ALFAsim

ESSS

1 ALFAsim

This document aims to describe the procedure to run ALFAsim simulations using directly the executable, from an `.alfacase` input file, bypassing the graphic user interface. The use of the API from ALFAsim-sdk to access the simulation output is also described.

1.1 Alfasim Runner

[1]:
```
def execute_simulator(alfasim_runner: str, alfacase_file: str,␣
↪alfasim_plugins_dir: str) -> int:
    """
    Execute ALFAsim simulation bypassing the GUI from an .alfacase input file
    @param alfasim_runner:
        Path of ALFAsim executable file
    @param working_dir:
        Path of the directory containing the .alfacase input file
    @param alfacase_file:
        Path of the .alfacase input file
    @param alfasim_plugins_dir:
        Path of the directory containing the user plugins
    """
    import os
    import subprocess
    from pathlib import Path

    env = os.environ.copy()

    # Setting env variables needed
    env.pop('LD_LIBRARY_PATH', None)
    env['ALFASIM_PID_FILE_SECRET'] = 'w'
    env['ALFASIM_PLUGINS_DIR'] = alfasim_plugins_path
    env['PYTHONUNBUFFERED'] = "1"

    omp_flag = "--number-of-threads=0"
    omp_num_threads = os.getenv("OMP_NUM_THREADS")
    if omp_num_threads:
        env['OMP_NUM_THREADS'] = omp_num_threads
        omp_flag = ""
```

```
    alfacase_path = Path(alfacase_file)
    alfacase_filename = alfacase_path.stem

    """
    Prepare directory used to store simulation files. Running simulation␣
 ↪directly from cmd
    bypasses part of the code responsible for manageing this directory so it is
    done manually here.
    """
    data_dir = alfacase_path.parent / (alfacase_filename + ".data")
    if data_dir.exists():
        for item in data_dir.iterdir():
            try:
                if item.is_file() or item.is_symlink():
                    item.unlink()
                elif item.is_dir():
                    shutil.rmtree(item)
            except Exception as e:
                print(f"Failed to delete {item}. Reason: {e}")
    else:
        data_dir.mkdir(parents=True)

    cmd = [
        alfasim_runner,
        "--run-simulation",
        "--alfacase-file",
        alfacase_file,
        omp_flag,
    ]

    output = subprocess.run(
        cmd,
        env=env,
    )

    print(f'ALFAsim return code = {output}')

    return output.returncode
```

1.2 Setup paths and run simulation

[5]:
```
# Path to the ALFAsim executable
alfasim_path = "W:\\ALFAsim_app\\24.2\\ALFAsim\\bin\\ALFAsim.exe"

# Plugin instalation path. Default is "C:\\Users\\<user>\\.alfasim_plugins"
alfasim_plugins_path = "C:\\Users\\<user>\\.alfasim_plugins"
```

```
# Directory containing the .alfacase input file
alfacase_file = "W:\\data\\command_line\\simple_case.alfacase"

return_code = execute_simulator(alfasim_path, alfacase_file,␣
 ↪alfasim_plugins_path)
```

```
ALFAsim return code =
CompletedProcess(args=['W:\\ALFAsim_app\\24.2\\ALFAsim\\bin\\ALFAsim.exe', '--
run-simulation', '--alfacase-file',
'W:\\data\\command_line\\simple_case.alfacase', '--number-of-threads=0'],
returncode=0)
```

1.3 Getting results

Alfasim-SDK provides an API to handle ALFAsim simulation outputs through the class `Results`. The ALFAsim-SDK is a Python package that helps developers in the process to create a Plugin for ALFAsim and access simulation results.

It is recommended installing ALFAsim-SDK using the conda package manager. With the commands below, a conda environment will be created with ALFAsim-SDK installed on it:

```
>>> conda create -n sdk_env
>>> conda activate sdk_env
>>> pip install alfasim-sdk
```

Alternatively, an environment.yml could be added to your project with `alfasim-sdk` declared at pip dependencies. For more details, see Getting started with conda.

