

---

# **PyFR Documentation**

***Release 1.13.0***

**Imperial College London**

**Feb 07, 2022**



# CONTENTS

<b>1 Installation</b>	<b>3</b>
1.1 Quick-start . . . . .	3
1.1.1 macOS . . . . .	3
1.1.2 Ubuntu . . . . .	4
1.2 Compiling from source . . . . .	4
1.2.1 Dependencies . . . . .	4
<b>2 User Guide</b>	<b>7</b>
2.1 Running PyFR . . . . .	7
2.1.1 Running in Parallel . . . . .	8
2.2 Configuration File (.ini) . . . . .	8
2.2.1 Backends . . . . .	8
2.2.2 Systems . . . . .	10
2.2.3 Boundary and Initial Conditions . . . . .	17
2.2.4 Nodal Point Sets . . . . .	20
2.2.5 Plugins . . . . .	24
2.2.6 Additional Information . . . . .	29
<b>3 Developer Guide</b>	<b>31</b>
3.1 A Brief Overview of the PyFR Framework . . . . .	31
3.1.1 Where to Start . . . . .	31
3.1.2 Controller . . . . .	31
3.1.3 Stepper . . . . .	35
3.1.4 PseudoStepper . . . . .	42
3.1.5 System . . . . .	48
3.1.6 Elements . . . . .	51
3.1.7 Interfaces . . . . .	56
3.1.8 Backend . . . . .	60
3.1.9 Pointwise Kernel Provider . . . . .	62
3.1.10 Kernel Generator . . . . .	64
3.2 PyFR-Mako . . . . .	66
3.2.1 PyFR-Mako Kernels . . . . .	66
3.2.2 PyFR-Mako Macros . . . . .	67
3.2.3 Syntax . . . . .	67
<b>4 Performance Tuning</b>	<b>69</b>
4.1 OpenMP Backend . . . . .	69
4.1.1 libxsmm . . . . .	69
4.1.2 AVX-512 . . . . .	69
4.1.3 Cores vs. threads . . . . .	69

4.1.4	MPI processes vs. OpenMP threads . . . . .	69
4.2	CUDA Backend . . . . .	70
4.2.1	CUDA-aware MPI . . . . .	70
4.3	Partitioning . . . . .	70
4.3.1	METIS vs SCOTCH . . . . .	70
4.3.2	Mixed grids . . . . .	70
4.4	Parallel I/O . . . . .	71
4.5	Start-up Time . . . . .	71
<b>5</b>	<b>Examples</b>	<b>73</b>
5.1	Euler Equations . . . . .	73
5.1.1	2D Euler Vortex . . . . .	73
5.2	Compressible Navier–Stokes Equations . . . . .	75
5.2.1	2D Couette Flow . . . . .	75
5.3	Incompressible Navier–Stokes Equations . . . . .	76
5.3.1	2D Incompressible Cylinder Flow . . . . .	76
<b>6</b>	<b>Indices and Tables</b>	<b>77</b>
<b>Index</b>		<b>79</b>

PyFR 1.13.0 is an open-source flow solver that uses the high-order flux reconstruction method. For more information on the PyFR project visit our [website](#), or to ask a question visit our [forum](#).

Contents:



## **INSTALLATION**

### **1.1 Quick-start**

PyFR 1.13.0 can be installed using [pip](#) and [virtualenv](#), as shown in the quick-start guides below.

Alternatively, PyFR 1.13.0 can be installed from source, see [Compiling from source](#).

#### **1.1.1 macOS**

We recommend using the package manager [homebrew](#). Open the terminal and install the dependencies with the following commands:

```
brew install python3 open-mpi metis  
pip3 install virtualenv
```

For visualisation of results, either install ParaView from the command line:

```
brew cask install paraview
```

or download the app from the ParaView [website](#). Then create a virtual environment and activate it:

```
virtualenv --python=python3 ENV3  
source ENV3/bin/activate
```

Finally, install PyFR with [pip](#) in the virtual environment:

```
pip install pyfr
```

This concludes the installation. In order to run PyFR with the OpenMP backend (see [Running PyFR](#)), use the following settings in the [Configuration File \(.ini\)](#):

```
[backend-openmp]  
cc = gcc-8
```

Note the version of the compiler which must support the `openmp` flag. This has been tested on macOS 11.6 for ARM and Intel CPUs.

## 1.1.2 Ubuntu

Open the terminal and install the dependencies with the following commands:

```
sudo apt install python3 python3-pip libopenmpi-dev openmpi-bin  
sudo apt install metis libmetis-dev  
pip3 install virtualenv
```

For visualisation of results, either install ParaView from the command line:

```
sudo apt install paraview
```

or download the app from the ParaView [website](#). Then create a virtual environment and activate it:

```
python3 -m virtualenv pyfr-venv  
source pyfr-venv/bin/activate
```

Finally, install PyFR with [pip](#) in the virtual environment:

```
pip install pyfr
```

This concludes the installation.

This has been tested on Ubuntu 20.04.

## 1.2 Compiling from source

PyFR can be obtained [here](#). To install the software from source, use the provided `setup.py` installer or add the root PyFR directory to `PYTHONPATH` using:

```
user@computer ~/PyFR$ export PYTHONPATH=.:${PYTHONPATH}
```

When installing from source, we strongly recommend using [pip](#) and [virtualenv](#) to manage the Python dependencies.

### 1.2.1 Dependencies

PyFR 1.13.0 has a hard dependency on Python 3.9+ and the following Python packages:

1. [gimmik](#) >= 2.2
2. [h5py](#) >= 2.10
3. [mako](#) >= 1.0.0
4. [mpi4py](#) >= 3.0
5. [numpy](#) >= 1.20
6. [platformdirs](#) >= 2.2.0
7. [pytools](#) >= 2016.2.1

Note that due to a bug in NumPy, PyFR is not compatible with 32-bit Python distributions.

### 1.2.1.1 CUDA Backend

The CUDA backend targets NVIDIA GPUs with a compute capability of 3.0 or greater. The backend requires:

1. `CUDA >= 8.0`

### 1.2.1.2 HIP Backend

The HIP backend targets AMD GPUs which are supported by the ROCm stack. The backend requires:

1. `ROCM >= 4.5.0`
2. `rocBLAS >= 2.41.0`

### 1.2.1.3 OpenCL Backend

The OpenCL backend targets a range of accelerators including GPUs from AMD, Intel, and NVIDIA. The backend requires:

1. `OpenCL`
2. `CLBlast`

Note that when running on NVIDIA GPUs the OpenCL backend terminate with a segmentation fault after the simulation has finished. This is due to a long-standing bug in how the NVIDIA OpenCL implementation handles sub-buffers. As it occurs during the termination phase—after all data has been written out to disk—the issue does *not* impact the functionality or correctness of PyFR.

### 1.2.1.4 OpenMP Backend

The OpenMP backend targets multi-core CPUs. The backend requires:

1. `GCC >= 4.9` or another C compiler with OpenMP support
2. Optionally `libxsmm >= commit 14b6cea61376653b2712e3eefa72b13c5e76e421` compiled as a shared library (`STATIC=0`) with `BLAS=0` and `CODE_BUFSIZE=262144`

In order for PyFR to find libxsmm it must be located in a directory which is on the library search path. Alternatively, the path can be specified explicitly by exporting the environment variable `PYFR_XSMM_LIBRARY_PATH=/path/to/libxsmm.so`.

### 1.2.1.5 Parallel

To partition meshes for running in parallel it is also necessary to have one of the following partitioners installed:

1. `METIS >= 5.0`
2. `SCOTCH >= 6.0`

In order for PyFR to find these libraries they must be located in a directory which is on the library search path. Alternatively, the paths can be specified explicitly by exporting the environment variables `PYFR_METIS_LIBRARY_PATH=/path/to/libmetis.so` and/or `PYFR_SCOTCH_LIBRARY_PATH=/path/to/libscotch.so`.



**USER GUIDE**

For information on how to install PyFR see [Installation](#).

## 2.1 Running PyFR

PyFR 1.13.0 uses three distinct file formats:

1. `.ini` — configuration file
2. `.pyfrm` — mesh file
3. `.pyfrs` — solution file

The following commands are available from the `pyfr` program:

1. `pyfr import` — convert a `Gmsh` `.msh` file into a PyFR `.pyfrm` file.

Example:

```
pyfr import mesh.msh mesh.pyfrm
```

2. `pyfr partition` — partition an existing mesh and associated solution files.

Example:

```
pyfr partition 2 mesh.pyfrm solution.pyfrs .
```

3. `pyfr run` — start a new PyFR simulation. Example:

```
pyfr run mesh.pyfrm configuration.ini
```

4. `pyfr restart` — restart a PyFR simulation from an existing solution file. Example:

```
pyfr restart mesh.pyfrm solution.pyfrs
```

5. `pyfr export` — convert a PyFR `.pyfrs` file into an unstructured VTK `.vtu` or `.pvtu` file. If a `-k` flag is provided with an integer argument then `.pyfrs` elements are converted to high-order VTK cells which are exported, where the order of the VTK cells is equal to the value of the integer argument. Example:

```
pyfr export -k 4 mesh.pyfrm solution.pyfrs solution.vtu
```

If a `-d` flag is provided with an integer argument then `.pyfrs` elements are subdivided into linear VTK cells which are exported, where the number of sub-divisions is equal to the value of the integer argument. Example:

```
pyfr export -d 4 mesh.pyfrm solution.pyfrs solution.vtu
```

If no flags are provided then .pyfrs elements are converted to high-order VTK cells which are exported, where the order of the cells is equal to the order of the solution data in the .pyfrs file.

## 2.1.1 Running in Parallel

PyFR can be run in parallel. To do so prefix `pyfr` with `mpiexec -n <cores/devices>`. Note that the mesh must be pre-partitioned, and the number of cores or devices must be equal to the number of partitions.

## 2.2 Configuration File (.ini)

The .ini configuration file parameterises the simulation. It is written in the [INI](#) format. Parameters are grouped into sections. The roles of each section and their associated parameters are described below. Note that both ; and # may be used as comment characters.

### 2.2.1 Backends

The backend sections detail how the solver will be configured for a range of different hardware platforms. If a hardware specific backend section is omitted, then PyFR will fall back to built-in default settings.

#### 2.2.1.1 [backend]

Parameterises the backend with

1. `precision` — number precision:

`single | double`

2. `rank-allocator` — MPI rank allocator:

`linear | random`

Example:

```
[backend]
precision = double
rank-allocator = linear
```

#### 2.2.1.2 [backend-cuda]

Parameterises the CUDA backend with

1. `device-id` — method for selecting which device(s) to run on:

`int | round-robin | local-rank`

2. `mpi-type` — type of MPI library that is being used:

`standard | cuda-aware`

3. `cflags` — additional NVIDIA realtime compiler (nvrtc) flags:

`string`

Example:

```
[backend-cuda]
device-id = round-robin
mpi-type = standard
```

### 2.2.1.3 [backend-hip]

Parameterises the HIP backend with

1. `device-id` — method for selecting which device(s) to run on:

*int | local-rank*

2. `mpi-type` — type of MPI library that is being used:

*standard | hip-aware*

Example:

```
[backend-hip]
device-id = local-rank
mpi-type = standard
```

### 2.2.1.4 [backend-opencl]

Parameterises the OpenCL backend with

1. `platform-id` — for selecting platform id:

*int | string*

2. `device-type` — for selecting what type of device(s) to run on:

*all | cpu | gpu | accelerator*

3. `device-id` — for selecting which device(s) to run on:

*int | string | local-rank*

4. `gimmik-max-nnz` — cutoff for GiMMiK in terms of the number of non-zero entires in a constant matrix:

*int*

Example:

```
[backend-opencl]
platform-id = 0
device-type = gpu
device-id = local-rank
gimmik-max-nnz = 512
```

### 2.2.1.5 [backend-openmp]

Parameterises the OpenMP backend with

1. cc — C compiler:

*string*

2. cflags — additional C compiler flags:

*string*

3. alignb — alignment requirement in bytes; must be a power of two and at least 32:

*int*

4. gmmik-max-nnz — cutoff for GiMMiK in terms of the number of non-zero entries in a constant matrix:

*int*

Example:

```
[backend-openmp]
cc = gcc
```

## 2.2.2 Systems

These sections of the input file setup and control the physical system being solved, as well as characteristics of the spatial and temporal schemes to be used.

### 2.2.2.1 [constants]

Sets constants used in the simulation

1. gamma — ratio of specific heats for euler | navier-stokes:

*float*

2. mu — dynamic viscosity for navier-stokes:

*float*

3. nu — kinematic viscosity for ac-navier-stokes:

*float*

4. Pr — Prandtl number for navier-stokes:

*float*

5. cpTref — product of specific heat at constant pressure and reference temperature for navier-stokes with Sutherland's Law:

*float*

6. cpTs — product of specific heat at constant pressure and Sutherland temperature for navier-stokes with Sutherland's Law:

*float*

7. ac-zeta — artificial compressibility factor for ac-euler | ac-navier-stokes

*float*

Other constant may be set by the user which can then be used throughout the .ini file.

Example:

```
[constants]
; PyFR Constants
gamma = 1.4
mu = 0.001
Pr = 0.72

; User Defined Constants
V_in = 1.0
P_out = 20.0
```

### 2.2.2.2 [solver]

Parameterises the solver with

1. system — governing system:

euler | navier-stokes | ac-euler | ac-navier-stokes

where

navier-stokes requires

- viscosity-correction — viscosity correction:  
none | sutherland
- shock-capturing — shock capturing scheme:  
none | artificial-viscosity

2. order — order of polynomial solution basis:

*int*

3. anti-alias — type of anti-aliasing:

flux | surf-flux | flux, surf-flux

Example:

```
[solver]
system = navier-stokes
order = 3
anti-alias = flux
viscosity-correction = none
shock-capturing = artificial-viscosity
```

### 2.2.2.3 [solver-time-integrator]

Parameterises the time-integration scheme used by the solver with

1. **formulation** — formulation:

`std | dual`

where

`std` requires

- **scheme** — time-integration scheme
  - `euler | rk34 | rk4 | rk45 | tvd-rk3`
- **tstart** — initial time
  - float*
- **tend** — final time
  - float*
- **dt** — time-step
  - float*
- **controller** — time-step controller
  - `none | pi`
  - where
    - `pi` only works with `rk34` and `rk45` and requires
      - **atol** — absolute error tolerance
        - float*
      - **rtol** — relative error tolerance
        - float*
      - **errest-norm** — norm to use for estimating the error
        - `uniform | 12`
      - **safety-fact** — safety factor for step size adjustment (suitable range 0.80-0.95)
        - float*
      - **min-fact** — minimum factor by which the time-step can change between iterations (suitable range 0.1-0.5)
        - float*
      - **max-fact** — maximum factor by which the time-step can change between iterations (suitable range 2.0-6.0)
        - float*
      - **dt-max** — maximum permissible time-step
        - float*

`dual` requires

- **scheme** — time-integration scheme

`backward-euler | bdf2 | bdf3 | sdirk33 | sdirk43`

- `pseudo-scheme` — pseudo time-integration scheme  
`euler | rk34 | rk4 | rk45 | tvd-rk3 | vermeire`
- `tstart` — initial time  
`float`
- `tend` — final time  
`float`
- `dt` — time-step  
`float`
- `pseudo-dt` — pseudo time-step  
`float`
- `controller` — pseudo time-step controller  
`none`
- `pseudo-niters-max` — minimum number of iterations  
`int`
- `pseudo-niters-min` — maximum number of iterations  
`int`
- `pseudo-resid-tol` — pseudo residual tolerance  
`float`
- `pseudo-resid-norm` — pseudo residual norm  
`uniform | l2`
- `pseudo-controller` — pseudo time-step controller  
`none | local-pi`  
where  
`local-pi` only works with `rk34` and `rk45` and requires
  - `atol` — absolute error tolerance  
`float`
  - `safety-fact` — safety factor for pseudo time-step size adjustment (suitable range 0.80-0.95)  
`float`
  - `min-fact` — minimum factor by which the local pseudo time-step can change between iterations (suitable range 0.98-0.998)  
`float`
  - `max-fact` — maximum factor by which the local pseudo time-step can change between iterations (suitable range 1.001-1.01)  
`float`

- **pseudo-dt-max-mult** — maximum permissible local pseudo time-step given as a multiplier of **pseudo-dt** (suitable range 2.0-5.0)

*float*

Example:

```
[solver-time-integrator]
formulation = std
scheme = rk45
controller = pi
tstart = 0.0
tend = 10.0
dt = 0.001
atol = 0.00001
rtol = 0.00001
errest-norm = 12
safety-fact = 0.9
min-fact = 0.3
max-fact = 2.5
```

