solver()
```

### Boundary conditions

```
import ansys.fluent.core as pyfluent
from ansys.fluent.core import examples
file_name =
    examples.download_file("mixing_elbow.cas.h5",
    "pyfluent/mixing_elbow")
solver = pyfluent.launch_fluent()
solver.settings.file.read_case(
    file_name=file_name)
cold_inlet = pyfluent.VelocityInlet(solver,
    name="cold-inlet")
cold_inlet.momentum.velocity.set_state(0.4)
inlet_turbulence = cold_inlet.turbulence
turbulence_specification =
    inlet_turbulence.turbulence_specification
turbulence_specification.allowed_values()
turbulence_specification.set_state("Intensity and
    Hydraulic Diameter")
turbulent_intensity =
    inlet_turbulence.turbulent_intensity
turbulent_intensity.min(),
    turbulent_intensity.max()
turbulent_intensity.set_state(0.5)
inlet_turbulence.hydraulic_diameter.set_state("4
    [in]")
cold_inlet.thermal.temperature.set_state(293.15)
```

### Cell zone conditions

```
elbow_fluid =
    pyfluent.solver.FluidCellZone(solver,
    name="elbow-fluid")
elbow_fluid.laminar.set_state(True)
```

### Copy material from database

```
import ansys.fluent.core as pyfluent
from ansys.fluent.core import examples
```

```
file_name =
    examples.download_file("mixing_elbow.cas.h5",
    "pyfluent/mixing_elbow")
solver = pyfluent.launch_fluent()
solver.settings.file.read_case(file_name=file_name)
materials = pyfluent.Materials(solver)
fluids = materials.fluid
fluids.make_a_copy(from_="air",to="air-2")
air_copy = fluids["air-2"]
air_copy.viscosity.value.set_state(1.81e-05)
cz = solver.settings.setup.cell_zone_conditions
cz.fluid["elbow-fluid"].material.set_state("air-2")
```

### Access the object state using pprint

```
from pprint import pprint
pprint(air_copy.get_state(), width=1)
pprint(air_copy.viscosity.option.allowed_values(),
    width=1)
```

### Create new material

```
mysolid = materials.solid.create("mysolid")
mysolid.chemical_formula.set_state("SiO2")
mysolid.density.value.set_state(2650)
mysolid.specific_heat.value.set_state(1887)
mysolid.thermal_conductivity.value.set_state(7.6)
```

### Energy model

```
import ansys.fluent.core as pyfluent
from ansys.fluent.core import examples
file_name =
    examples.download_file("mixing_elbow.cas.h5",
    "pyfluent/mixing_elbow")
solver = pyfluent.launch_fluent()
solver.settings.file.read_case(file_name=file_name)
energy = pyfluent.solver.Energy(solver)
energy.enabled.get_state()
from pprint import pprint
pprint(energy.get_state(), width=1)
energy.enabled.set_state(False)
pprint(energy.get_state(), width=1)
energy.enabled.set_state(True)
pprint(energy.get_state(), width=1)
energy.viscous_dissipation.set_state(True)
pprint(energy.get_state(), width=1)
```

### Viscous model

```
vs = pyfluent.solver.Viscous(solver)
from pprint import pprint
pprint(vs.get_state(), width=1)
pprint(vs.model.allowed_values(), width=1)
vs.options.corner_flow_correction.is_active()
vs.model.set_state('k-epsilon')
vs.options.corner_flow_correction.is_active()
vs.k_epsilon_model.get_state()
vs.k_omega_model.is_active()
vs.k_epsilon_model.allowed_values()
vs_ops = vs.options
vs_ops.production_kato_launder_enabled.is_active()
vs_ops.production_kato_launder_enabled.get_state()
vs.k_epsilon_model.set_state("realizable")
vs_ops.production_kato_launder_enabled.is_active()
```

### Discrete phase model

```
dpm = pyfluent.solver.DiscretePhase(solver)
dpm_models = dpm.physical_models
dpm_vmf = dpm_models.virtual_mass_force
dpm_vmf.enabled.get_state()
dpm_vmf.virtual_mass_factor.is_active()
dpm_vmf.enabled.set_state(True)
dpm_vmf.virtual_mass_factor.get_state()
```

### Radiation model