[6]:
```
from alfasim_sdk.result_reader.reader import Results
from pathlib import Path

alfacase_path = Path(alfacase_file)
alfacase_name = alfacase_path.stem
results_path = alfacase_path.parent / (alfacase_name + ".data")

results = Results(results_path)
```

The method `list_profiles` lists all the profile curves available in the output file

[7]:
```
results.list_profiles()
```

[7]:
```
[ProfileMetadata(property_name='elevation', element_name='Conn 1',
timesteps_count=12),
 ProfileMetadata(property_name='flow pattern', element_name='Conn 1',
timesteps_count=12),
 ProfileMetadata(property_name='gas phase volume fraction', element_name='Conn
1', timesteps_count=12),
 ProfileMetadata(property_name='holdup', element_name='Conn 1',
timesteps_count=12),
```

```
 ProfileMetadata(property_name='liquid mass flow rate', element_name='Conn 1',
timesteps_count=12),
 ProfileMetadata(property_name='liquid volumetric flow rate std',
element_name='Conn 1', timesteps_count=12),
 ProfileMetadata(property_name='mixture temperature', element_name='Conn 1',
timesteps_count=12),
 ProfileMetadata(property_name='oil phase volume fraction', element_name='Conn
1', timesteps_count=12),
 ProfileMetadata(property_name='pressure', element_name='Conn 1',
timesteps_count=12),
 ProfileMetadata(property_name='total gas mass flow rate', element_name='Conn
1', timesteps_count=12),
 ProfileMetadata(property_name='total gas volumetric flow rate std',
element_name='Conn 1', timesteps_count=12),
 ProfileMetadata(property_name='total mass flow rate', element_name='Conn 1',
timesteps_count=12),
 ProfileMetadata(property_name='total oil mass flow rate', element_name='Conn
1', timesteps_count=12),
 ProfileMetadata(property_name='total oil volumetric flow rate std',
element_name='Conn 1', timesteps_count=12),
 ProfileMetadata(property_name='total water mass flow rate', element_name='Conn
1', timesteps_count=12),
 ProfileMetadata(property_name='total water volumetric flow rate std',
element_name='Conn 1', timesteps_count=12),
 ProfileMetadata(property_name='water phase volume fraction', element_name='Conn
1', timesteps_count=12)]
```

The method `get_profile_curve` can be used to retrieve the curve of each property available in output. It receives the propery name, the name of the element in the network and the time step counter. Values available are shown above, using `list_profiles`. It returns a `Curve` object that contains the domain, image and unit of the property.

[8]:
```
temperature_curve = results.get_profile_curve( "mixture temperature", "Conn 1",␣
↪11)
print(temperature_curve)
```

```
Curve(K, m)[(323.15, 0.0) (322.54017613003566, 100.0) (322.45712864396, 200.0)
(322.6030903835268, 300.0) (322.7562204385999, 400.0) (322.93503047960786,
500.0) (322.98986207742433, 600.0) (323.0351864602084, 700.0)
(323.07695079965663, 800.0) (323.106744734592, 900.0) (323.106744734592,
1000.0)]
```

The domain and image of the curve are accessed with `GetDomain` and `GetImage`, respectively. They return an `Array` object from barril:

[9]:
```
temperature_domain = temperature_curve.GetDomain()
temperature_image = temperature_curve.GetImage()
```

```
# Array with domaing values and unit
print(temperature_domain)

# Array with image values and unit
print(temperature_image)

 # numpy array with image values converted to degC
print(temperature_image.GetValues("degC"))
```

```
0 100 200 300 400 500 600 700 800 900 1000 [m]
323.15 322.54 322.457 322.603 322.756 322.935 322.99 323.035 323.077 323.107
323.107 [K]
[50.         49.39017613 49.30712864 49.45309038 49.60622044 49.78503048
 49.83986208 49.88518646 49.9269508  49.95674473 49.95674473]
```

`list_global_trends` returns all the global trends available in the output.

[10]:
```
results.list_global_trends()
```

[10]:
```
[GlobalTrendMetadata(property_name='timestep')]
```

A global trend curve can be accessed with `get_global_trend_curve` passing the property name. Again, a `Curve` object is returned.