#### 2.2.2.4 [solver-dual-time-integrator-multip]

Parameterises multi-p for dual time-stepping with

1. **pseudo-dt-fact** — factor by which the pseudo time-step size changes between multi-p levels:

*float*

2. **cycle** — nature of a single multi-p cycle:

```
[(order,nsteps), (order,nsteps), ... (order,nsteps)]
```

where **order** in the first and last bracketed pair must be the overall polynomial order used for the simulation, and **order** can only change by one between subsequent bracketed pairs

Example:

```
[solver-dual-time-integrator-multip]
pseudo-dt-fact = 2.3
cycle = [(3, 1), (2, 1), (1, 1), (0, 2), (1, 1), (2, 1), (3, 3)]
```

#### 2.2.2.5 [solver-interfaces]

Parameterises the interfaces with

1. **riemann-solver** — type of Riemann solver:

**rusanov** | **hll** | **hllc** | **roe** | **roem**

where

**hll** | **hllc** | **roe** | **roem** do not work with **ac-euler** | **ac-navier-stokes**

2. **ldg-beta** — beta parameter used for LDG:

*float*

3. **ldg-tau** — tau parameter used for LDG:

*float*

Example:

```
[solver-interfaces]
riemann-solver = rusanov
ldg-beta = 0.5
ldg-tau = 0.1
```

### 2.2.2.6 [solver-source-terms]

Parameterises solution, space (x, y, [z]), and time (t) dependent source terms with

1. **rho** — density source term for euler | navier-stokes:

*string*

2. **rhou** — x-momentum source term for euler | navier-stokes :

*string*

3. **rhov** — y-momentum source term for euler | navier-stokes :

*string*

4. **rhow** — z-momentum source term for euler | navier-stokes :

*string*

5. **E** — energy source term for euler | navier-stokes :

*string*

6. **p** — pressure source term for ac-euler | ac-navier-stokes:

*string*

7. **u** — x-velocity source term for ac-euler | ac-navier-stokes:

*string*

8. **v** — y-velocity source term for ac-euler | ac-navier-stokes:

*string*

9. **w** — w-velocity source term for ac-euler | ac-navier-stokes:

*string*

Example:

```
[solver-source-terms]
rho = t
rhou = x*y*sin(y)
rhov = z*rho
rhow = 1.0
E = 1.0/(1.0+x)
```

### 2.2.2.7 [solver-artificial-viscosity]

Parameterises artificial viscosity for shock capturing with

1. **max-artvisc** — maximum artificial viscosity:

*float*

2. **s0** — sensor cut-off:

*float*

3. **kappa** — sensor range:

*float*

Example:

```
[solver-artificial-viscosity]
max-artvisc = 0.01
s0 = 0.01
kappa = 5.0
```

### 2.2.2.8 [soln-filter]

Parameterises an exponential solution filter with

1. **nsteps** — apply filter every **nsteps**:

*int*

2. **alpha** — strength of filter:

*float*

3. **order** — order of filter:

*int*

4. **cutoff** — cutoff frequency below which no filtering is applied:

*int*

Example:

```
[soln-filter]
nsteps = 10
alpha = 36.0
order = 16
cutoff = 1
```

## 2.2.3 Boundary and Initial Conditions

These sections allow users to set the boundary and initial conditions of calculations.

### 2.2.3.1 [soln-bcs-name]

Parameterises constant, or if available space (x, y, [z]) and time (t) dependent, boundary condition labelled *name* in the .pyfrm file with

1. type — type of boundary condition:

```
ac-char-riem-inv | ac-in-fv | ac-out-fp | char-riem-inv | no-slp-adia-wall
| no-slp-isot-wall | no-slp-wall | slp-adia-wall | slp-wall | sub-in-frv |
sub-in-ftptang | sub-out-fp | sup-in-fa | sup-out-fn
```

where

ac-char-riem-inv only works with ac-euler | ac-navier-stokes and requires

- ac-zeta — artificial compressibility factor for boundary (increasing ac-zeta makes the boundary less reflective allowing larger deviation from the target state)

float

- niters — number of Newton iterations

int

- p — pressure

float | string

- u — x-velocity

float | string

- v — y-velocity

float | string

- w — z-velocity

float | string

ac-in-fv only works with ac-euler | ac-navier-stokes and requires

- u — x-velocity

float | string

- v — y-velocity

float | string

- w — z-velocity

float | string

ac-out-fp only works with ac-euler | ac-navier-stokes and requires

- p — pressure

float | string

char-riem-inv only works with euler | navier-stokes and requires

- rho — density

- *u* — x-velocity
  - float | string*
- *v* — y-velocity
  - float | string*
- *w* — z-velocity
  - float | string*
- *p* — static pressure
  - float | string*

`no-slp-adia-wall` only works with `navier-stokes`

`no-slp-isot-wall` only works with `navier-stokes` and requires

- *u* — x-velocity of wall
  - float*
- *v* — y-velocity of wall
  - float*
- *w* — z-velocity of wall
  - float*
- *cpTw* — product of specific heat capacity at constant pressure and temperature of wall
  - float*

`no-slp-wall` only works with `ac-navier-stokes` and requires

- *u* — x-velocity of wall
  - float*
- *v* — y-velocity of wall
  - float*
- *w* — z-velocity of wall
  - float*

`slp-adia-wall` only works with `euler | navier-stokes`

`slp-wall` only works with `ac-euler | ac-navier-stokes`

`sub-in-frv` only works with `navier-stokes` and requires

- *rho* — density
  - float | string*
- *u* — x-velocity
  - float | string*
- *v* — y-velocity
  - float | string*
- *w* — z-velocity
  - float | string*

*float | string*

sub-in-ftpttang only works with navier-stokes and requires

- pt — total pressure  
*float*
- cpTt — product of specific heat capacity at constant pressure and total temperature  
*float*
- theta — azimuth angle (in degrees) of inflow measured in the x-y plane relative to the positive x-axis  
*float*
- phi — inclination angle (in degrees) of inflow measured relative to the positive z-axis  
*float*

sub-out-fp only works with navier-stokes and requires

- p — static pressure  
*float | string*

sup-in-fa only works with euler | navier-stokes and requires

- rho — density  
*float | string*
- u — x-velocity  
*float | string*
- v — y-velocity  
*float | string*
- w — z-velocity  
*float | string*
- p — static pressure  
*float | string*

sup-out-fn only works with euler | navier-stokes

Example:

```
[soln-bcs-bcwallupper]
type = no-slp-isot-wall
cpTw = 10.0
u = 1.0
```

Simple periodic boundary conditions are supported; however, their behaviour is not controlled through the .ini file, instead it is handled at the mesh generation stage. Two faces may be tagged with `periodic_x_l` and `periodic_x_r`, where `x` is a unique identifier for the pair of boundaries. Currently, only periodicity in a single cardinal direction is supported, for example, the planes `(x,y,0)` and `(x,y,10)`.

### 2.2.3.2 [soln-ics]

Parameterises space (x, y, [z]) dependent initial conditions with

1. rho — initial density distribution for euler | navier-stokes:

*string*

2. u — initial x-velocity distribution for euler | navier-stokes | ac-euler | ac-navier-stokes:

*string*

3. v — initial y-velocity distribution for euler | navier-stokes | ac-euler | ac-navier-stokes:

*string*

4. w — initial z-velocity distribution for euler | navier-stokes | ac-euler | ac-navier-stokes:

*string*

5. p — initial static pressure distribution for euler | navier-stokes | ac-euler | ac-navier-stokes:

*string*

Example:

```
[soln-ics]
rho = 1.0
u = x*y*sin(y)
v = z
w = 1.0
p = 1.0/(1.0+x)
```

## 2.2.4 Nodal Point Sets

Solution point sets must be specified for each element type that is used and flux point sets must be specified for each interface type that is used. If anti-aliasing is enabled then quadrature point sets for each element and interface type that is used must also be specified. For example, a 3D mesh comprised only of prisms requires a solution point set for prism elements and flux point set for quadrilateral and triangular interfaces.

### 2.2.4.1 [solver-interfaces-line{-mg-porder}]

Parameterises the line interfaces, or if -mg-porder is suffixed the line interfaces at multi-p level *order*, with

1. flux-pts — location of the flux points on a line interface:

gauss-legendre | gauss-legendre-lobatto

2. quad-deg — degree of quadrature rule for anti-aliasing on a line interface:

*int*

3. quad-pts — name of quadrature rule for anti-aliasing on a line interface:

gauss-legendre | gauss-legendre-lobatto

Example:

```
[solver-interfaces-line]
flux-pts = gauss-legendre
quad-deg = 10
quad-pts = gauss-legendre
```

#### 2.2.4.2 [solver-interfaces-tri{-mg-porder}]

Parameterises the triangular interfaces, or if -mg-porder is suffixed the triangular interfaces at multi-p level *order*, with

1. flux-pts — location of the flux points on a triangular interface:

williams-shunn

2. quad-deg — degree of quadrature rule for anti-aliasing on a triangular interface:

*int*

3. quad-pts — name of quadrature rule for anti-aliasing on a triangular interface:

williams-shunn | witherden-vincent

Example:

```
[solver-interfaces-tri]
flux-pts = williams-shunn
quad-deg = 10
quad-pts = williams-shunn
```

#### 2.2.4.3 [solver-interfaces-quad{-mg-porder}]

Parameterises the quadrilateral interfaces, or if -mg-porder is suffixed the quadrilateral interfaces at multi-p level *order*, with

1. flux-pts — location of the flux points on a quadrilateral interface:

gauss-legendre | gauss-legendre-lobatto

2. quad-deg — degree of quadrature rule for anti-aliasing on a quadrilateral interface:

*int*

3. quad-pts — name of quadrature rule for anti-aliasing on a quadrilateral interface:

gauss-legendre | gauss-legendre-lobatto | witherden-vincent

Example:

```
[solver-interfaces-quad]
flux-pts = gauss-legendre
quad-deg = 10
quad-pts = gauss-legendre
```

#### 2.2.4.4 [solver-elements-tri{-mg-porder}]

Parameterises the triangular elements, or if -mg-porder is suffixed the triangular elements at multi-p level *order*, with

1. **soln-pts** — location of the solution points in a triangular element:

williams-shunn

2. **quad-deg** — degree of quadrature rule for anti-aliasing in a triangular element:

*int*

3. **quad-pts** — name of quadrature rule for anti-aliasing in a triangular element:

williams-shunn | witherden-vincent

Example:

```
[solver-elements-tri]
soln-pts = williams-shunn
quad-deg = 10
quad-pts = williams-shunn
```

#### 2.2.4.5 [solver-elements-quad{-mg-porder}]

Parameterises the quadrilateral elements, or if -mg-porder is suffixed the quadrilateral elements at multi-p level *order*, with

1. **soln-pts** — location of the solution points in a quadrilateral element:

gauss-legendre | gauss-legendre-lobatto

2. **quad-deg** — degree of quadrature rule for anti-aliasing in a quadrilateral element:

*int*

3. **quad-pts** — name of quadrature rule for anti-aliasing in a quadrilateral element:

gauss-legendre | gauss-legendre-lobatto | witherden-vincent

Example:

```
[solver-elements-quad]
soln-pts = gauss-legendre
quad-deg = 10
quad-pts = gauss-legendre
```

#### 2.2.4.6 [solver-elements-hex{-mg-porder}]

Parameterises the hexahedral elements, or if -mg-porder is suffixed the hexahedral elements at multi-p level *order*, with

1. **soln-pts** — location of the solution points in a hexahedral element:

gauss-legendre | gauss-legendre-lobatto

2. **quad-deg** — degree of quadrature rule for anti-aliasing in a hexahedral element:

*int*

3. **quad-pts** — name of quadrature rule for anti-aliasing in a hexahedral element:

gauss-legendre | gauss-legendre-lobatto | witherden-vincent

Example:

```
[solver-elements-hex]
soln-pts = gauss-legendre
quad-deg = 10
quad-pts = gauss-legendre
```

#### 2.2.4.7 [solver-elements-tet{-mg-porder}]

Parameterises the tetrahedral elements, or if -mg-porder is suffixed the tetrahedral elements at multi-p level *order*, with

1. soln-pts — location of the solution points in a tetrahedral element:

shunn-ham

2. quad-deg — degree of quadrature rule for anti-aliasing in a tetrahedral element:

*int*

3. quad-pts — name of quadrature rule for anti-aliasing in a tetrahedral element:

shunn-ham | witherden-vincent

Example:

```
[solver-elements-tet]
soln-pts = shunn-ham
quad-deg = 10
quad-pts = shunn-ham
```

#### 2.2.4.8 [solver-elements-pri{-mg-porder}]

Parameterises the prismatic elements, or if -mg-porder is suffixed the prismatic elements at multi-p level *order*, with

1. soln-pts — location of the solution points in a prismatic element:

williams-shunn~gauss-legendre | williams-shunn~gauss-legendre-lobatto

2. quad-deg — degree of quadrature rule for anti-aliasing in a prismatic element:

*int*

3. quad-pts — name of quadrature rule for anti-aliasing in a prismatic element:

williams-shunn~gauss-legendre | williams-shunn~gauss-legendre-lobatto |  
witherden-vincent

Example:

```
[solver-elements-pri]
soln-pts = williams-shunn~gauss-legendre
quad-deg = 10
quad-pts = williams-shunn~gauss-legendre
```

### 2.2.4.9 [solver-elements-pyr{-mg-*porder*}]

Parameterises the pyramidal elements, or if -mg-*porder* is suffixed the pyramidal elements at multi-p level *order*, with

1. **soln-pts** — location of the solution points in a pyramidal element:  
*gauss-legendre | gauss-legendre-lobatto*
2. **quad-deg** — degree of quadrature rule for anti-aliasing in a pyramidal element:  
*int*
3. **quad-pts** — name of quadrature rule for anti-aliasing in a pyramidal element:  
*witherden-vincent*

Example:

```
[solver-elements-pyr]
soln-pts = gauss-legendre
quad-deg = 10
quad-pts = witherden-vincent
```

## 2.2.5 Plugins

Plugins allow for powerful additional functionality to be swapped in and out. It is possible to load multiple instances of the same plugin by appending a tag, for example:

```
[soln-plugin-writer]
...
[soln-plugin-writer-2]
...
[soln-plugin-writer-three]
...
```

### 2.2.5.1 [soln-plugin-writer]

Periodically write the solution to disk in the pyfrs format. Parameterised with

1. **dt-out** — write to disk every dt-out time units:  
*float*
2. **basedir** — relative path to directory where outputs will be written:  
*string*
3. **basename** — pattern of output names:  
*string*
4. **post-action** — command to execute after writing the file:  
*string*
5. **post-action-mode** — how the post-action command should be executed:  
*blocking | non-blocking*

4. **region** — region to be written, specified as either the entire domain using `*`, a cuboidal sub-region via diametrically opposite vertices, or a sub-region of elements that have faces on a specific domain boundary via the name of the domain boundary

`* | [(x, y, [z]), (x, y, [z])] | string`

Example:

```
[soln-plugin-writer]
dt-out = 0.01
basedir =
basename = files-{t:.2f}
post-action = echo "Wrote file {soln} at time {t} for mesh {mesh}."
post-action-mode = blocking
region = [(-5, -5, -5), (5, 5, 5)]
```

### 2.2.5.2 [soln-plugin-fluidforce-name]

Periodically integrates the pressure and viscous stress on the boundary labelled `name` and writes out the resulting force and moment (if requested) vectors to a CSV file. Parameterised with

1. **nsteps** — integrate every `nsteps`:

`int`

2. **file** — output file path; should the file already exist it will be appended to:

`string`

3. **header** — if to output a header row or not:

`boolean`

4. **morigin** — origin used to compute moments (optional):

`(x, y, [z])`

Example:

```
[soln-plugin-fluidforce-wing]
nsteps = 10
file = wing-forces.csv
header = true
morigin = (0.0, 0.0, 0.5)
```

### 2.2.5.3 [soln-plugin-nancheck]

Periodically checks the solution for NaN values. Parameterised with

1. **nsteps** — check every `nsteps`:

`int`

Example:

```
[soln-plugin-nancheck]
nsteps = 10
```

#### 2.2.5.4 [soln-plugin-residual]

Periodically calculates the residual and writes it out to a CSV file. Parameterised with

1. **nsteps** — calculate every **nsteps**:

*int*

2. **file** — output file path; should the file already exist it will be appended to:

*string*

3. **header** — if to output a header row or not:

*boolean*

Example:

```
[soln-plugin-residual]
nsteps = 10
file = residual.csv
header = true
```

#### 2.2.5.5 [soln-plugin-dtstats]

Write time-step statistics out to a CSV file. Parameterised with

1. **flushsteps** — flush to disk every **flushsteps**:

*int*

2. **file** — output file path; should the file already exist it will be appended to:

*string*

3. **header** — if to output a header row or not:

*boolean*

Example:

```
[soln-plugin-dtstats]
flushsteps = 100
file = dtstats.csv
header = true
```

#### 2.2.5.6 [soln-plugin-pseudostats]

Write pseudo-step convergence history out to a CSV file. Parameterised with

1. **flushsteps** — flush to disk every **flushsteps**:

*int*

2. **file** — output file path; should the file already exist it will be appended to:

*string*

3. **header** — if to output a header row or not:

*boolean*

Example:

```
[soln-plugin-pseudostats]
flushsteps = 100
file = pseudostats.csv
header = true
```

### 2.2.5.7 [soln-plugin-sampler]

Periodically samples specific points in the volume and writes them out to a CSV file. The point location process automatically takes advantage of `scipy.spatial.cKDTree` where available. Parameterised with

1. `nsteps` — sample every `nsteps`:

*int*

2. `samp-pts` — list of points to sample:

`[(x, y), (x, y), ...] | [(x, y, z), (x, y, z), ...]`

3. `format` — output variable format:

`primitive | conservative`

4. `file` — output file path; should the file already exist it will be appended to:

*string*

5. `header` — if to output a header row or not:

*boolean*

Example:

```
[soln-plugin-sampler]
nsteps = 10
samp-pts = [(1.0, 0.7, 0.0), (1.0, 0.8, 0.0)]
format = primitive
file = point-data.csv
header = true
```

### 2.2.5.8 [soln-plugin-tavg]

Time average quantities. Parameterised with

1. `nsteps` — accumulate the average every `nsteps` time steps:

*int*

2. `dt-out` — write to disk every `dt-out` time units:

*float*

3. `tstart` — time at which to start accumulating average data:

*float*

4. `mode` — output file accumulation mode:

continuous | windowed

Windowed outputs averages over each dt-out period. Whereas, continuous outputs averages over all dt-out periods thus far completed within a given invocation of PyFR. The default is windowed.

5. **basedir** — relative path to directory where outputs will be written:

*string*

6. **basename** — pattern of output names:

*string*

7. **precision** — output file number precision:

single | double

8. **region** — region to be averaged, specified as either the entire domain using \*, a cuboidal sub-region via diametrically opposite vertices, or a sub-region of elements that have faces on a specific domain boundary via the name of the domain boundary

\* | [(x, y, [z]), (x, y, [z])] | *string*

9. **avg-name** — expression to time average, written as a function of the primitive variables and gradients thereof; multiple expressions, each with their own *name*, may be specified:

*string*

10. **fun-avg-name** — expression to compute at file output time, written as a function of any ordinary average terms; multiple expressions, each with their own *name*, may be specified:

*string*

As fun-avg terms are evaluated at write time, these are only indirectly effected by the averaging mode.

Example:

```
[soln-plugin-tavg]
nsteps = 10
dt-out = 2.0
mode = windowed
basedir = .
basename = files-{t:06.2f}

avg-u = u
avg-v = v
avg-uu = u*u
avg-vv = v*v
avg-uv = u*v

fun-avg-upup = uu - u*u
fun-avg-vpvp = vv - v*v
fun-avg-upvp = uv - u*v
fun-avg-urms = sqrt(uu - u*u + vv - v*v)
```

### 2.2.5.9 [soln-plugin-integrate]

Integrate quantities over the computational domain. Parameterised with:

1. `nsteps` — calculate the integral every `nsteps` time steps:

*int*

2. `file` — output file path; should the file already exist it will be appended to:

*string*

3. `header` — if to output a header row or not:

*boolean*

4. `quad-deg` — degree of quadrature rule (optional):

5. `quad-pts-{etype}` — name of quadrature rule (optional):

6. `int-name` — expression to integrate, written as a function of the primitive variables and gradients thereof, the physical coordinates [x, y, [z]] and/or the physical time [t]; multiple expressions, each with their own *name*, may be specified:

*string*

Example:

```
[soln-plugin-integrate]
nsteps = 50
file = integral.csv
header = true
quad-deg = 9
vor1 = (grad_w_y - grad_v_z)
vor2 = (grad_u_z - grad_w_x)
vor3 = (grad_v_x - grad_u_y)

int-E = rho*(u*u + v*v + w*w)
int-enst = rho*(%(vor1)s*%(vor1)s + %(vor2)s*%(vor2)s + %(vor3)s*%(vor3)s)
```

## 2.2.6 Additional Information

The *INI* file format is very versatile. A feature that can be useful in defining initial conditions is the substitution feature and this is demonstrated in the [\[soln-plugin-integrate\]](#) example.

To prevent situations where you have solutions files for unknown configurations, the contents of the `.ini` file are added as an attribute to `.pyfrs` files. These files use the HDF5 format and can be straightforwardly probed with tools such as `h5dump`.

In several places within the `.ini` file expressions may be used. As well as the constant `pi`, expressions containing the following functions are supported:

1. `+`, `-`, `*`, `/` — basic arithmetic
2. `sin`, `cos`, `tan` — basic trigonometric functions (radians)
3. `asin`, `acos`, `atan`, `atan2` — inverse trigonometric functions
4. `exp`, `log` — exponential and the natural logarithm
5. `tanh` — hyperbolic tangent

6. `pow` — power, note `**` is not supported
7. `sqrt` — square root
8. `abs` — absolute value
9. `min`, `max` — two variable minimum and maximum functions, arguments can be arrays

## DEVELOPER GUIDE

### 3.1 A Brief Overview of the PyFR Framework

#### 3.1.1 Where to Start

The symbolic link `pyfr.scripts.pyfr` points to the script `pyfr.scripts.main`, which is where it all starts! Specifically, the function `process_run` calls the function `_process_common`, which in turn calls the function `get_solver`, returning an Integrator – a composite of a *Controller* and a *Stepper*. The Integrator has a method named `run`, which is then called to run the simulation.

#### 3.1.2 Controller

A *Controller* acts to advance the simulation in time. Specifically, a *Controller* has a method named `advance_to` which advances a *System* to a specified time. There are three types of physical-time *Controller* available in PyFR 1.13.0:

`StdNoneController` [Click to show](#)

```
class pyfr.integrators.std.controllers.StdNoneController(*args, **kwargs):
    _accept_step(dt, idxcurr, err=None)
    _add(*args, subdims=None)
    _addv(consts, regidxs, subdims=None)
    _check_abort()
    _get_axnpby_kerns(*rs, subdims=None)
    _get_gndofs()
    _get_plugins(initsoln)
    _get_reduction_kerns(*rs, **kwargs)
    _reject_step(dt, idxold, err=None)
    advance_to(t)
    call_plugin_dt(dt)
    property cfgmeta
    collect_stats(stats)
    controller_name = 'none'
    property controller_needs_erreest
```

```
formulation = 'std'

static get_plugin_data_prefix(name, suffix)

property grad_soln

property nsteps

run()

property soln

step(t, dt)
```

*StdPIController* Click to show

```
class pyfr.integrators.std.controllers.StdPIController(*args, **kwargs)
    _accept_step(dt, idxcurr, err=None)

    _add(*args, subdims=None)

    _addv(consts, regidxs, subdims=None)

    _check_abort()

    _errest(rcurr, rprev, rerr)

    _get_axnpby_kerns(*rs, subdims=None)

    _get_gndofs()

    _get_plugins(initsoln)

    _get_reduction_kerns(*rs, **kwargs)

    _reject_step(dt, idxold, err=None)

    advance_to(t)

    call_plugin_dt(dt)

    property cfgmeta

    collect_stats(stats)

    controller_name = 'pi'

    property controller_needs_errest

    formulation = 'std'

    static get_plugin_data_prefix(name, suffix)

    property grad_soln

    property nsteps

    run()

    property soln

    step(t, dt)
```

*DualNoneController* Click to show

```
class pyfr.integrators.dual.phys.controllers.DualNoneController(*args, **kwargs)
    _accept_step(idxcurr)

    _check_abort()

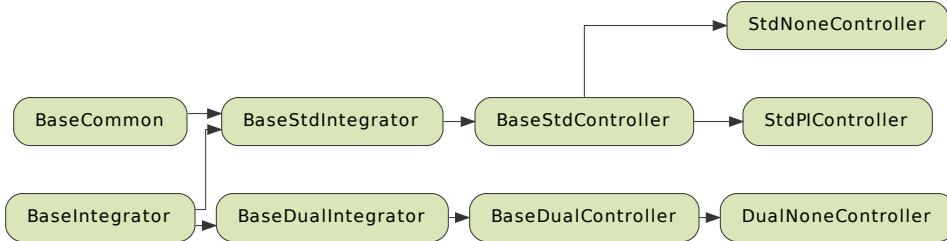
    _get_plugins(initsoln)
```

```

advance_to(t)
call_plugin_dt(dt)
property cfgmeta
collect_stats(stats)
controller_name = 'none'
formulation = 'dual'
static get_plugin_data_prefix(name, suffix)
property grad_soln
property nsteps
property pseudostepinfo
run()
property soln
step(t, dt)
property system

```

Types of physical-time *Controller* are related via the following inheritance diagram:



There are two types of pseudo-time *Controller* available in PyFR 1.13.0:

*DualNonePseudoController* [Click to show](#)

```

class pyfr.integrators.dual.pseudo.pseudocontrollers.DualNonePseudoController(*args,
**kwargs)
    _add(*args, subdims=None)
    _addv(consts, regidxs, subdims=None)

```

```
_get_axnpby_kerns(*rs, subdims=None)
_get_gndofs()
_get_reduction_kerns(*rs, **kwargs)
property _pseudo stepper regidx
_resid(rcurr, rold, dt_fac)
property _source regidx
property _stage regidx
property _stepper regidx
_update_pseudostepinfo(niters, resid)
aux_nregs = 0
convmon(i, minniters, dt_fac=1)
discard_oldest_source()
formulation = 'dual'
init_stage(currstg, stepper_coeffs)
obtain_solution(bcoeffs)
pseudo_advance(tcurr)
pseudo_controller_name = 'none'
pseudo_controller_needs_lerrest = False
store_current_soln()
```

*DualPIPpseudoController* [Click to show](#)

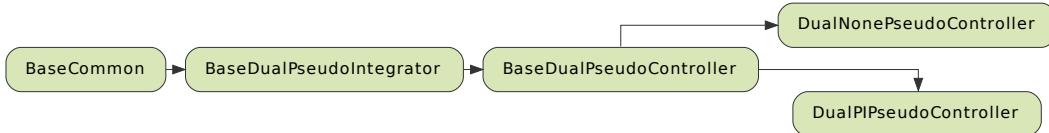
```
class pyfr.integrators.dual.pseudo.pseudocontrollers.DualPIPpseudoController(*args,
                                                               **kwargs)
    _add(*args, subdims=None)
    _addv(consts, regidxs, subdims=None)
    _get_axnpby_kerns(*rs, subdims=None)
    _get_gndofs()
    _get_reduction_kerns(*rs, **kwargs)
    property _pseudo stepper regidx
    _resid(rcurr, rold, dt_fac)
    property _source regidx
    property _stage regidx
    property _stepper regidx
    _update_pseudostepinfo(niters, resid)
    aux_nregs = 0
    convmon(i, minniters, dt_fac=1)
    discard_oldest_source()
    formulation = 'dual'
```

```

init_stage(currstg, stepper_coeff)
localerrest(errbank)
obtain_solution(bcoeffs)
pseudo_advance(tcurr)
pseudo_controller_name = 'local-pi'
pseudo_controller_needs_lerrest = True
store_current_soln()

```

Types of pseudo-time *Controller* are related via the following inheritance diagram:



### 3.1.3 Stepper

A *Stepper* acts to advance the simulation by a single time-step. Specifically, a *Stepper* has a method named `step` which advances a *System* by a single time-step. There are eight types of *Stepper* available in PyFR 1.13.0:

`StdEulerStepper` [Click to show](#)

```

class pyfr.integrators.std.steppers.StdEulerStepper(backend, systemcls, rallocs, mesh, initsoln, cfg):
    _add(*args, subdims=None)
    _addv(consts, regidxs, subdims=None)
    _check_abort()
    _get_axnpby_kerns(*rs, subdims=None)
    _get_gndofs()
    _get_plugins(initsoln)
    _get_reduction_kerns(*rs, **kwargs)
    property _stepper_nfevals
    advance_to(t)
    call_plugin_dt(dt)

```

```
property cfgmeta
collect_stats(stats)

property controller_needs_errest
formulation = 'std'

static get_plugin_data_prefix(name, suffix)

property grad_soln

property nsteps

run()

property soln

step(t, dt)

stepper_has_errest = False
stepper_name = 'euler'
stepper_nregs = 2
stepper_order = 1

StdRK4Stepper Click to show

class pyfr.integrators.std.steppers.StdRK4Stepper(backend, systemcls, rallocs, mesh, initsoln, cfg)
    _add(*args, subdims=None)
    _addv(consts, regidxs, subdims=None)
    _check_abort()
    _get_axnpby_kerns(*rs, subdims=None)
    _get_gndofs()
    _get_plugins(initsoln)
    _get_reduction_kerns(*rs, **kwargs)
    property _stepper_nfevals
        advance_to(t)
        call_plugin_dt(dt)
    property cfgmeta
        collect_stats(stats)
    property controller_needs_errest
        formulation = 'std'
        static get_plugin_data_prefix(name, suffix)
    property grad_soln
    property nsteps
    run()
    property soln
    step(t, dt)
```

```

stepper_has_errest = False
stepper_name = 'rk4'
stepper_nregs = 3
stepper_order = 4

StdRK34Stepper Click to show

class pyfr.integrators.std.steppers.StdRK34Stepper(*args, **kwargs)
    _add(*args, subdims=None)
    _addv(consts, regidxs, subdims=None)
    _check_abort()
    _get_axnpby_kerns(*rs, subdims=None)
    _get_gndofs()
    _get_plugins(initsoln)
    _get_reduction_kerns(*rs, **kwargs)
    _get_rkvdh2_kerns(stage, r1, r2, rold=None, rerr=None)
property _stepper_nfevals
a = [0.32416573882874605, 0.5570978645055429, -0.08605491431272755]
advance_to(t)
b = [0.10407986927510238, 0.6019391368822611, 2.9750900268840206,
-2.681109033041384]
bhat = [0.3406814840808433, 0.09091523008632837, 2.866496742725443,
-2.298093456892615]
call_plugin_dt(dt)
property cfgmeta
collect_stats(stats)
property controller_needs_errest
formulation = 'std'
static get_plugin_data_prefix(name, suffix)
property grad_soln
property nsteps
run()
property soln
step(t, dt)
property stepper_has_errest
stepper_name = 'rk34'
property stepper_nregs
stepper_order = 3

```

*StdRK45Stepper Click to show*

```
class pyfr.integrators.std.steppers.StdRK45Stepper(*args, **kwargs)
    _add(*args, subdims=None)
    _adv(*consts, regidxs, subdims=None)
    _check_abort()
    _get_axnpby_kerns(*rs, subdims=None)
    _get_gndofs()
    _get_plugins(initsoln)
    _get_reduction_kerns(*rs, **kwargs)
    _get_rkvdh2_kerns(stage, r1, r2, rold=None, rerr=None)
    property _stepper_nfevals
        a = [0.22502245872571303, 0.5440433129514047, 0.14456824349399464,
              0.7866643421983568]
        advance_to(t)
        b = [0.05122930664033915, 0.3809548257264019, -0.3733525963923833,
              0.5925012850263623, 0.34866717899927996]
        bhat = [0.13721732210321927, 0.19188076232938728, -0.2292067211595315,
              0.6242946765438954, 0.27581396018302956]
        call_plugin_dt(dt)
    property cfgmeta
    collect_stats(stats)
    property controller_needs_errest
    formulation = 'std'
    static get_plugin_data_prefix(name, suffix)
    property grad_soln
    property nsteps
    run()
    property soln
    step(t, dt)
    property stepper_has_errest
    stepper_name = 'rk45'
    property stepper_nregs
    stepper_order = 4

StdTVDRK3Stepper Click to show
class pyfr.integrators.std.steppers.StdTVDRK3Stepper(backend, systemcls, rallocs, mesh, initsoln, cfg)
    _add(*args, subdims=None)
    _adv(*consts, regidxs, subdims=None)
    _check_abort()
    _get_axnpby_kerns(*rs, subdims=None)
```

---

```

_get_gndofs()
_get_plugins(initsoln)
_get_reduction_kerns(*rs, **kwargs)
property _stepper_nfevals
advance_to(t)
call_plugin_dt(dt)
property cfgmeta
collect_stats(stats)
property controller_needs_errest
formulation = 'std'
static get_plugin_data_prefix(name, suffix)
property grad_soln
property nsteps
run()
property soln
step(t, dt)
stepper_has_errest = False
stepper_name = 'tvd-rk3'
stepper_nregs = 3
stepper_order = 3

```

*DualBDF2Stepper* [Click to show](#)

```

class pyfr.integrators.dual.phys.steppers.DualBDF2Stepper(backend, systemcls, rallocs, mesh,
                                                          initsoln, cfg)
    _check_abort()
    _finalize_step()
    _get_plugins(initsoln)
    advance_to(t)
    call_plugin_dt(dt)
    property cfgmeta
    collect_stats(stats)
    formulation = 'dual'
    static get_plugin_data_prefix(name, suffix)
    property grad_soln
    nstages = 1
    property nsteps
    property pseudostepinfo
    run()

```

```
property soln
stage_nregs = 0
step(t, dt)
stepper_coeffs(dt)
stepper_name = 'bdf2'
property stepper_nregs
stepper_order = 2
stepper_static_coeffs = [-1.5, 2.0, -0.5]
property system
```

*DualBDF3Stepper* [Click to show](#)

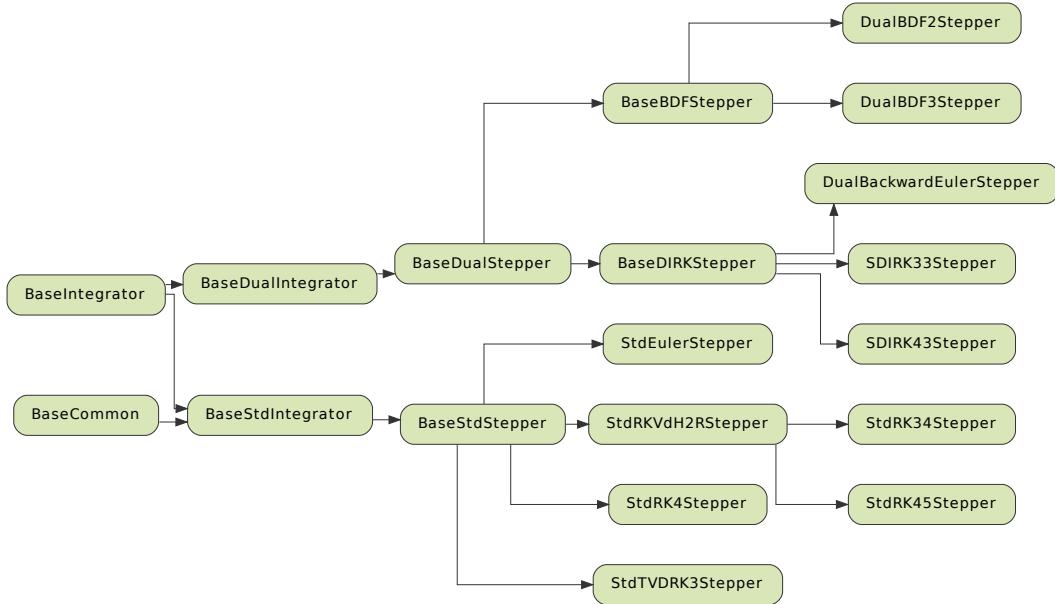
```
class pyfr.integrators.dual.phys.steppers.DualBDF3Stepper(backend, systemcls, rallocs, mesh,
                                                          initsoln, cfg)
    _check_abort()
    _finalize_step()
    _get_plugins(initsoln)
    advance_to(t)
    call_plugin_dt(dt)
    property cfgmeta
    collect_stats(stats)
    formulation = 'dual'
    static get_plugin_data_prefix(name, suffix)
    property grad_soln
    nstages = 1
    property nsteps
    property pseudostepinfo
    run()
    property soln
    stage_nregs = 0
    step(t, dt)
    stepper_coeffs(dt)
    stepper_name = 'bdf3'
    property stepper_nregs
    stepper_order = 3
    stepper_static_coeffs = [-1.833333333333333, 3.0, -1.5, 0.333333333333333]
    property system
```

*DualBackwardEulerStepper* [Click to show](#)

```
class pyfr.integrators.dual.phys.steppers.DualBackwardEulerStepper(*args, **kwargs)
```

```
_check_abort()
_finalize_step()
_get_plugins(initsoln)
a = [[1]]
advance_to(t)
call_plugin_dt(dt)
property cfgmeta
collect_stats(stats)
formulation = 'dual'
fsal = True
static get_plugin_data_prefix(name, suffix)
property grad_soln
nstages = 1
property nsteps
property pseudostepinfo
run()
property soln
property stage_nregs
step(t, dt)
stepper_coeffs(s, dt)
stepper_name = 'backward-euler'
stepper_nregs = 1
property system
```

Types of [Stepper](#) are related via the following inheritance diagram:



### 3.1.4 PseudoStepper

A `PseudoStepper` acts to advance the simulation by a single pseudo-time-step. They are used to converge implicit `Stepper` time-steps via a dual time-stepping formulation. There are six types of `PseudoStepper` available in PyFR 1.13.0:

`DualDenseRKPpseudoStepper` [Click to show](#)

```
class pyfr.integrators.dual.pseudo.pseudosteppers.DualDenseRKPpseudoStepper(*args, **kwargs)
    _add(*args, subdims=None)
    _addv(consts, regidxs, subdims=None)
    _get_axnpy_kerns(*rs, subdims=None)
    _get_gndofs()
    _get_reduction_kerns(*rs, **kwargs)
    property _pseudo_stepper_regidx
    _rhs_with_dts(t, uin, fout)
    property _source_regidx
    property _stage_regidx
    property _stepper_regidx
```

---

```

aux_nregs = 0
collect_stats(stats)
discard_oldest_source()
formulation = 'dual'
init_stage(currstg, stepper_coeffs)
property ntotiters
obtain_solution(bcoeffs)
property pseudo_stepper_nfevals
step(t)
store_current_soln()

```

*DualRK4PseudoStepper* [Click to show](#)

```

class pyfr.integrators.dual.pseudo.pseudosteppers.DualRK4PseudoStepper(backend, systemcls,
                                                                     rallocs, mesh, initsoln,
                                                                     cfg, stepper_nregs,
                                                                     stage_nregs, dt)

    _add(*args, subdims=None)
    _addv(consts, regidxs, subdims=None)
    _get_axnpby_kerns(*rs, subdims=None)
    _get_gndofs()
    _get_reduction_kerns(*rs, **kwargs)
    property _pseudo_stepper_regidx
    _rhs_with_dts(t, uin, fout)
    property _source_regidx
    property _stage_regidx
    property _stepper_regidx
    aux_nregs = 0
    collect_stats(stats)
    discard_oldest_source()
    formulation = 'dual'
    init_stage(currstg, stepper_coeffs)
    property ntotiters
    obtain_solution(bcoeffs)
    pseudo_stepper_has_lerrest = False
    pseudo_stepper_name = 'rk4'
    property pseudo_stepper_nfevals
    pseudo_stepper_nregs = 3
    pseudo_stepper_order = 4

```

```
step(t)
store_current_soln()
```

*DualTVDRK3PseudoStepper* [Click to show](#)

```
class pyfr.integrators.dual.pseudo.pseudosteppers.DualTVDRK3PseudoStepper(backend, systemcls,
                                                                           rallocs, mesh,
                                                                           initSoln, cfg,
                                                                           stepper_nregs,
                                                                           stage_nregs, dt)

    _add(*args, subdims=None)
    _addv(consts, regidxs, subdims=None)
    _get_axnpby_kerns(*rs, subdims=None)
    _get_gndofs()
    _get_reduction_kerns(*rs, **kwargs)
    property _pseudo_stepper_regidx
    _rhs_with_dts(t, uin, fout)
    property _source_regidx
    property _stage_regidx
    property _stepper_regidx
    aux_nregs = 0
    collect_stats(stats)
    discard_oldest_source()
    formulation = 'dual'
    init_stage(currstg, stepper_coeffs)
    property ntotiters
    obtain_solution(bcoeffs)
    pseudo_stepper_has_lerrest = False
    pseudo_stepper_name = 'tvd-rk3'
    property pseudo_stepper_nfevals
    pseudo_stepper_nregs = 3
    pseudo_stepper_order = 3
    step(t)
    store_current_soln()
```

*DualEulerPseudoStepper* [Click to show](#)

```
class pyfr.integrators.dual.pseudo.pseudosteppers.DualEulerPseudoStepper(backend, systemcls,
                                                                           rallocs, mesh,
                                                                           initSoln, cfg,
                                                                           stepper_nregs,
                                                                           stage_nregs, dt)

    _add(*args, subdims=None)
```

---

```

__addv(consts, regidxs, subdims=None)
__get_axnpby_kerns(*rs, subdims=None)
__get_gndofs()
__get_reduction_kerns(*rs, **kwargs)
property _pseudo stepper_regidx
__rhs_with_dts(t, uin, fout)
property _source_regidx
property _stage_regidx
property _stepper_regidx
aux_nregs = 0
collect_stats(stats)
discard_oldest_source()
formulation = 'dual'
init_stage(currstg, stepper_coeffs)
property ntotiters
obtain_solution(bcoeffs)
pseudo stepper_has_lerrest = False
pseudo stepper_name = 'euler'
property pseudo stepper_nfevals
pseudo stepper_nregs = 2
pseudo stepper_order = 1
step(t)
store_current_soln()

```

*DualRK34PseudoStepper* [Click to show](#)

```

class pyfr.integrators.dual.pseudo.pseudosteppers.DualRK34PseudoStepper(*args, **kwargs)
    __add__(*args, subdims=None)
    __addv(consts, regidxs, subdims=None)
    __get_axnpby_kerns(*rs, subdims=None)
    __get_gndofs()
    __get_reduction_kerns(*rs, **kwargs)
    __get_rkvdh2pseudo_kerns(stage, r1, r2, rold, rerr=None)
property _pseudo stepper_regidx
__rhs_with_dts(t, uin, fout)
property _source_regidx
property _stage_regidx
property _stepper_regidx

```

```
a = [0.32416573882874605, 0.5570978645055429, -0.08605491431272755]
aux_nregs = 0
b = [0.10407986927510238, 0.6019391368822611, 2.9750900268840206,
-2.681109033041384]
bhat = [0.3406814840808433, 0.09091523008632837, 2.866496742725443,
-2.298093456892615]
collect_stats(stats)
discard_oldest_source()
formulation = 'dual'
init_stage(currstg, stepper_coeffs)
property ntotiters
obtain_solution(bcoeffs)
property pseudo stepper has lerrest
pseudo stepper name = 'rk34'
property pseudo stepper nfevals
property pseudo stepper nregs
pseudo stepper order = 3
step(t)
store_current_soln()
```

*DualRK45PseudoStepper* [Click to show](#)

```
class pyfr.integrators.dual.pseudo.pseudosteppers.DualRK45PseudoStepper(*args, **kwargs)
    _add(*args, subdims=None)
    _addv(consts, regidxs, subdims=None)
    _get_axnpby_kerns(*rs, subdims=None)
    _get_gndofs()
    _get_reduction_kerns(*rs, **kwargs)
    _get_rkvdh2pseudo_kerns(stage, r1, r2, rold, rerr=None)
    property _pseudo stepper regidx
    _rhs_with_dts(t, uin, fout)
    property _source_regidx
    property _stage_regidx
    property _stepper_regidx
    a = [0.22502245872571303, 0.5440433129514047, 0.14456824349399464,
0.7866643421983568]
    aux_nregs = 0
    b = [0.05122930664033915, 0.3809548257264019, -0.3733525963923833,
0.5925012850263623, 0.34866717899927996]
```

```

bhat = [0.13721732210321927, 0.19188076232938728, -0.2292067211595315,
0.6242946765438954, 0.27581396018302956]

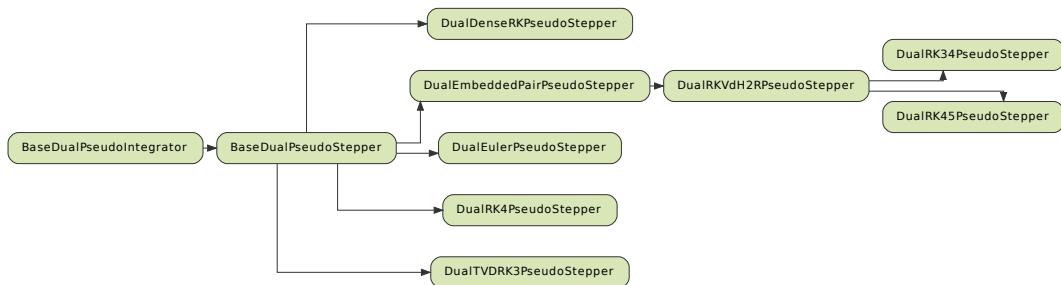
collect_stats(stats)
discard_oldest_source()
formulation = 'dual'
init_stage(currstg, stepper_coeffs)
property ntotiters
obtain_solution(bcoeffs)
property pseudo stepper_has_lerrest
pseudo stepper_name = 'rk45'
property pseudo stepper_nfevals
property pseudo stepper_nregs
pseudo stepper_order = 4
step(t)
store_current_soln()

```

Note that DualDenseRKPseudoStepper includes families of *PseudoStepper* whose coefficients are read from .txt files named thus:

*{scheme name}-s{stage count}-p{temporal order}-sp{optimal spatial polynomial order}.txt*

Types of *PseudoStepper* are related via the following inheritance diagram:



### 3.1.5 System

A *System* holds information/data for the system, including *Elements*, *Interfaces*, and the *Backend* with which the simulation is to run. A *System* has a method named `rhs`, which obtains the divergence of the flux (the ‘right-hand-side’) at each solution point. The method `rhs` invokes various kernels which have been pre-generated and loaded into queues. A *System* also has a method named `_gen_kernels` which acts to generate all the kernels required by a particular *System*. A kernel is an instance of a ‘one-off’ class with a method named `run` that implements the required kernel functionality. Individual kernels are produced by a kernel provider. PyFR 1.13.0 has various types of kernel provider. A *Pointwise Kernel Provider* produces point-wise kernels such as Riemann solvers and flux functions etc. These point-wise kernels are specified using an in-built platform-independent templating language derived from *Mako*, henceforth referred to as *PyFR-Mako*. There are four types of *System* available in PyFR 1.13.0:

*ACEulerSystem* [Click to show](#)

```
class pyfr.solvers.aceuler.system.ACEulerSystem(backend, rallocs, mesh, initsoln, nregs, cfg)
    _gen_kernels(nregs, eles, iint, mpiint, bcint)
    _gen_mpireqs(mpiint)
    _get_kernels(uinbank, foutbank)
    _load_bc_inters(rallocs, mesh, elemap)
    _load_eles(rallocs, mesh, initsoln, nregs, nonce)
    _load_int_inters(rallocs, mesh, elemap)
    _load_mpi_inters(rallocs, mesh, elemap)
    _nonce_seq = count(0)

bbcinterscls
    alias of pyfr.solvers.aceuler.inters.ACEulerBaseBCInters

compute_grads(t, uinbank)
ele_scal_upts(idx)
elementscls
    alias of pyfr.solvers.aceuler.elements.ACEulerElements
filt(uinoutbank)
intinterscls
    alias of pyfr.solvers.aceuler.inters.ACEulerIntInters
mpiinterscls
    alias of pyfr.solvers.aceuler.inters.ACEulerMPIInters
name = 'ac-euler'
rhs(t, uinbank, foutbank)
```

*ACNavierStokesSystem* [Click to show](#)

```
class pyfr.solvers.acnavstokes.system.ACNavierStokesSystem(backend, rallocs, mesh, initsoln, nregs,
    cfg)
    _gen_kernels(nregs, eles, iint, mpiint, bcint)
    _gen_mpireqs(mpiint)
    _get_kernels(uinbank, foutbank)
    _load_bc_inters(rallocs, mesh, elemap)
    _load_eles(rallocs, mesh, initsoln, nregs, nonce)
```

```

_load_int_inters(rallocs, mesh, elemap)
_load_mpi_inters(rallocs, mesh, elemap)
_nonce_seq = count(0)

bbcinterscls
    alias of pyfr.solvers.acnavstokes.inters.ACNavierStokesBaseBCInters

compute_grads(t, uinbank)
ele_scal_upts(idx)
elementscls
    alias of pyfr.solvers.acnavstokes.elements.ACNavierStokesElements
filt(uinoutbank)
intinterscls
    alias of pyfr.solvers.acnavstokes.inters.ACNavierStokesIntInters
mpiinterscls
    alias of pyfr.solvers.acnavstokes.inters.ACNavierStokesMPIInters
name = 'ac-navier-stokes'
rhs(t, uinbank, foutbank)

```

*EulerSystem* Click to show

```

class pyfr.solvers.euler.system.EulerSystem(backend, rallocs, mesh, initsohn, nregs, cfg)
    _gen_kernels(nregs, eles, iint, mpiint, bcint)
    _gen_mpireqs(mpiint)
    _get_kernels(uinbank, foutbank)
    _load_bc_inters(rallocs, mesh, elemap)
    _load_eles(rallocs, mesh, initsohn, nregs, nonce)
    _load_int_inters(rallocs, mesh, elemap)
    _load_mpi_inters(rallocs, mesh, elemap)
    _nonce_seq = count(0)

bbcinterscls
    alias of pyfr.solvers.euler.inters.EulerBaseBCInters

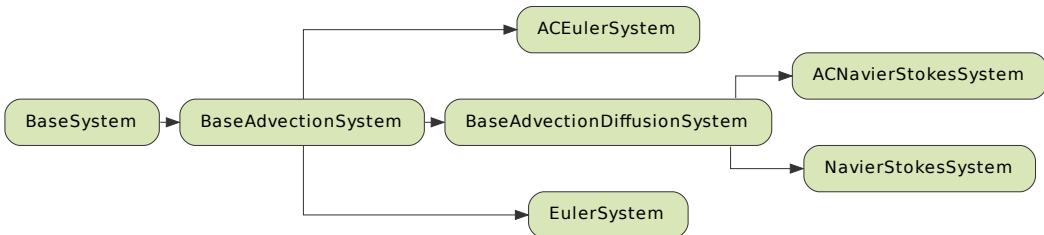
compute_grads(t, uinbank)
ele_scal_upts(idx)
elementscls
    alias of pyfr.solvers.euler.elements.EulerElements
filt(uinoutbank)
intinterscls
    alias of pyfr.solvers.euler.inters.EulerIntInters
mpiinterscls
    alias of pyfr.solvers.euler.inters.EulerMPIInters
name = 'euler'
rhs(t, uinbank, foutbank)

```

*NavierStokesSystem* [Click to show](#)

```
class pyfr.solvers.navstokes.system.NavierStokesSystem(backend, rallocs, mesh, initSoln, nregs, cfg)
    _gen_kernels(nregs, eles, iint, mpiint, bcint)
    _gen_mpireqs(mpiint)
    _get_kernels(uinbank, foutbank)
    _load_bc_inters(rallocs, mesh, elemap)
    _load_eles(rallocs, mesh, initSoln, nregs, nonce)
    _load_int_inters(rallocs, mesh, elemap)
    _load_mpi_inters(rallocs, mesh, elemap)
    _nonce_seq = count(0)
bbcinterscls
    alias of pyfr.solvers.navstokes.inters.NavierStokesBaseBCInters
compute_grads(t, uinbank)
ele_scal_upts(idx)
elementscls
    alias of pyfr.solvers.navstokes.elements.NavierStokesElements
filt(uinoutbank)
intinterscls
    alias of pyfr.solvers.navstokes.inters.NavierStokesIntInters
mpiinterscls
    alias of pyfr.solvers.navstokes.inters.NavierStokesMPIInters
name = 'navier-stokes'
rhs(t, uinbank, foutbank)
```

Types of *System* are related via the following inheritance diagram:



### 3.1.6 Elements

An *Elements* holds information/data for a group of elements. There are four types of *Elements* available in PyFR 1.13.0:

**ACEulerElements** [Click to show](#)

```
class pyfr.solvers.aceuler.elements.ACEulerElements(basiscls, eles, cfg)
    _gen_pnorm_fpts()

    property _mag_pnorm_fpts
    property _mesh_regions
    property _norm_pnorm_fpts
    property _ploc_in_src_exprs
    property _scratch_bufs
    _slice_mat(mat, region, ra=None, rb=None)
    property _smats_djacs_mpts
    property _soln_in_src_exprs
    property _src_exprs
    property _srtd_face_fpts
    static con_to_pri(convs, cfg)
    convarmap = {2: ['p', 'u', 'v'], 3: ['p', 'u', 'v', 'w']}
    dualcoeffs = {2: ['u', 'v'], 3: ['u', 'v', 'w']}
    formulations = ['dual']
    get_mag_pnorms(eidx, fidx)
    get_mag_pnorms_for_inter(eidx, fidx)
    get_norm_pnorms(eidx, fidx)
    get_norm_pnorms_for_inter(eidx, fidx)
    get_ploc_for_inter(eidx, fidx)
    get_scal_fpts_for_inter(eidx, fidx)
    get_vect_fpts_for_inter(eidx, fidx)
    opmat(expr)
    ploc_at(name, side=None)
    ploc_at_np(name)
    property plocfpts
    static pri_to_con(pris, cfg)
    privarmap = {2: ['p', 'u', 'v'], 3: ['p', 'u', 'v', 'w']}
    property qpts
    rcpdjac_at(name, side=None)
```

```
rcpdjac_at_np(name)
set_backend(*args, **kwargs)
set_ics_from_cfg()
set_ics_from_soln(solnmat, solncfg)
sliceat()
smat_at(name, side=None)
smat_at_np(name)
property upts
visvarmap = {2: [('velocity', ['u', 'v']), ('pressure', ['p'])], 3: [('velocity', ['u', 'v', 'w']), ('pressure', ['p'])]}

ACNavierStokesElements Click to show

class pyfr.solvers.acnavstokes.elements.ACNavierStokesElements(basiscls, eles, cfg)
    _gen_pnorm_fpts()

    property _mag_pnorm_fpts
    property _mesh_regions
    property _norm_pnorm_fpts
    property _ploc_in_src_exprs
    property _scratch_bufs
    _slice_mat(mat, region, ra=None, rb=None)
    property _smats_djacs_mpts
    property _soln_in_src_exprs
    property _src_exprs
    property _srtd_face_fpts
    static con_to_pri(convs, cfg)
    convvarmap = {2: ['p', 'u', 'v'], 3: ['p', 'u', 'v', 'w']}
    dualcoeffs = {2: ['u', 'v'], 3: ['u', 'v', 'w']}
    formulations = ['dual']
    get_artvisc_fpts_for_inter(eidx, fidx)
    get_mag_pnorms(eidx, fidx)
    get_mag_pnorms_for_inter(eidx, fidx)
    get_norm_pnorms(eidx, fidx)
    get_norm_pnorms_for_inter(eidx, fidx)
    get_ploc_for_inter(eidx, fidx)
    get_scal_fpts_for_inter(eidx, fidx)
    get_vect_fpts_for_inter(eidx, fidx)
    static grad_con_to_pri(cons, grad_cons, cfg)
    opmat(expr)
```

```

ploc_at(name, side=None)
ploc_at_np(name)
property plocfpts
static pri_to_con(pris, cfg)
privaremap = {2: ['p', 'u', 'v'], 3: ['p', 'u', 'v', 'w']}
property qpts
rcpdjac_at(name, side=None)
rcpdjac_at_np(name)
set_backend(*args, **kwargs)
set_ics_from_cfg()
set_ics_from_soln(solnmat, solncfg)
sliceat()
smat_at(name, side=None)
smat_at_np(name)
property upto
visvarmap = {2: [('velocity', ['u', 'v']), ('pressure', ['p'])], 3: [('velocity', ['u', 'v', 'w']), ('pressure', ['p'])]}

```

*EulerElements* Click to show

```

class pyfr.solvers.euler.elements.EulerElements(basiscls, eles, cfg)
    _gen_pnorm_fpts()

    property _mag_pnorm_fpts
    property _mesh_regions
    property _norm_pnorm_fpts
    property _ploc_in_src_exprs
    property _scratch_bufs
    _slice_mat(mat, region, ra=None, rb=None)
    property _smats_djacs_mpts
    property _soln_in_src_exprs
    property _src_exprs
    property _srtd_face_fpts
    static con_to_pri(cons, cfg)
    convvarmap = {2: ['rho', 'rhou', 'rhov', 'E'], 3: ['rho', 'rhou', 'rhov', 'rhow', 'E']}
    dualcoeffs = {2: ['rho', 'rhou', 'rhov', 'E'], 3: ['rho', 'rhou', 'rhov', 'rhow', 'E']}
    formulations = ['std', 'dual']
    get_mag_pnorms(eidx, fidx)

```

```
get_mag_pnorms_for_inter(eidx, fidx)
get_norm_pnorms(eidx, fidx)
get_norm_pnorms_for_inter(eidx, fidx)
get_ploc_for_inter(eidx, fidx)
get_scal_fpts_for_inter(eidx, fidx)
get_vect_fpts_for_inter(eidx, fidx)
opmat(expr)

ploc_at(name, side=None)
ploc_at_np(name)
property plocfpts
static pri_to_con(pris, cfg)
privaremap = {2: ['rho', 'u', 'v', 'p'], 3: ['rho', 'u', 'v', 'w', 'p']}
property qpts
rcpdjac_at(name, side=None)
rcpdjac_at_np(name)
set_backend(*args, **kwargs)
set_ics_from_cfg()
set_ics_from_soln(solnmat, solncfg)
sliceat()
smat_at(name, side=None)
smat_at_np(name)
property upto
visvarmap = {2: [('density', ['rho']), ('velocity', ['u', 'v']), ('pressure', ['p'])], 3: [('density', ['rho']), ('velocity', ['u', 'v', 'w']), ('pressure', ['p'])]}
```

*NavierStokesElements* Click to show

```
class pyfr.solvers.navstokes.elements.NavierStokesElements(basiscls, eles, cfg)
    _gen_pnorm_fpts()

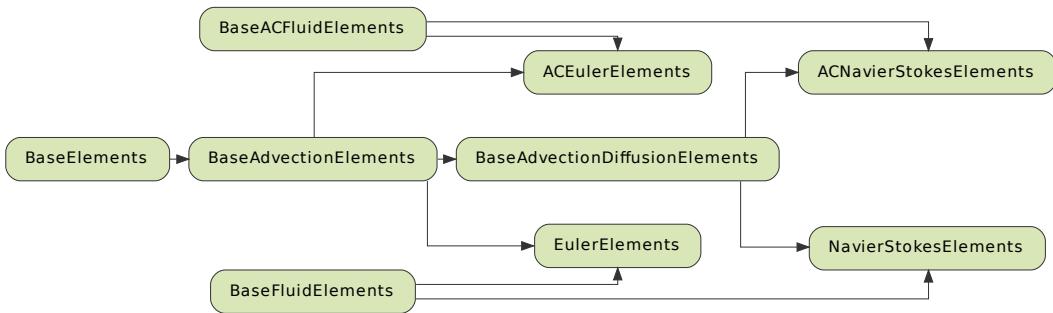
    property _mag_pnorm_fpts
    property _mesh_regions
    property _norm_pnorm_fpts
    property _ploc_in_src_exprs
    property _scratch_bufs
    _slice_mat(mat, region, ra=None, rb=None)
    property _smats_djacs_mpts
    property _soln_in_src_exprs
    property _src_exprs
```

```

property _srtd_face_fpts
static con_to_pri(cons, cfg)
convarmap = {2: ['rho', 'rhou', 'rhov', 'E'], 3: ['rho', 'rhou', 'rhov', 'rhow', 'E']}
dualcoeffs = {2: ['rho', 'rhou', 'rhov', 'E'], 3: ['rho', 'rhou', 'rhov', 'rhow', 'E']}
formulations = ['std', 'dual']
get_artvisc_fpts_for_inter(eidx, fidx)
get_mag_pnorms(eidx, fidx)
get_mag_pnorms_for_inter(eidx, fidx)
get_norm_pnorms(eidx, fidx)
get_norm_pnorms_for_inter(eidx, fidx)
get_ploc_for_inter(eidx, fidx)
get_scal_fpts_for_inter(eidx, fidx)
get_vect_fpts_for_inter(eidx, fidx)
static grad_con_to_pri(cons, grad_cons, cfg)
opmat(expr)
ploc_at(name, side=None)
ploc_at_np(name)
property plocfpts
static pri_to_con(pris, cfg)
privarmap = {2: ['rho', 'u', 'v', 'p'], 3: ['rho', 'u', 'v', 'w', 'p']}
property qpts
rcpdjac_at(name, side=None)
rcpdjac_at_np(name)
set_backend(*args, **kwargs)
set_ics_from_cfg()
set_ics_from_soln(solnmat, solncfg)
shockvar = 'rho'
sliceat()
smat_at(name, side=None)
smat_at_np(name)
property upto
visvarmap = {2: [('density', ['rho']), ('velocity', ['u', 'v']), ('pressure', ['p'])], 3: [('density', ['rho']), ('velocity', ['u', 'v', 'w']), ('pressure', ['p'])]}

```

Types of *Elements* are related via the following inheritance diagram:



### 3.1.7 Interfaces

An *Interfaces* holds information/data for a group of interfaces. There are eight types of (non-boundary) *Interfaces* available in PyFR 1.13.0:

*ACEulerIntInters* [Click to show](#)

```
class pyfr.solvers.aceuler.inters.ACEulerIntInters(*args, **kwargs)
    _const_mat(inter, meth)
    _gen_perm(lhs, rhs)
    _get_perm_for_view(inter, meth)
    _scal_view(inter, meth)
    _scal_xchg_view(inter, meth)
    _set_external(name, spec, value=None)
    _vect_view(inter, meth)
    _vect_xchg_view(inter, meth)
    _view(inter, meth, vshape=())
    _xchg_view(inter, meth, vshape=())
    prepare(t)
```

*ACEulerMPIInters* [Click to show](#)

```
class pyfr.solvers.aceuler.inters.ACEulerMPIInters(*args, **kwargs)
    MPI_TAG = 2314
    _const_mat(inter, meth)
```

---

```

_get_perm_for_view(inter, meth)
scal_view(inter, meth)
scal_xchg_view(inter, meth)
set_external(name, spec, value=None)
vect_view(inter, meth)
vect_xchg_view(inter, meth)
view(inter, meth, vshape=())
xchg_view(inter, meth, vshape=())
prepare(t)

```

*ACNavierStokesIntInters* [Click to show](#)

```

class pyfr.solvers.acnavstokes.inters.ACNavierStokesIntInters(be, lhs, rhs, elemap, cfg)
    _const_mat(inter, meth)
    _gen_perm(lhs, rhs)
    _get_perm_for_view(inter, meth)
    scal_view(inter, meth)
    scal_xchg_view(inter, meth)
    set_external(name, spec, value=None)
    vect_view(inter, meth)
    vect_xchg_view(inter, meth)
    view(inter, meth, vshape=())
    xchg_view(inter, meth, vshape=())
    prepare(t)

```

*ACNavierStokesMPIInters* [Click to show](#)

```

class pyfr.solvers.acnavstokes.inters.ACNavierStokesMPIInters(be, lhs, rhsrank, rallocs, elemap,
                                                               cfg)
    MPI_TAG = 2314
    _const_mat(inter, meth)
    _get_perm_for_view(inter, meth)
    scal_view(inter, meth)
    scal_xchg_view(inter, meth)
    set_external(name, spec, value=None)
    vect_view(inter, meth)
    vect_xchg_view(inter, meth)
    view(inter, meth, vshape=())
    xchg_view(inter, meth, vshape=())
    prepare(t)

```

*EulerIntInters* [Click to show](#)

```
class pyfr.solvers.euler.inters.EulerIntInters(*args, **kwargs)
    _const_mat(inter, meth)
    _gen_perm(lhs, rhs)
    _get_perm_for_view(inter, meth)
    _scal_view(inter, meth)
    _scal_xchg_view(inter, meth)
    _set_external(name, spec, value=None)
    _vect_view(inter, meth)
    _vect_xchg_view(inter, meth)
    _view(inter, meth, vshape=())
    _xchg_view(inter, meth, vshape=())
    prepare(t)
```

*EulerMPIInters* [Click to show](#)

```
class pyfr.solvers.euler.inters.EulerMPIInters(*args, **kwargs)
    MPI_TAG = 2314
    _const_mat(inter, meth)
    _get_perm_for_view(inter, meth)
    _scal_view(inter, meth)
    _scal_xchg_view(inter, meth)
    _set_external(name, spec, value=None)
    _vect_view(inter, meth)
    _vect_xchg_view(inter, meth)
    _view(inter, meth, vshape=())
    _xchg_view(inter, meth, vshape=())
    prepare(t)
```

*NavierStokesIntInters* [Click to show](#)

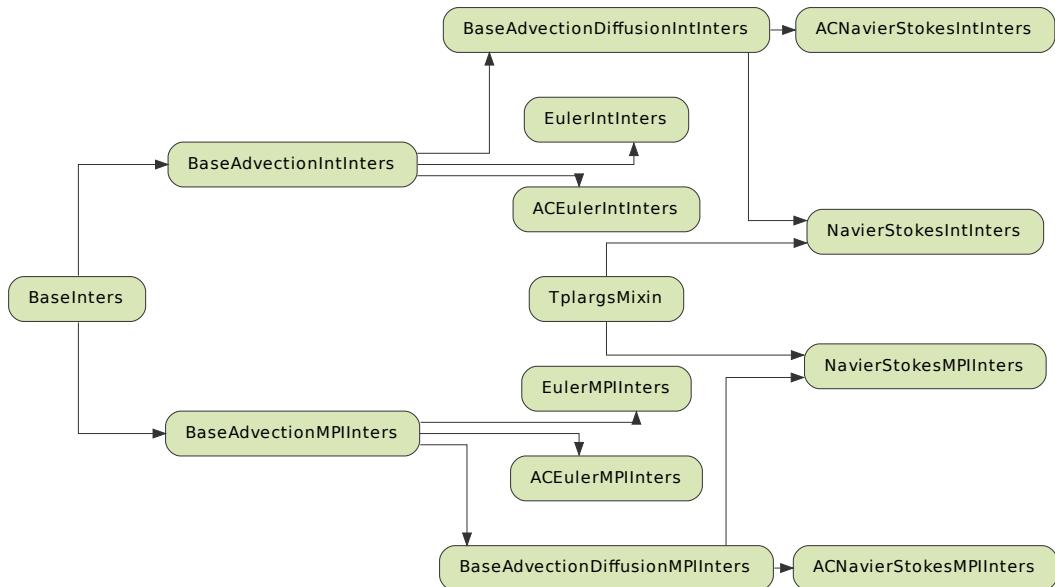
```
class pyfr.solvers.navstokes.inters.NavierStokesIntInters(be, lhs, rhs, elemap, cfg)
    _const_mat(inter, meth)
    _gen_perm(lhs, rhs)
    _get_perm_for_view(inter, meth)
    _scal_view(inter, meth)
    _scal_xchg_view(inter, meth)
    _set_external(name, spec, value=None)
    _vect_view(inter, meth)
    _vect_xchg_view(inter, meth)
    _view(inter, meth, vshape=())
    _xchg_view(inter, meth, vshape=())
```

```
prepare(t)
```

*NavierStokesMPIInters* Click to show

```
class pyfr.solvers.navstokes.inters.NavierStokesMPIInters(be, lhs, rhsrank, rallocs, elemap, cfg)
    MPI_TAG = 2314
    _const_mat(inter, meth)
    _get_perm_for_view(inter, meth)
    _scal_view(inter, meth)
    _scal_xchg_view(inter, meth)
    _set_external(name, spec, value=None)
    _vect_view(inter, meth)
    _vect_xchg_view(inter, meth)
    _view(inter, meth, vshape=())
    _xchg_view(inter, meth, vshape=())
prepare(t)
```

Types of (non-boundary) *Interfaces* are related via the following inheritance diagram:



### 3.1.8 Backend

A *Backend* holds information/data for a backend. There are four types of *Backend* available in PyFR 1.13.0:

*CUDABackend* [Click to show](#)

```
class pyfr.backends.cuda.base.CUDABackend(cfg)
    _malloc_impl(nbytes)
    alias(obj, aobj)
    blocks = False
    commit()
    const_matrix(initval, extent=None, tags={})
    kernel(name, *args, **kwargs)
    property lookup
    malloc(obj, extent)
    matrix(ioshape, initval=None, extent=None, aliases=None, tags={})
    matrix_slice(mat, ra, rb, ca, cb)
    name = 'cuda'
    queue()
    view(matmap, rmap, cmap, rstridemap=None, vshape=(), tags={})
    xchg_matrix(ioshape, initval=None, extent=None, aliases=None, tags={})
    xchg_matrix_for_view(view, tags={})
    xchg_view(matmap, rmap, cmap, rstridemap=None, vshape=(), tags={})
```

*HIPBackend* [Click to show](#)

```
class pyfr.backends.hip.base.HIPBackend(cfg)
    _malloc_impl(nbytes)
    alias(obj, aobj)
    blocks = False
    commit()
    const_matrix(initval, extent=None, tags={})
    kernel(name, *args, **kwargs)
    property lookup
    malloc(obj, extent)
    matrix(ioshape, initval=None, extent=None, aliases=None, tags={})
    matrix_slice(mat, ra, rb, ca, cb)
    name = 'hip'
    queue()
    view(matmap, rmap, cmap, rstridemap=None, vshape=(), tags={})
    xchg_matrix(ioshape, initval=None, extent=None, aliases=None, tags={})
```

---

```
xchg_matrix_for_view(view, tags={})
xchg_view(matmap, rmap, cmap, rstridemap=None, vshape=(), tags={})
```

*OpenCLBackend* [Click to show](#)

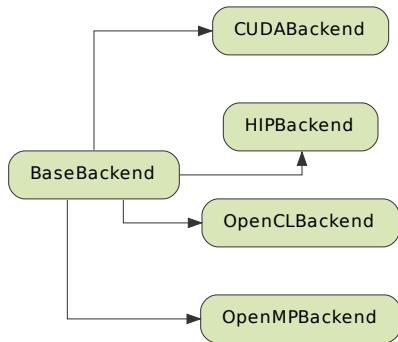
```
class pyfr.backends.opencl.base.OpenCLBackend(cfg)
    _malloc_impl(nbytes)
    alias(obj, aobj)
    blocks = False
    commit()
    const_matrix(initval, extent=None, tags={})
    kernel(name, *args, **kwargs)
    property lookup
    malloc(obj, extent)
    matrix(ioshape, initval=None, extent=None, aliases=None, tags={})
    matrix_slice(mat, ra, rb, ca, cb)
    name = 'opencl'
    queue()
    view(matmap, rmap, cmap, rstridemap=None, vshape=(), tags={})
    xchg_matrix(ioshape, initval=None, extent=None, aliases=None, tags={})
    xchg_matrix_for_view(view, tags={})
    xchg_view(matmap, rmap, cmap, rstridemap=None, vshape=(), tags={})
```

*OpenMPBackend* [Click to show](#)

```
class pyfr.backends.openmp.base.OpenMPBackend(cfg)
    _malloc_impl(nbytes)
    alias(obj, aobj)
    blocks = True
    commit()
    const_matrix(initval, extent=None, tags={})
    kernel(name, *args, **kwargs)
    property lookup
    malloc(obj, extent)
    matrix(ioshape, initval=None, extent=None, aliases=None, tags={})
    matrix_slice(mat, ra, rb, ca, cb)
    name = 'openmp'
    queue()
    view(matmap, rmap, cmap, rstridemap=None, vshape=(), tags={})
    xchg_matrix(ioshape, initval=None, extent=None, aliases=None, tags={})
    xchg_matrix_for_view(view, tags={})
```

`xchg_view(matmap, rmap, cmap, rstridemap=None, vshape=(), tags={})`

Types of `Backend` are related via the following inheritance diagram:



### 3.1.9 Pointwise Kernel Provider

A `Pointwise Kernel Provider` produces point-wise kernels. Specifically, a `Pointwise Kernel Provider` has a method named `register`, which adds a new method to an instance of a `Pointwise Kernel Provider`. This new method, when called, returns a kernel. A kernel is an instance of a ‘one-off’ class with a method named `run` that implements the required kernel functionality. The kernel functionality itself is specified using `PyFR-Mako`. Hence, a `Pointwise Kernel Provider` also has a method named `_render_kernel`, which renders `PyFR-Mako` into low-level platform-specific code. The `_render_kernel` method first sets the context for Mako (i.e. details about the `Backend` etc.) and then uses Mako to begin rendering the `PyFR-Mako` specification. When Mako encounters a `pyfr:kernel` an instance of a `Kernel Generator` is created, which is used to render the body of the `pyfr:kernel`. There are four types of `Pointwise Kernel Provider` available in PyFR 1.13.0:

`CUDAPointwiseKernelProvider` [Click to show](#)

```
class pyfr.backends.cuda.provider.CUDAPointwiseKernelProvider(backend)
    _build_arglst(dims, argn, argt, argdict)
    _build_kernel(name, src, argtypes)
    _instantiate_kernel(dims, fun, arglst, argmv)
    _render_kernel(name, mod, extrns, tplargs)
    kernel_generator_cls
        alias of pyfr.backends.cuda.generator.CUDAKernelGenerator
    register(mod)
```

*HIPPointwiseKernelProvider* Click to show

```
class pyfr.backends.hip.provider.HIPPointwiseKernelProvider(*args, **kwargs)
    _build_arglst(dims, argn, argt, argdict)
    _build_kernel(name, src, argtypes)
    _instantiate_kernel(dims, fun, arglst, argmv)
    _render_kernel(name, mod, extrns, tplargs)
    kernel_generator_cls = None
    register(mod)
```

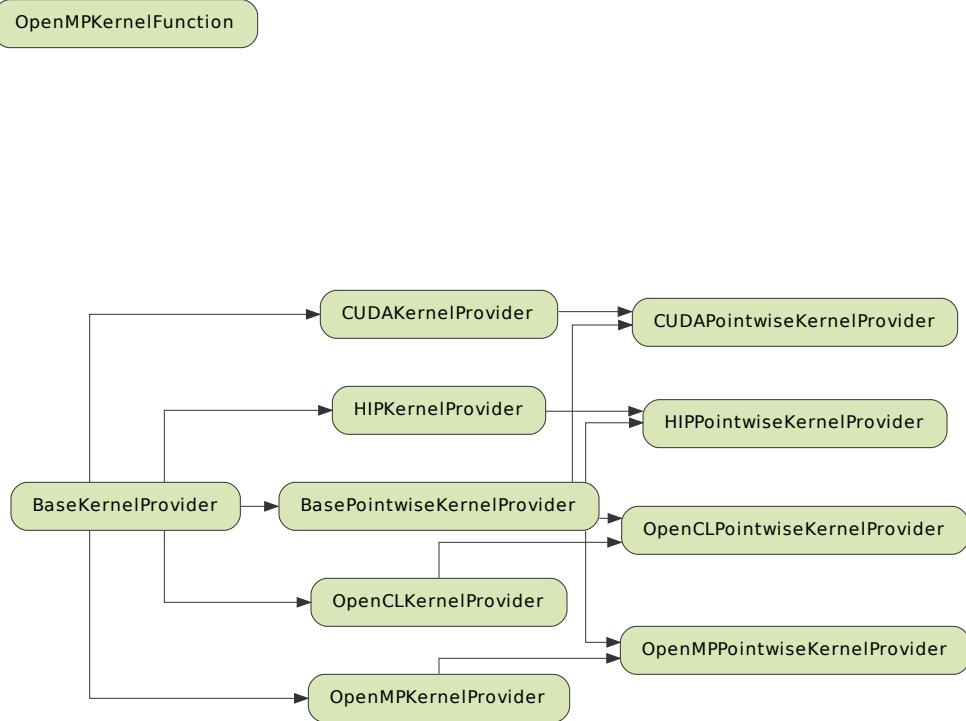
*OpenCLPointwiseKernelProvider* Click to show

```
class pyfr.backends.opencl.provider.OpenCLPointwiseKernelProvider(backend)
    _build_arglst(dims, argn, argt, argdict)
    _build_kernel(name, src, argtypes)
    _build_program(src)
    _instantiate_kernel(dims, fun, arglst, argmv)
    _render_kernel(name, mod, extrns, tplargs)
    kernel_generator_cls
        alias of pyfr.backends.opencl.generator.OpenCLKernelGenerator
    register(mod)
```

*OpenMPPointwiseKernelProvider* Click to show

```
class pyfr.backends.openmp.provider.OpenMPPointwiseKernelProvider(backend)
    _build_arglst(dims, argn, argt, argdict)
    _build_function(name, src, argtypes, restype=None)
    _build_kernel(name, src, argtypes)
    _build_library(src)
    _get_arg_cls(argtypes)
    _instantiate_kernel(dims, fun, arglst, argmv)
    _render_kernel(name, mod, extrns, tplargs)
    kernel_generator_cls
        alias of pyfr.backends.openmp.generator.OpenMPKernelGenerator
    register(mod)
```

Types of *Pointwise Kernel Provider* are related via the following inheritance diagram:



### 3.1.10 Kernel Generator

A *Kernel Generator* renders the *PyFR-Mako* in a `pyfr:kernel` into low-level platform-specific code. Specifically, a *Kernel Generator* has a method named `render`, which applies *Backend* specific regex and adds *Backend* specific ‘boiler plate’ code to produce the low-level platform-specific source – which is compiled, linked, and loaded. There are four types of *Kernel Generator* available in PyFR 1.13.0:

`CUDAKernelGenerator` [Click to show](#)

```
class pyfr.backends.cuda.generator.CUDAKernelGenerator(*args, **kwargs):
    _deref_arg_array_1d(arg)
    _deref_arg_array_2d(arg)
    _deref_arg_view(arg)
    _render_body(body)
    _render_spec()
    argspec()
    ldim_size(name, *factor)
```

---

```
needs_ldim(arg)
```

```
render()
```

*HIPKernelGenerator* [Click to show](#)

```
class pyfr.backends.hip.generator.HIPKernelGenerator(*args, **kwargs)
    _deref_arg_array_1d(arg)
    _deref_arg_array_2d(arg)
    _deref_arg_view(arg)
    _render_body(body)
    _render_spec()
    argspec()
    block1d = None
    block2d = None
    ldim_size(name, *factor)
    needs_ldim(arg)
    render()
```

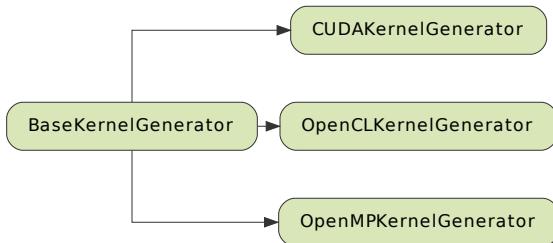
*OpenCLKernelGenerator* [Click to show](#)

```
class pyfr.backends.opencl.generator.OpenCLKernelGenerator(*args, **kwargs)
    _deref_arg_array_1d(arg)
    _deref_arg_array_2d(arg)
    _deref_arg_view(arg)
    _render_body(body)
    _render_spec()
    argspec()
    ldim_size(name, *factor)
    needs_ldim(arg)
    render()
```

*OpenMPKernelGenerator* [Click to show](#)

```
class pyfr.backends.openmp.generator.OpenMPKernelGenerator(name, ndim, args, body, fpdtype)
    _deref_arg_array_1d(arg)
    _deref_arg_array_2d(arg)
    _deref_arg_view(arg)
    _render_args(argn)
    _render_body(body)
    argspec()
    ldim_size(name, *factor)
    needs_ldim(arg)
    render()
```

Types of *Kernel Generator* are related via the following inheritance diagram:



## 3.2 PyFR-Mako

### 3.2.1 PyFR-Mako Kernels

PyFR-Mako kernels are specifications of point-wise functionality that can be invoked directly from within PyFR. They are opened with a header of the form:

```
<%pyfr:kernel name='kernel-name' ndim='data-dimensionality' [argument-name='argument-intent argument-attribute argument-data-type' ...]>
```

where

1. **kernel-name** — name of kernel  
*string*
2. **data-dimensionality** — dimensionality of data  
*int*
3. **argument-name** — name of argument  
*string*
4. **argument-intent** — intent of argument  
*in | out | inout*
5. **argument-attribute** — attribute of argument  
*mpi | scalar | view*
6. **argument-data-type** — data type of argument  
*string*

and are closed with a footer of the form:

```
</%pyfr:kernel>
```

### 3.2.2 PyFR-Mako Macros

PyFR-Mako macros are specifications of point-wise functionality that cannot be invoked directly from within PyFR, but can be embedded into PyFR-Mako kernels. PyFR-Mako macros can be viewed as building blocks for PyFR-mako kernels. They are opened with a header of the form:

```
<%pyfr:macro name='macro-name' params='[parameter-name, ...]'>
```

where

1. **macro-name** — name of macro  
*string*
2. **parameter-name** — name of parameter  
*string*

and are closed with a footer of the form:

```
</%pyfr:macro>
```

PyFR-Mako macros are embedded within a kernel using an expression of the following form:

```
 ${pyfr.expand('macro-name', ['parameter-name', ...])};
```

where

1. **macro-name** — name of the macro  
*string*
2. **parameter-name** — name of parameter  
*string*

### 3.2.3 Syntax

#### 3.2.3.1 Basic Functionality

Basic functionality can be expressed using a restricted subset of the C programming language. Specifically, use of the following is allowed:

1. **+**, **-**, **\***, **/** — basic arithmetic
2. **sin**, **cos**, **tan** — basic trigonometric functions
3. **exp** — exponential
4. **pow** — power
5. **fabs** — absolute value
6. **output = ( condition ? satisfied : unsatisfied )** — ternary if
7. **min** — minimum

8. `max` — maximum

However, conditional if statements, as well as for/while loops, are not allowed.

### 3.2.3.2 Expression Substitution

Mako expression substitution can be used to facilitate PyFR-Mako kernel specification. A Python expression expression prescribed thus  `${expression}` is substituted for the result when the PyFR-Mako kernel specification is interpreted at runtime.

Example:

```
E = s[${ndims - 1}]
```

### 3.2.3.3 Conditionals

Mako conditionals can be used to facilitate PyFR-Mako kernel specification. Conditionals are opened with `% if` condition: and closed with `% endif`. Note that such conditionals are evaluated when the PyFR-Mako kernel specification is interpreted at runtime, they are not embedded into the low-level kernel.

Example:

```
% if ndims == 2:  
    fout[0][1] += t_xx;      fout[1][1] += t_xy;  
    fout[0][2] += t_xy;      fout[1][2] += t_yy;  
    fout[0][3] += u*t_xx + v*t_xy + ${-c['mu']*c['gamma']/c['Pr']}*T_x;  
    fout[1][3] += u*t_xy + v*t_yy + ${-c['mu']*c['gamma']/c['Pr']}*T_y;  
% endif
```

### 3.2.3.4 Loops

Mako loops can be used to facilitate PyFR-Mako kernel specification. Loops are opened with `% for` condition: and closed with `% endfor`. Note that such loops are unrolled when the PyFR-Mako kernel specification is interpreted at runtime, they are not embedded into the low-level kernel.

Example:

```
% for i in range(ndims):  
    rhov[ ${i} ] = s[ ${i + 1} ];  
    v[ ${i} ] = invrho*rhov[ ${i} ];  
% endfor
```

## PERFORMANCE TUNING

The following sections contain best practices for *tuning* the performance of PyFR. Note, however, that it is typically not worth pursuing the advice in this section until a simulation is working acceptably and generating the desired results.

### 4.1 OpenMP Backend

#### 4.1.1 libxsmm

If libxsmm is not available then PyFR will make use of GiMMiK for all matrix-matrix multiplications. Although functional, the performance is typically sub-par compared with that of libxsmm. As such libxsmm is *highly* recommended.

#### 4.1.2 AVX-512

When running on an AVX-512 capable CPU Clang and GCC will, by default, only make use of 256-bit vectors. Given that the kernels in PyFR benefit meaningfully from longer vectors it is desirable to override this behaviour. This can be accomplished through the `cflags` key as:

```
[backend-openmp]
cflags = -mprefer-vector-width=512
```

#### 4.1.3 Cores vs. threads

PyFR does not typically derive any benefit from SMT. As such the number of OpenMP threads should be chosen to be equal to the number of physical cores.

#### 4.1.4 MPI processes vs. OpenMP threads

When using the OpenMP backend it is recommended to employ *one MPI rank per NUMA zone*. For most systems each socket represents its own NUMA zone. Thus, on a two socket system it is suggested to run PyFR with two MPI ranks, with each process being bound to a single socket. The specifics of how to accomplish this depend on both the job scheduler and MPI distribution.

## 4.2 CUDA Backend

### 4.2.1 CUDA-aware MPI

PyFR is capable of taking advantage of CUDA-aware MPI. This enables CUDA device pointers to be directly passed to MPI routines. Under the right circumstances this can result in improved performance for simulations which are near the strong scaling limit. Assuming `mpi4py` has been built against an MPI distribution which is CUDA-aware this functionality can be enabled through the `mpi-type` key as:

```
[backend-cuda]
mpi-type = cuda-aware
```

## 4.3 Partitioning

### 4.3.1 METIS vs SCOTCH

The partitioning module in PyFR includes support for both METIS and SCOTCH. Both usually result in high-quality decompositions. However, for long running simulations on complex geometries it may be worth partitioning a grid with both and observing which decomposition performs best.

### 4.3.2 Mixed grids

When running PyFR in parallel on mixed element grids it is necessary to take some additional care when partitioning the grid. A good domain decomposition is one where each partition contains the same amount of computational work. For grids with a single element type the amount of computational work is very well approximated by the number of elements assigned to a partition. Thus the goal is simply to ensure that all of the partitions have roughly the same number of elements. However, when considering mixed grids this relationship begins to break down since the computational cost of one element type can be appreciably more than that of another.

Thus in order to obtain a good decomposition it is necessary to assign a weight to each type of element in the domain. Element types which are more computationally intensive should be assigned a larger weight than those that are less intensive. Unfortunately, the relative cost of different element types depends on a variety of factors, including:

- The polynomial order.
- If anti-aliasing is enabled in the simulation, and if so, to what extent.
- The hardware which the simulation will be run on.

Weights can be specified when partitioning the mesh as `-e shape:weight`. For example, if on a particular system a quadrilateral is found to be 50% more expensive than a triangle this can be specified as:

```
pyfr partition -e quad:3 -e tri:2 ...
```

If precise profiling data is not available regarding the performance of each element type in a given configuration a helpful rule of thumb is to under-weight the dominant element type in the domain. For example, if a domain is 90% tetrahedra and 10% prisms then, absent any additional information about the relative performance of tetrahedra and prisms, a safe choice is to assume the prisms are appreciably *more* expensive than the tetrahedra.

## 4.4 Parallel I/O

PyFR incorporates support for parallel file I/O via HDF5 and will use it automatically where available. However, for this work several prerequisites must be satisfied:

- HDF5 must be explicitly compiled with support for parallel I/O.
- The mpi4py Python module *must* be compiled against the same MPI distribution as HDF5. A version mismatch here can result in subtle and difficult to diagnose errors.
- The h5py Python module *must* be built with support for parallel I/O.

After completing this process it is highly recommended to verify everything is working by trying the `h5py parallel hdf5` example.

## 4.5 Start-up Time

The start-up time required by PyFR can be reduced by ensuring that Python is compiled from source with profile guided optimisations (PGO) which can be enabled by passing `--enable-optimizations` to the `configure` script.

It is also important that NumPy be configured to use an optimized BLAS/LAPACK distribution. Further details can be found in the [NumPy building from source](#) guide.

If the point sampler plugin is being employed with a large number of sample points it is further recommended to install SciPy.



## EXAMPLES

PyFR includes several test cases to showcase the functionality of the solver. It is important to note, however, that these examples are all relatively small 2D simulations and, as such, are *not* suitable for scalability or performance studies.

### 5.1 Euler Equations

#### 5.1.1 2D Euler Vortex

Proceed with the following steps to run a parallel 2D Euler vortex simulation on a structured mesh:

1. Create a working directory called `euler_vortex_2d/`
2. Copy the configuration file `PyFR/examples/euler_vortex_2d/euler_vortex_2d.ini` into `euler_vortex_2d/`
3. Copy the `Gmsh` file `PyFR/examples/euler_vortex_2d/euler_vortex_2d.msh` into `euler_vortex_2d/`
4. Run `pyfr` to convert the `Gmsh` mesh file into a PyFR mesh file called `euler_vortex_2d.pyfrm`:

```
pyfr import euler_vortex_2d.msh euler_vortex_2d.pyfrm
```

5. Run `pyfr` to partition the PyFR mesh file into two pieces:

```
pyfr partition 2 euler_vortex_2d.pyfrm .
```

6. Run `pyfr` to solve the Euler equations on the mesh, generating a series of PyFR solution files called `euler_vortex_2d*.pyfrs`:

```
mpiexec -n 2 pyfr run -b cuda -p euler_vortex_2d.pyfrm euler_vortex_2d.ini
```

7. Run `pyfr` on the solution file `euler_vortex_2d-100.0.pyfrs` converting it into an unstructured VTK file called `euler_vortex_2d-100.0.vtu`:

```
pyfr export euler_vortex_2d.pyfrm euler_vortex_2d-100.0.pyfrs euler_vortex_2d-100.0.  
→vtu
```

8. Visualise the unstructured VTK file in Paraview

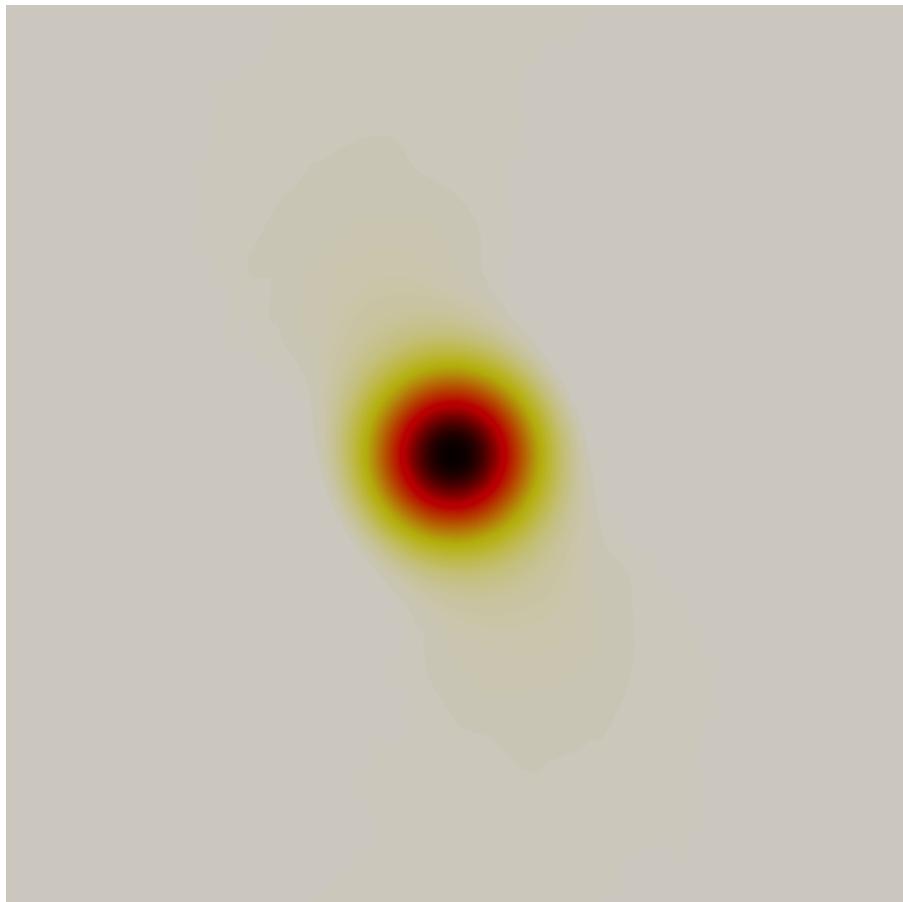


Fig. 1: Colour map of density distribution at 100 time units.

## 5.2 Compressible Navier–Stokes Equations

### 5.2.1 2D Couette Flow

Proceed with the following steps to run a serial 2D Couette flow simulation on a mixed unstructured mesh:

1. Create a working directory called `couette_flow_2d/`
2. Copy the configuration file `PyFR/examples/couette_flow_2d/couette_flow_2d.ini` into `couette_flow_2d/`
3. Copy the `Gmsh` mesh file `PyFR/examples/couette_flow_2d/couette_flow_2d.msh` into `couette_flow_2d/`
4. Run `pyfr` to convert the `Gmsh` mesh file into a PyFR mesh file called `couette_flow_2d.pyfrm`:

```
pyfr import couette_flow_2d.msh couette_flow_2d.pyfrm
```

5. Run `pyfr` to solve the Navier-Stokes equations on the mesh, generating a series of PyFR solution files called `couette_flow_2d-* .pyfrs`:

```
pyfr run -b cuda -p couette_flow_2d.pyfrm couette_flow_2d.ini
```

6. Run `pyfr` on the solution file `couette_flow_2d-040.pyfrs` converting it into an unstructured VTK file called `couette_flow_2d-040.vtu`:

```
pyfr export couette_flow_2d.pyfrm couette_flow_2d-040.pyfrs couette_flow_2d-040.vtu
```

7. Visualise the unstructured VTK file in Paraview

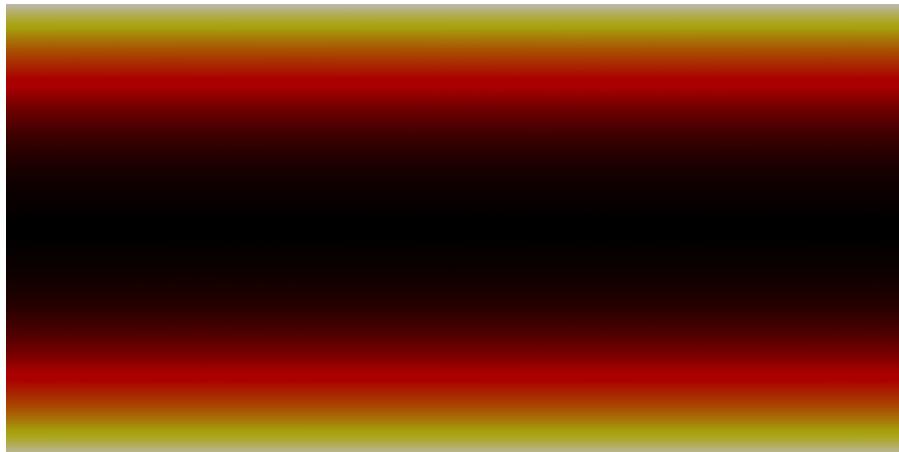


Fig. 2: Colour map of steady-state density distribution.

## 5.3 Incompressible Navier–Stokes Equations

### 5.3.1 2D Incompressible Cylinder Flow

Proceed with the following steps to run a serial 2D incompressible cylinder flow simulation on a mixed unstructured mesh:

1. Create a working directory called `inc_cylinder_2d/`
2. Copy the configuration file `PyFR/examples/inc_cylinder_2d/inc_cylinder_2d.ini` into `inc_cylinder_2d/`
3. Copy the compressed `Gmsh` mesh file `PyFR/examples/inc_cylinder_2d/inc_cylinder_2d.msh.gz` into `inc_cylinder_2d/`
4. Unzip the file and run `pyfr` to convert the `Gmsh` mesh file into a PyFR mesh file called `inc_cylinder_2d.pyfrm`:

```
zcat inc_cylinder_2d.msh.gz | pyfr import -tgms - inc_cylinder_2d.pyfrm
```

5. Run `pyfr` to solve the incompressible Navier-Stokes equations on the mesh, generating a series of PyFR solution files called `inc_cylinder_2d-* .pyfrs`:

```
pyfr run -b cuda -p inc_cylinder_2d.pyfrm inc_cylinder_2d.ini
```

6. Run `pyfr` on the solution file `inc_cylinder_2d-75.00.pyfrs` converting it into an unstructured VTK file called `inc_cylinder_2d-75.00.vtu`:

```
pyfr export inc_cylinder_2d.pyfrm inc_cylinder_2d-75.00.pyfrs inc_cylinder_2d-75.00.  
vtu
```

7. Visualise the unstructured VTK file in Paraview

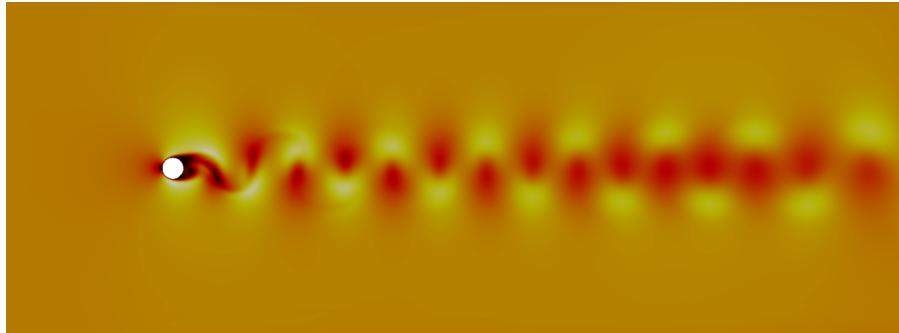


Fig. 3: Colour map of velocity magnitude distribution at 75 time units.