```
rn = pyfluent.solver.Radiation(solver)
from pprint import pprint
pprint(rn.get_state(), width=1)
pprint(rn.model.allowed_values(), width=1)
rn.model.set_state("monte-carlo")
pprint(rn.get_state(), width=1)
rn.monte_carlo.number_of_histories.set_state(1e7)
rn.multiband.create("solar").set_state({
   "start": 0,
   "end": 2.8,
})
rn.multiband.create("thermal-ir").set_state({
   "start": 2.8,
   "end": 100,
})
radiation_freq = rn.solve_frequency
pprint(radiation_freq.get_state(), width=1)
pprint(rn.get_state(), width=1)
```

### Species model

```
solver.settings.file.read_case(file_name=file_name)
species = pyfluent.solver.Species(solver)
species.get_state()
from pprint import pprint
pprint(species.model.option.allowed_values(),
    width=1)
species.model.option.set_state("species-transport")
pprint(species.get_state(), width=1)
species.model.material.get_state()
species.model.material.allowed_values()
```

### Battery model

```
battery = pyfluent.solver.Battery(solver)
battery.enabled.set_state(True)
battery.solution_method.allowed_values()
```

### Steady or transient solution model

```
setup = pyfluent.solver.Setup(solver)
solver_time = setup.general.solver.time
solver_time.get_state()
solver_time.allowed_values()
solver_time.set_state("unsteady-1st-order")
```

### Pressure-based or density-based solver

```
setup = pyfluent.solver.Setup(solver)
solver_type = setup.general.solver.type
solver_type.get_state()
solver_type.allowed_values()
solver_type.set_state("density-based-explicit")
solver_type.get_state()
```

### Velocity coupling scheme and gradient options

```
methods = pyfluent.solver.Methods(solver)
flow_scheme = methods.p_v_coupling.flow_scheme
flow_scheme.allowed_values()
flow_scheme.set_state("Coupled")
gradient_scheme = methods.gradient_scheme
gradient_scheme.allowed_values()
gradient_scheme.set_state("green-gauss-node-based")
```

### Solution controls

```
controls = pyfluent.solver.Controls(solver)
pvc = controls.p_v_controls
emur = pvc.explicit_momentum_under_relaxation
emur.min()
emur.max()
emur.set_state(0.4)
flow_courant_number = pvc.flow_courant_number
flow_courant_number.min()
flow_courant_number.max()
flow_courant_number.set_state(0.3)
```

### Create a report definition

```
rep_defs =
    pyfluent.solver.ReportDefinitions(solver)
surface = rep_defs.surface
defn_name = "outlet-temp-avg"
surface[defn_name] = {}
out_temp = surface[defn_name]
out_temp.report_type.set_state("surface-massavg")
out_temp.field.set_state("temperature")
```

### Initialize and solve

```
solution = solver.settings.solution
solution.initialization.hybrid_initialize()
solution.run_calculation.iterate(iter_count=100)
```

### CaseFile reader

```
from ansys.fluent.core import examples
from ansys.fluent.core.filereader.case_file import
    CaseFile
case_file_name = examples.download_file(
    "Static_Mixer_Parameters.cas.h5",
    "pyfluent/static_mixer")
reader = CaseFile(case_file_name=case_file_name)
reader.precision()
reader.num_dimensions()
{p.name: p.value for p in
    reader.input_parameters()}
{p.name: p.units for p in
    reader.output_parameters()}
```

### Additionl features

```
reader = CaseFile(
    project_file_name="Dir1/Dir2/project.flprj")
reader.rp_vars()
reader.config_vars()
```

### Extract mesh data

```
from ansys.fluent.core import examples
from ansys.fluent.core.filereader.case_file import
    CaseFile
case_file_name =
    examples.download_file("elbow1.cas.h5",
    "pyfluent/file_session")
reader = CaseFile(case_file_name=case_file_name)
reader.get_mesh().get_surface_ids()
reader.get_mesh().get_surface_names()
reader.get_mesh().get_surface_locs(3)
reader.get_mesh().get_connectivity(3)
reader.get_mesh().get_vertices(3)
```

### DataFile reader

```
from ansys.fluent.core import examples
from ansys.fluent.core.filereader.data_file import
    DataFile
from ansys.fluent.core.filereader.case_file import
    CaseFile
data_file_name =
    examples.download_file("elbow1.dat.h5",
    "pyfluent/file_session")
reader = DataFile(
    data_file_name=data_file_name,
    case_file_handle=CaseFile(case_file_name))
reader.case_file
reader.variables()
reader.get_phases()
reader.get_face_variables("phase-1")
```

### Single-phase FileSession

```
from ansys.fluent.core import examples
from ansys.fluent.core.file_session import
    FileSession
case_file_name =
    examples.download_file("elbow1.cas.h5",
    "pyfluent/file_session")
data_file_name =
    examples.download_file("elbow1.dat.h5",
    "pyfluent/file_session")
fs = FileSession()
fs.read_case(case_file_name)
fs.read_data(data_file_name)
fs.fields.field_info.get_scalar_field_range("SV_T")
fs.fields.field_info.get_surfaces_info()
fs.fields.field_info.get_scalar_fields_info()
```

### Multiphase FileSession

```
from ansys.fluent.core import examples
from ansys.fluent.core.file_session import
    FileSession
case_file_name = examples.download_file(
    "mixing_elbow_mul_ph.cas.h5",
    "pyfluent/file_session")
data_file_name = examples.download_file(
    "mixing_elbow_mul_ph.dat.h5",
    "pyfluent/file_session")
fs = FileSession()
fs.read_case(case_file_name)
fs.read_data(data_file_name)
fs.fields.field_info.get_scalar_field_range(
    "phase-2:SV_P")
fs.fields.field_info.get_scalar_fields_info()
```

### Post-processing using ansys-fluent-visualization

```
from ansys.fluent.visualization import set_config
set_config(blocking=True,
    set_view_on_display="isometric")
import ansys.fluent.core as pyfluent
from ansys.fluent.visualization.matplotlib import
    Plots
from ansys.fluent.visualization.pyvista import
    Graphics
from ansys.fluent.core.file_session import
    FileSession
fileSession=FileSession()
fileSession.read_case("elbow1.cas.h5")
fileSession.read_data("elbow1.dat.h5")
graphics = Graphics(session=fileSession)
```

### Display mesh at wall

```
mesh1 = graphics.Meshes["mesh-1"]
mesh1.show_edges = True
mesh1.surfaces_list = [ "wall"]
mesh1.display("w1")
```

### Display temperature contour at symmetry

```
contour1 = graphics.Contours["mesh-1"]
contour1.node_values = False
contour1.field = "SV_T"
contour1.surfaces_list = ['symmetry']
contour1.display('w2')
```

### Display velocity vector data at symmetry and wall

```
velocity_vector =
    graphics.Vectors["velocity-vector"]
velocity_vector.field = "SV_T"
velocity_vector.surfaces_list = ['symmetry',
    'wall']
velocity_vector.display("w3")
```

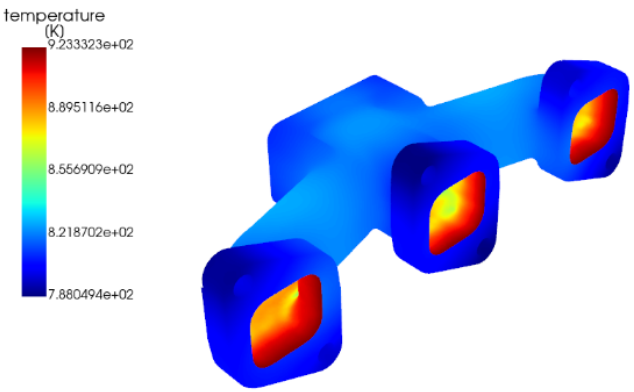


Figure 1: Temperature contour

### Accessing field data objects

```
import ansys.fluent.core as pyfluent
from ansys.fluent.core import examples
case_path = examples.download_file(
    file_name="exhaust_system.cas.h5",
    directory="pyfluent/exhaust_system"
)
data_path = examples.download_file(
    file_name="exhaust_system.dat.h5",
    directory="pyfluent/exhaust_system"
)
solver = pyfluent.launch_fluent(
    mode=pyfluent.FluentMode.SOLVER)
solver.settings.file.read_case_data(
    file_name=case_path
)
field_data = solver.fields.field_data
```

### Get surface data

```
from ansys.fluent.core.services.field_data import
    SurfaceDataType
data = field_data.get_surface_data(
    surfaces=["inlet"],
    data_types=[SurfaceDataType.Vertices]
)
data["inlet"][SurfaceDataType.Vertices].shape
data["inlet"][SurfaceDataType.Vertices][5]
faces_normal_and_centroid_data =
    field_data.get_surface_data(
    data_types=[SurfaceDataType.FacesNormal,
    SurfaceDataType.FacesCentroid],
    surfaces=["inlet"]
)
inlet = faces_normal_and_centroid_data["inlet"]
inlet[SurfaceDataType.FacesNormal].shape
inlet[SurfaceDataType.FacesCentroid][15]
faces_connectivity_data =
    field_data.get_surface_data(
    data_types=[SurfaceDataType.FacesConnectivity],
    surfaces=["inlet"]
)
inlet = faces_connectivity_data["inlet"]
inlet[SurfaceDataType.FacesConnectivity][5]
```

### Get scalar field data

```
abs_press_data = field_data.get_scalar_field_data(
    field_name="absolute-pressure",
    surfaces=["inlet"]
)
abs_press_data["inlet"].shape
abs_press_data["inlet"][120]
```

### Get vector field data

```
velocity_vector_data =
    field_data.get_vector_field_data(
    field_name="velocity",
    surfaces=["inlet", "inlet1"]
)
velocity_vector_data["inlet"].shape
velocity_vector_data["inlet1"].shape
```

### Get pathlines field data

```
path_lines_data =
    field_data.get_pathlines_field_data(
    field_name="velocity",
    surfaces=["inlet"]
)
path_lines_data["inlet"]["vertices"].shape
len(path_lines_data["inlet"]["lines"])
path_lines_data["inlet"]["velocity"].shape
path_lines_data["inlet"]["lines"][100]
```

### Accessing field info objects

```
import ansys.fluent.core as pyfluent
solver = pyfluent.launch_fluent(
    mode=pyfluent.FluentMode.SOLVER)
solver.settings.file.read(file_type="case-dats",
    file_name=mixing_elbow_case_path)
init = solver.settings.solution.initialization
init.hybrid_initialize()
field_info = solver.fields.field_info
```

### Get fields info and range

```
field_info.get_scalar_fields_info()
field_info.get_scalar_field_range("cell-weight")
```

### Get vector fields and surfaces info

```
field_info.get_vector_fields_info()
field_info.get_surfaces_info()
```

### Accessing reduction functions

```
import ansys.fluent.core as pyfluent
from ansys.fluent.core.solver.function import
    reduction
from ansys.fluent.core.examples import
    download_file

solver1 = pyfluent.launch_fluent(
    mode=pyfluent.FluentMode.SOLVER
)
case_path = download_file(
    file_name="exhaust_system.cas.h5",
    directory="pyfluent/exhaust_system")
data_path = download_file(
    file_name="exhaust_system.dat.h5",
    directory="pyfluent/exhaust_system")
solver1.settings.file.read_case_data(
    file_name=case_path
)

solver2 = pyfluent.launch_fluent(
    mode=pyfluent.FluentMode.SOLVER
)
case_path = download_file(
    "elbow1.cas.h5",
    "pyfluent/file_session"
)
data_path = download_file(
    "elbow1.dat.h5",
    "pyfluent/file_session"
)
solver2.settings.file.read_case_data(
    file_name=case_path
)

solver = solver1
```

### Usage of reduction context

```
import ansys.fluent.core as pyfluent
init = solver.settings.solution.initialization
init.hybrid_initialize()
inlet = pyfluent.VelocityInlets(solver)
solver.fields.reduction.area(
  locations=[inlet["inlet1"]]
)
solver.fields.reduction.area(
    locations=["inlet1"],
    ctxt=solver)
```

### Current reduction capabilities

```
reduction.area(locations)
reduction.area_average(expression, locations)
reduction.area_integral(expression, locations)
reduction.volume(locations)
reduction.volume_average(expression, locations)
reduction.volume_integral(expression, locations)
reduction.centroid(locations)
reduction.force(locations)
reduction.pressure_force(locations)
reduction.viscous_force(locations)
reduction.moment(expression, locations)
reduction.count(locations)
reduction.count_if(condition, locations)
reduction.minimum(expression, locations)
reduction.maximum(expression, locations)
reduction.mass_average(expression, locations)
reduction.mass_integral(expression, locations)
reduction.mass_flow_average_absolute(expression,
    locations)
reduction.mass_flow_average(expression, locations)
reduction.mass_flow_integral(expression, locations)
reduction.sum(expression, locations, weight)
reduction.sum_if(expression, condition, locations,
    weight)
```

### Reduction example use cases

```
import ansys.fluent.core as pyfluent
inlet = pyfluent.VelocityInlets(solver)

area_inlet_1 = solver.fields.reduction.area(
  locations=[inlet["inlet1"]])
area_inlet = solver.fields.reduction.area(
  locations=[inlet])
solver.fields.reduction.centroid(
  locations=[inlet["inlet2"]])
po_1 = pyfluent.PressureOutlets(solver1)
po_2 = pyfluent.PressureOutlets(solver2)
solver.fields.reduction.minimum(
    expression="AbsolutePressure",
    locations=[po_1, po_2],
)
solver.fields.reduction.sum(
  expression="AbsolutePressure",
```

```
  locations=[inlet],
  weight="Area")
solver.fields.reduction.sum_if(
  expression="AbsolutePressure",
  condition="AbsolutePressure > 0[Pa]",
  locations=[inlet],
  weight="Area")
```

### Accessing solution variable objects

```
import ansys.fluent.core as pyfluent
from ansys.fluent.core import examples
import_filename =
    examples.download_file("mixing_elbow.msh.h5",
    "pyfluent/mixing_elbow")
solver = pyfluent.launch_fluent(
    mode=pyfluent.FluentMode.SOLVER)
solver.settings.file.read(file_type="case",
    file_name=import_filename)
solution_variable_info =
    solver.fields.solution_variable_info
solution_variable_data =
    solver.fields.solution_variable_data
```

### Get zone information

```
zones_info =
    solution_variable_info.get_zones_info()
zones_info.domains
zones_info.zones
zone_info = zones_info['wall']
zone_info
zone_info.name
zone_info.count
zone_info.zone_id
zone_info.zone_type
```

### Get solution variable information

```
wall_fluid_info =
    solution_variable_info.get_variables_info(
    zone_names=['wall' , "fluid"],
    domain_name="mixture")
wall_fluid_info.solution_variables
solution_variable_info_centroid =
    wall_fluid_info['SV_CENTROID']
solution_variable_info_centroid
solution_variable_info_centroid.name
solution_variable_info_centroid.dimension
solution_variable_info_centroid.field_type
```

### Get solution variable data

```
sv_t_wall_fluid= solution_variable_data.get_data(
    variable_name="SV_T",
    zone_names=["fluid", "wall"],
    domain_name="mixture")
sv_t_wall_fluid.domain
sv_t_wall_fluid.zones
fluid_temp = sv_t_wall_fluid['fluid']
fluid_temp.size
fluid_temp.dtype
fluid_temp
```

### Set solution variable data

```
wall_temp_array =
    solution_variable_data.create_empty_array(
    "SV_T", "wall", "mixture")
fluid_temp_array =
    solution_variable_data.create_empty_array(
    "SV_T", "fluid", "mixture")
wall_temp_array[:] = 500
fluid_temp_array[:] = 600
zone_names_to_solution_variable_data =
    {'wall':wall_temp_array,
    'fluid':fluid_temp_array}
solution_variable_data.set_data(
    variable_name="SV_T",
    zone_names_to_data=zone_names_to_solution_variabl
    domain_name="mixture")
```

### Multiple requests in a single transaction

```
transaction =
    solver.fields.field_data.new_transaction()

transaction.add_surfaces_request(
    surfaces=[1],
    data_types=[SurfaceDataType.Vertices,
    SurfaceDataType.FacesCentroid]
)
transaction.add_scalar_fields_request(
    surfaces=[1, 2],
    field_name="pressure",
    node_value=True,
    boundary_value=True
)
transaction.add_vector_fields_request(
    surfaces=[1, 2], field_name="velocity"
)

payload_data = transaction.get_fields()
```

### Field data allowed values

```
sfd = field_data.get_scalar_field_data
sfd.field_name.allowed_values()
sfd.surface_name.allowed_values()

transaction = field_data.new_transaction()
asfr = transaction.add_scalar_fields_request
asfr.field_name.allowed_values()

sd = field_data.get_surface_data
sd.surface_ids.allowed_values()
```

### Monitor convergence of a solution

```
# get started with case and data loaded
import ansys.fluent.core as pyfluent
from ansys.fluent.core import examples
import pandas as pd
from tabulate import tabulate
solver =
    pyfluent.launch_fluent(start_transcript=False)
import_case = examples.download_file(
    file_name="exhaust_system.cas.h5",
        directory="pyfluent/exhaust_system"
)
import_data = examples.download_file(
    file_name="exhaust_system.dat.h5",
        directory="pyfluent/exhaust_system"
)
solver.file.read_case_data(file_name=import_case)
# check the active report plot monitors using the
    settings relevant object
solver.settings.solution.monitor.report_plots()
# initialize so that monitors object is usable
solver.solution.initialization.hybrid_initialize()
# check which monitors are available
sorted(solver.monitors.get_monitor_set_names())
# create and register a callback function that will
def display_monitor_table(
    monitor_set_name="mass-bal-rplot"):
    def display_table():
        data =
            solver.monitors.get_monitor_set_data(
            monitor_set_name=monitor_set_name)
        # extract iteration numbers
        iterations = data[0]
        # filter out additional callbacks
        if len(iterations) >
            display_table.iter_count:
            display_table.iter_count =
                len(iterations)
            # extract results
            results = data[1]
            # create a DataFrame
            df = pd.DataFrame(results,
                index=iterations)
            df.index.name = 'Iteration'
            df.reset_index(inplace=True)
            # The streamed data contains
                duplicates, so eliminate them
            df = df.drop_duplicates(subset=
            'Iteration')
            print(tabulate(df, headers='keys',
                tablefmt='psql'))
        display_table.iter_count = 0
    return display_table

register_id = solver.monitors.register_callback(
    display_monitor_table())
# run the solver and see the full tabulated
    monitor data on each iteration
solver.solution.run_calculation.iterate(
    iter_count=10)
```

### Observing events

```
from ansys.fluent.core import MeshingEvent,
    SolverEvent
def on_case_loaded(session, event_info):
    print("Case loaded. Index = ",
        event_info.index)
callback = meshing.events.register_callback(
    MeshingEvent.CASE_LOADED, on_case_loaded)
def on_iteration_ended(session, event_info):
    print("Iteration ended. Index = ",
        event_info.index)
callback_id = solver.events.register_callback(
    SolverEvent.ITERATION_ENDED,
        on_iteration_ended)
```

### Transfer a case or mesh file between PyFluent sessions

```
import ansys.fluent.core as pyfluent
from ansys.fluent.core.examples import
    download_file
from ansys.fluent.core.utils.data_transfer import
    transfer_case
mesh_file_name = download_file(
    "mixing_elbow.msh.h5",
    "pyfluent/mixing_elbow"
)
pure_meshing_session = pyfluent.launch_fluent(
    mode=pyfluent.FluentMode.PURE_MESHING
)
pure_meshing_session.tui.file.read_mesh(
    import_file_name
)
solver_session = pyfluent.launch_fluent(
    mode=pyfluent.FluentMode.SOLVER
)
transfer_case(
    source_instance=meshing, solvers=[solver],
    file_type="mesh", file_name_stem='',
    num_files_to_try=1, clean_up_temp_file=True,
    overwrite_previous=True
)
```

### PyAnsys Units to work in arbitrary physical examples

```
from ansys.units import Quantity
bc = solver.settings.setup.boundary_conditions
vi = bc.velocity_inlet
hyd_dia =
    vi["hot-inlet"].turbulence.hydraulic_diameter
hyd_dia.set_state(.02)
hyd_dia.get_state()
hyd_dia.state_with_units()
hyd_dia.set_state(Quantity(15, "mm"))
hyd_dia.state_with_units()
diam = hyd_dia.as_quantity()
diam
diam = diam * 2
diam
hyd_dia.set_state(diam)
hyd_dia.as_quantity()
```

### Local file transfer service

```
import ansys.fluent.core as pyfluent
from ansys.fluent.core import examples
from ansys.fluent.core.utils.file_transfer_service
    import StandaloneFileTransferStrategy

mesh_file_name = examples.download_file(
    "mixing_elbow.msh.h5",
    "pyfluent/mixing_elbow")
meshing_session = pyfluent.launch_fluent(
    mode=pyfluent.FluentMode.MESHING,
    file_transfer_service=
    StandaloneFileTransferStrategy())
meshing_session.upload(
    file_name=mesh_file_name,
    remote_file_name="elbow.msh.h5")
meshing_session.meshing.File.ReadMesh(
    FileName="elbow.msh.h5")
meshing_session.meshing.File.WriteMesh(
    FileName="write_elbow.msh.h5")
meshing_session.download(
    file_name="write_elbow.msh.h5",
    local_directory="<local_directory_path>")
```

### Remote file transfer service

```
import ansys.fluent.core as pyfluent
from ansys.fluent.core import examples
from ansys.fluent.core.utils.file_transfer_service
    import ContainerFileTransferStrategy

case_file_name = examples.download_file(
    "mixing_elbow.cas.h5",
    "pyfluent/mixing_elbow")
solver_session = pyfluent.launch_fluent(
    mode=pyfluent.FluentMode.SOLVER,
    file_transfer_service=
    ContainerFileTransferStrategy())
solver_session.upload(
    file_name=case_file_name,
    remote_file_name="elbow.cas.h5")
solver_session.file.read_case(
    file_name="elbow.cas.h5")
solver_session.file.write_case(
    file_name="write_elbow.cas.h5")
solver_session.download(
    file_name="write_elbow.cas.h5",
    local_directory="<local_directory_path>")
```

### Record Fluent interactions as Python scripts (journals)

```
solver.journal.start(
    file_name="pyfluent_journal.py")
solver.journal.stop()
```

### PyFluent logging functionality

```
import ansys.fluent.core as pyfluent
config_dict = pyfluent.logger.get_default_config()
config_dict['handlers']['pyfluent_file'][
    'filename'] = 'test.log'
pyfluent.logger.enable(custom_config=config_dict)
pyfluent.logger.list_loggers()
logger = pyfluent.logger.get_logger(
    'pyfluent.networking')
logger.setLevel('ERROR')
pyfluent.logger.set_global_level('DEBUG')
```

### API search

```
# Semantic search
import ansys.fluent.core as pyfluent
pyfluent.search("font")

# Whole word search
pyfluent.search("ApplicationFontSize",
    match_whole_word=True)

# Wildcard pattern search
pyfluent.search("local*", wildcard=True)
```

### Containerization of Fluent

```
# Within the PyFluent source navigate to the
    `docker` directory.
cd pyfluent/docker

# Copy needed files
python copy_ansys_files.py <path to 'ansys_inc'
    directory> <path to 'docker/fluent_<version>'
    directory>

# Build the Docker image
sudo docker build -t ansys_inc <path to
    'docker/fluent_<version>' directory>
```

### Run Docker container using the command line

```
# Solver mode
sudo docker run -it --name ansys-inc -e
    ANSYSLMD_LICENSE_FILE=<license file or
    server> ansys_inc 3ddp -gu
# Meshing mode
sudo docker run -it --name ansys-inc -e
    ANSYSLMD_LICENSE_FILE=<license file or
    server> ansys_inc 3ddp -gu -meshing
```

### Run Docker container using PyFluent

```
import os
import ansys.fluent.core as pyfluent
os.environ["ANSYSLMD_LICENSE_FILE"] = "<license
    file or server>"

custom_config = {
    'fluent_image': 'ansys_inc:latest',
    'mount_source': f"{os.getcwd()}",
    'auto_remove': False}

solver = pyfluent.launch_fluent(
    container_dict=custom_config)
```

### Scheme code evaluation

```
import ansys.fluent.core as pyfluent
session.scheme.exec(('(ti-menu-load-string
    "/report/system/proc-stats")',))
# Returns TUI output string
session.scheme.eval("(+ 2 3)")
session.scheme.eval("(rpgetvar 'mom/relax)")
```