[11]:
```
timestep_trend = results.get_global_trend_curve("timestep")

print(timestep_trend.GetDomain())
print(timestep_trend.GetImage())
```

```
0 296.553 596.553 896.553 1196.55 1496.55 1796.55 2096.55 2396.55 2696.55
2996.55 3296.55 3596.55 3601.55 [s]
0.0001 5 5 5 5 5 5 5 5 5 5 5 5 [s]
```

A list of all positional trends can be accessed with `list_positiona_trends`.

[12]:
```
results.list_positional_trends()
```

[12]:
```
[PositionalTrendMetadata(property_name='elevation', element_name='Conn 1',
position=Scalar(50.0, 'm', 'length')),
 PositionalTrendMetadata(property_name='flow pattern', element_name='Conn 1',
position=Scalar(50.0, 'm', 'length')),
 PositionalTrendMetadata(property_name='gas phase volume fraction',
element_name='Conn 1', position=Scalar(50.0, 'm', 'length')),
 PositionalTrendMetadata(property_name='holdup', element_name='Conn 1',
position=Scalar(50.0, 'm', 'length')),
 PositionalTrendMetadata(property_name='liquid mass flow rate',
element_name='Conn 1', position=Scalar(50.0, 'm', 'length')),
 PositionalTrendMetadata(property_name='liquid volumetric flow rate std',
element_name='Conn 1', position=Scalar(50.0, 'm', 'length')),
 PositionalTrendMetadata(property_name='mixture temperature', element_name='Conn
```

```
1', position=Scalar(50.0, 'm', 'length')),
 PositionalTrendMetadata(property_name='oil phase volume fraction',
element_name='Conn 1', position=Scalar(50.0, 'm', 'length')),
 PositionalTrendMetadata(property_name='pressure', element_name='Conn 1',
position=Scalar(50.0, 'm', 'length')),
 PositionalTrendMetadata(property_name='total gas mass flow rate',
element_name='Conn 1', position=Scalar(50.0, 'm', 'length')),
 PositionalTrendMetadata(property_name='total gas volumetric flow rate std',
element_name='Conn 1', position=Scalar(50.0, 'm', 'length')),
 PositionalTrendMetadata(property_name='total mass flow rate',
element_name='Conn 1', position=Scalar(50.0, 'm', 'length')),
 PositionalTrendMetadata(property_name='total oil mass flow rate',
element_name='Conn 1', position=Scalar(50.0, 'm', 'length')),
 PositionalTrendMetadata(property_name='total oil volumetric flow rate std',
element_name='Conn 1', position=Scalar(50.0, 'm', 'length')),
 PositionalTrendMetadata(property_name='total water mass flow rate',
element_name='Conn 1', position=Scalar(50.0, 'm', 'length')),
 PositionalTrendMetadata(property_name='total water volumetric flow rate std',
element_name='Conn 1', position=Scalar(50.0, 'm', 'length')),
 PositionalTrendMetadata(property_name='water phase volume fraction',
element_name='Conn 1', position=Scalar(50.0, 'm', 'length'))]
```

Any positional trend curve listed above can be accessed with get_positional_trend_curve passing the property name, the name of the element in the network and the position of the trend, which is a tuple of the position (float) and its unit (string). Again, a Curve object is returned.

[13]:
```
pressure_trend_curve = results.get_positional_trend_curve("pressure", "Conn 1",␣
↪(50, "m"))

# Array with pressure curve image values and unit
print(pressure_trend_curve.GetImage())

# numpy array with pressure curve image values in psi
print(pressure_trend_curve.GetImage().GetValues("psi"))
```

```
5e+06 5.00004e+06 5.00003e+06 5.00003e+06 5.00003e+06 5.00003e+06 5.00002e+06
5.00002e+06 5.00002e+06 5.00002e+06 5.00002e+06 5.00002e+06 5.00002e+06
5.00002e+06 [Pa]
[725.18871949 725.19421797 725.19352969 725.19289423 725.19293144
 725.19240759 725.19204346 725.19162527 725.19183076 725.19172578
 725.19149069 725.19149416 725.19169055 725.1916871 ]
```

[ ]: