A Pythonic way to use YAC

Jul 18 2024 - Nils-Arne Dreier - natESM Training

Why Python?!

- **simple**
    - accessible for *beginners*
    - rapid prototyping
- **open source** and active community
- extensive **libraries** – *batteries included*
    - scientific computing: `numpy`, `scipy`, …
    - visualization: `matplotlib`, `cartopy`, …
    - analytics: `pandas`, `xarray`, …
    - machine learning: `pytorch`, `tensorflow`, …
    - …

YAC python bindings

- **slim** python wrappers around `yac.h`
  - YAC IDs → classes
- generated with **cython**
- `./configure --enable-python-bindings`
- depends on numpy (data buffers)

Applications

- in-situ visualizations
- hiopy: **hi**erarchical **o**utput with **py**thon
- asynchronous input - e.g. ozone and aerosols
- tropical cyclone tracker

planned:

- model - sensor comparison

HowTo: definitions

```python
from yac import *
yac = YAC()

comp = yac.def_comp("python_component")
lon = np.linspace(0, 2*np.pi, 360, endpoint=False)
lat = np.linspace(0, np.pi, 180)
grid = Reg2dGrid("python_grid", lon, lat)
points = grid.def_points(Location.CORNER, lon, lat)

field = Field.create("tas", comp, points, collection_size=1,
                     timestep="PT3H", timeunit=TimeUnit.ISO_FORMAT)
```

HowTo: configure coupling

- in the API

```
nnn = InterpolationStack()
nnn.add_nnn(NNNReductionType.AVG, 1, 1.)

yac.def_couple("atmo", "icon_atmos_grid", "tas"
               "python_component", "python_grid", "tas",
               coupling_timestep="PT3H", timeunit=TimeUnit.ISO_FORMAT,
               time_reduction=Reduction.TIME_NONE, interp_stack=nnn)
```

- *or in a yaml file*

HowTo: synchronization

config synchronization:

```
yac.sync_def()
```

access metadata

```
for comp_name in yac.component_names:
    print(yac.get_component_metadata(comp))
    for field_name in yac.get_field_names(comp_name, "some_grid"):
        print(yac.get_field_metadata(comp_name, "some_grid", field_name))
```

end definition phase:

```
yac.enddef()
```

HowTo: get/put

```python
data = None
for t in range(no_timesteps):
    data, info = field.get(data)
    ## Do anything with data
```

Coupling modes: MPMD

- *parallel coupling*
- MPI MPMD paradigm, e.g.

```
mpirun -n 4 ./icon : -n 1 python myscript.py
```

- runs a python process next to the ICON processes

Output coupling

- ICON can be coupled to a generic “output component”:

```
&coupling_mode_nml
  coupled_to_output = .TRUE.
```

- registers **all** suitable variables in ICON variable list at YAC
- in the timeloop: calls `yac_fput` for all **coupled** fields

Coupling modes: SPMD (with ComIn)

- *sequential coupling*
- *ComIn* allows to embed python into ICON
- requires ComIn version 0.2 (to be released)
- needs some special linker flags

Coupling modes: SPMD - example

```python
comin.var_request_add(...)

icon_comp = yac.predef_comp(...)
icon_grid = UnstructuredGrid(...)
if rank == 0:
    rank0_comp = yac.predef_comp(...)
    regular_grid = Reg2dGrid(...)

@comin.register_callback(comin.EP_SECONDARY_CONSTRUCTOR)
def secondary_constructor():
    comin_var = comin.var_get(...)

@comin.register_callback(comin.EP_ATM_YAC_DEF_COMP_AFTER)
def def_comp_after():
    yac_icon_field = Field.create(...)
    if rank == 0:
        yac_rank0_field = Field.create(...)
    yac.def_couple(...)

@comin.register_callback(comin.EP_ATM_TIME_LOOP_START)
def time_loop_start():
    if rank == 0:
        yac_rank0_field.put(rank0_data)
    comin_var_np = numpy.asarray(comin_var)
    comin_var_np[:], info = yac_icon_field.get()
```

- Distribute global data on a regular grid to ICON
- **ComIn**
  - access to ICONs yac instance
  - access to ICON variables
  - callbacks in the ICON timeloop
- **YAC**
  - redistribution
  - interpolation

Questions?

Hands-On!

https://gitlab.dkrz.de/YAC/2407_tutorial