---

**CHAPTER  
SIX**

---

**INDICES AND TABLES**

- genindex
- modindex
- search



# INDEX

## Symbols

\_accept\_step() (pyfr.integrators.dual.phys.controllers.DualNoneController method), 32  
\_accept\_step() (pyfr.integrators.std.controllers.StdNoneController method), 31  
\_accept\_step() (pyfr.integrators.std.controllers.StdPIController method), 32  
\_add() (pyfr.integrators.dual.pseudo.pseudocontrollers.DualNoneController method), 33  
\_add() (pyfr.integrators.dual.pseudo.pseudocontrollers.DualHPSseudoController method), 34  
\_add() (pyfr.integrators.dual.pseudo.pseudosteppers.DualDenseRKStepper method), 42  
\_add() (pyfr.integrators.dual.pseudo.pseudosteppers.DualEulerPseudoStepper method), 44  
\_addv() (pyfr.integrators.dual.pseudo.pseudosteppers.DualRK34PseudoStepper method), 45  
\_addv() (pyfr.integrators.dual.pseudo.pseudosteppers.DualRK45PseudoStepper method), 46  
\_addv() (pyfr.integrators.dual.pseudo.pseudosteppers.DualRK4PseudoStepper method), 43  
\_addv() (pyfr.integrators.dual.pseudo.pseudosteppers.DualTVDRK3PseudoStepper method), 44  
\_addv() (pyfr.integrators.std.controllers.StdNoneController method), 31  
\_addv() (pyfr.integrators.std.controllers.StdPIController method), 32  
\_addv() (pyfr.integrators.std.steppers.StdEulerStepper method), 35  
\_addv() (pyfr.integrators.std.steppers.StdRK34Stepper method), 37  
\_addv() (pyfr.integrators.std.steppers.StdRK45Stepper method), 38  
\_addv() (pyfr.integrators.std.steppers.StdRK4Stepper method), 36  
\_addv() (pyfr.integrators.std.steppers.StdTVDRK3Stepper method), 38  
\_build\_arglst() (pyfr.backends.cuda.provider.CUDAPointwiseKernelProvider method), 62  
\_build\_arglst() (pyfr.backends.hip.provider.HIPPointwiseKernelProvider method), 63  
\_build\_arglst() (pyfr.backends.opencl.provider.OpenCLPointwiseKernelProvider method), 63  
\_build\_arglst() (pyfr.backends.openmp.provider.OpenMPPointwiseKernelProvider method), 63  
\_build\_function() (pyfr.backends.openmp.provider.OpenMPPointwiseKernelProvider method), 63  
\_build\_kernel() (pyfr.backends.cuda.provider.CUDAPointwiseKernelProvider method), 62  
\_build\_kernel() (pyfr.backends.hip.provider.HIPPointwiseKernelProvider method), 63  
\_build\_kernel() (pyfr.backends.opencl.provider.OpenCLPointwiseKernelProvider method), 63  
\_build\_kernel() (pyfr.backends.openmp.provider.OpenMPPointwiseKernelProvider method), 63

```
    method), 63
_build_library() (pyfr.backends.openmp.provider.OpenMPProvider._build_library()
    method), 63
_build_program() (pyfr.backends.opencl.provider.OpenCLPointwiseKernelProvider
    method), 63
_check_abort() (pyfr.integrators.dual.phys.controllers.DualNoneController._check_abort()
    method), 32
_check_abort() (pyfr.integrators.dual.phys.steppers.DualBDF2Stepper._check_abort()
    method), 39
_check_abort() (pyfr.integrators.dual.phys.steppers.DualBDF3Stepper._check_abort()
    method), 40
_check_abort() (pyfr.integrators.dual.phys.steppers.DualBackwardEulerStepper._check_abort()
    method), 40
_check_abort() (pyfr.integrators.std.controllers.StdNoneController._check_abort()
    method), 31
_check_abort() (pyfr.integrators.std.controllers.StdPICController._check_abort()
    method), 32
_check_abort() (pyfr.integrators.std.steppers.StdEulerStepper._check_abort()
    method), 35
_check_abort() (pyfr.integrators.std.steppers.StdRK34Stepper._check_abort()
    method), 37
_check_abort() (pyfr.integrators.std.steppers.StdRK45Stepper._check_abort()
    method), 38
_check_abort() (pyfr.integrators.std.steppers.StdRK4Stepper._check_abort()
    method), 36
_check_abort() (pyfr.integrators.std.steppers.StdTVDRK3Stepper._check_abort()
    method), 38
_const_mat() (pyfr.solvers.aceuler.inters.AC EulerIntInters._const_mat()
    method), 56
_const_mat() (pyfr.solvers.aceuler.inters.AC EulerMPIInters._const_mat()
    method), 56
_const_mat() (pyfr.solvers.acnavstokes.inters.AC NavierStokesInters._const_mat()
    method), 57
_const_mat() (pyfr.solvers.acnavstokes.inters.AC NavierStokesMPIInters._const_mat()
    method), 57
_const_mat() (pyfr.solvers.euler.inters.EulerIntInters._const_mat()
    method), 58
_const_mat() (pyfr.solvers.euler.inters.EulerMPIInters._const_mat()
    method), 58
_const_mat() (pyfr.solvers.navstokes.inters.NavierStokesInters._const_mat()
    method), 58
_const_mat() (pyfr.solvers.navstokes.inters.NavierStokesMPIInters._const_mat()
    method), 59
_deref_arg_array_1d() (pyfr.backends.cuda.generator.CUDAKernelGenerator._deref_arg_array_1d()
    method), 64
_deref_arg_array_1d() (pyfr.backends.hip.generator.HIPKernelGenerator._deref_arg_array_1d()
    method), 65
_deref_arg_array_1d() (pyfr.backends.opencl.generator.OpenCLKernelGenerator._deref_arg_array_1d()
    method), 65
_deref_arg_array_1d() (pyfr.backends.openmp.generator.OpenMPKernelGenerator._deref_arg_array_1d()
    method), 65
_deref_arg_view() (pyfr.backends.hip.generator.HIPKernelGenerator._deref_arg_view()
    method), 65
_deref_arg_view() (pyfr.backends.opencl.generator.OpenCLKernelGenerator._deref_arg_view()
    method), 65
_deref_arg_view() (pyfr.backends.openmp.generator.OpenMPKernelGenerator._deref_arg_view()
    method), 65
_deref_arg_view() (pyfr.backends.cuda.generator.CUDAKernelGenerator._deref_arg_view()
    method), 64
_deref_arg_view() (pyfr.backends.hip.generator.HIPKernelGenerator._deref_arg_view()
    method), 65
_deref_arg_view() (pyfr.backends.opencl.generator.OpenCLKernelGenerator._deref_arg_view()
    method), 65
_deref_arg_view() (pyfr.backends.openmp.generator.OpenMPKernelGenerator._deref_arg_view()
    method), 65
_deref_arg_view() (pyfr.backends.cuda.generator.CUDAKernelGenerator._deref_arg_view()
    method), 64
_deref_arg_view() (pyfr.backends.hip.generator.HIPKernelGenerator._deref_arg_view()
    method), 65
_deref_arg_view() (pyfr.backends.opencl.generator.OpenCLKernelGenerator._deref_arg_view()
    method), 65
_deref_arg_view() (pyfr.backends.openmp.generator.OpenMPKernelGenerator._deref_arg_view()
    method), 65
_deref_arg_view() (pyfr.backends.cuda.generator.CUDAKernelGenerator._deref_arg_view()
    method), 64
_deref_arg_view() (pyfr.backends.hip.generator.HIPKernelGenerator._deref_arg_view()
    method), 65
_deref_arg_view() (pyfr.backends.opencl.generator.OpenCLKernelGenerator._deref_arg_view()
    method), 65
_deref_arg_view() (pyfr.backends.openmp.generator.OpenMPKernelGenerator._deref_arg_view()
    method), 65
_deref_arg_view() (pyfr.backends.cuda.generator.CUDAKernelGenerator._deref_arg_view()
    method), 64
_deref_arg_view() (pyfr.backends.hip.generator.HIPKernelGenerator._deref_arg_view()
    method), 65
_deref_arg_view() (pyfr.backends.opencl.generator.OpenCLKernelGenerator._deref_arg_view()
    method), 65
_deref_arg_view() (pyfr.backends.openmp.generator.OpenMPKernelGenerator._deref_arg_view()
    method), 65
_gen_norm_fpts() (pyfr.solvers.aceuler.elements.AC EulerElements._gen_norm_fpts()
    method), 58
```



```
(pyfr.solvers.euler.inters.EulerMPIInters _get_reduction_kerns()
method), 58 (pyfr.integrators.std.controllers.StdNoneController
_get_perm_for_view() method), 31
(pyfr.solvers.navstokes.inters.NavierStokesIntInters _get_reduction_kerns()
method), 58 (pyfr.integrators.std.controllers.StdPICController
_get_perm_for_view() method), 32
(pyfr.solvers.navstokes.inters.NavierStokesMPIInter _get_reduction_kerns()
method), 59 (pyfr.integrators.std.steppers.StdEulerStepper
_get_plugins() (pyfr.integrators.dual.phys.controllers.DualNoneCont _get_reduction_kerns()
method), 35
method), 32
_get_reduction_kerns()
_get_plugins() (pyfr.integrators.dual.phys.steppers.DualBDF2Step (pyfr.integrators.std.steppers.StdRK34Stepper
method), 39 _get_reduction_kerns()
method), 37
_get_plugins() (pyfr.integrators.dual.phys.steppers.DualBDF3Step _get_reduction_kerns()
method), 40 (pyfr.integrators.std.steppers.StdRK45Stepper
_get_plugins() (pyfr.integrators.dual.phys.steppers.DualBackward _get_reduction_kerns()
method), 41 (pyfr.integrators.std.steppers.StdRK4Stepper
method), 31
_get_plugins() (pyfr.integrators.std.controllers.StdNoneController (pyfr.integrators.std.steppers.StdTVDRK3Stepper
method), 32 _get_reduction_kerns()
method), 36
_get_plugins() (pyfr.integrators.std.steppers.StdEulerStepper _get_rkvdh2_kerns()
method), 35
_get_plugins() (pyfr.integrators.std.steppers.StdRK34Stepper (pyfr.integrators.std.steppers.StdRK34Stepper
method), 37 _get_reduction_kerns()
method), 37
_get_plugins() (pyfr.integrators.std.steppers.StdRK45Stepper _get_rkvdh2_kerns()
method), 38
_get_plugins() (pyfr.integrators.std.steppers.StdRK4Stepper (pyfr.integrators.std.steppers.StdRK45Stepper
method), 36 _get_rkvdh2pseudo_kerns()
method), 38
_get_plugins() (pyfr.integrators.std.steppers.StdTVDRK3Stepper (pyfr.integrators.dual.pseudo.pseudosteppers.DualRK34PseudoSt
method), 39 _get_reduction_kerns()
method), 45
_get_reduction_kerns() (pyfr.integrators.dual.pseudo.pseudocontrollers.DualNoneP _get_rkvdh2pseudo_kerns()
method), 34 (pyfr.integrators.dual.pseudo.pseudocontrollers.DualPIPSe _instantiate_kernel()
method), 34 (pyfr.integrators.dual.pseudo.pseudocontrollers.DualPIPSe _instantiate_kernel()
method), 62
_get_reduction_kerns() (pyfr.integrators.dual.pseudo.pseudosteppers.DualDenseRK _pyfr.integrators.dual.pseudo.pseudosteppers.DualRK45PseudoSt
method), 42 _get_reduction_kerns()
method), 63
_get_reduction_kerns() (pyfr.integrators.dual.pseudo.pseudosteppers.DualEulerP _pyfr.integrators.dual.pseudo.pseudosteppers.DualRK45PseudoSt
method), 45 _get_reduction_kerns()
method), 63
_get_reduction_kerns() (pyfr.integrators.dual.pseudo.pseudosteppers.DualRK34PSe _pyfr.integrators.dual.pseudo.pseudosteppers.DualRK45PseudoSt
method), 45 _get_reduction_kerns()
method), 63
_get_reduction_kerns() (pyfr.integrators.dual.pseudo.pseudosteppers.DualRK45PSe _load_bc_inters() (pyfr.solvers.aceuler.system.ACEulerSystem
method), 46 _load_bc_inters()
method), 48
_get_reduction_kerns() (pyfr.integrators.dual.pseudo.pseudosteppers.DualRK45PSe _load_bc_inters() (pyfr.solvers.acnavstokes.system.ACNavierStokesSyst
method), 43 _load_bc_inters()
method), 49
_get_reduction_kerns() (pyfr.integrators.dual.pseudo.pseudosteppers.DualRK45PSe _load_bc_inters() (pyfr.solvers.navstokes.system.NavierStokesSystem
method), 44 _load_eles()
method), 48
```

`_method), 48`  
`_load_eles() (pyfr.solvers.acnavstokes.system.ACNavierStokesElements, 48`  
`_method), 48`  
`_load_eles() (pyfr.solvers.euler.system.EulerSystem, 49`  
`_method), 49`  
`_load_eles() (pyfr.solvers.navstokes.system.NavierStokesElements, 50`  
`_method), 50`  
`_load_int_inters() (pyfr.solvers.aceuler.system.ACEulerElements, 48`  
`_method), 48`  
`_load_int_inters() (pyfr.solvers.acnavstokes.system.ACNavierStokesElements, 48`  
`_method), 48`  
`_load_int_inters() (pyfr.solvers.euler.system.EulerSystem, 49`  
`_method), 49`  
`_load_int_inters() (pyfr.solvers.navstokes.system.NavierStokesElements, 50`  
`_method), 50`  
`_load_mpi_inters() (pyfr.solvers.aceuler.system.ACEulerElements, 48`  
`_method), 48`  
`_load_mpi_inters() (pyfr.solvers.acnavstokes.system.ACNavierStokesElements, 49`  
`_method), 49`  
`_load_mpi_inters() (pyfr.solvers.euler.system.EulerSystem, 49`  
`_method), 49`  
`_load_mpi_inters() (pyfr.solvers.navstokes.system.NavierStokesElements, 50`  
`_method), 50`  
`_mag_pnorm_fpts(pyfr.solvers.aceuler.elements.ACEulerElements, 51`  
`_property), 51`  
`_mag_pnorm_fpts(pyfr.solvers.acnavstokes.elements.ACNavierStokesElements, 52`  
`_property), 52`  
`_mag_pnorm_fpts(pyfr.solvers.euler.elements.EulerElements, 53`  
`_property), 53`  
`_mag_pnorm_fpts(pyfr.solvers.navstokes.elements.NavierStokesElements, 54`  
`_property), 54`  
`_malloc_impl() (pyfr.backends.cuda.base.CUDABackend, 60`  
`_method), 60`  
`_malloc_impl() (pyfr.backends.hip.base.HIPBackend, 60`  
`_method), 60`  
`_malloc_impl() (pyfr.backends.opencl.base.OpenCLBackend, 61`  
`_method), 61`  
`_malloc_impl() (pyfr.backends.openmp.base.OpenMPBackend, 61`  
`_method), 61`  
`_mesh_regions(pyfr.solvers.aceuler.elements.ACEulerElements, 51`  
`_property), 51`  
`_mesh_regions(pyfr.solvers.acnavstokes.elements.ACNavierStokesElements, 52`  
`_property), 52`  
`_mesh_regions(pyfr.solvers.euler.elements.EulerElements_reject_step(), 53`  
`_property), 53`  
`_mesh_regions(pyfr.solvers.navstokes.elements.NavierStokesElements, 54`  
`_property), 54`  
`_nonce_seq(pyfr.solvers.aceuler.system.ACEulerSystem, 48`  
`_attribute), 48`  
`_nonce_seq(pyfr.solvers.acnavstokes.system.ACNavierStokesElements, 49`  
`_attribute), 49`  
`_nonce_seq(pyfr.solvers.euler.system.EulerSystem, 49`  
`_attribute), 49`  
`_nonce_seq(pyfr.solvers.navstokes.system.NavierStokesElements, 50`  
`_attribute), 50`  
`_norm_pnorm_fpts(pyfr.solvers.aceuler.elements.ACEulerElements, 51`  
`_property), 51`  
`_norm_pnorm_fpts(pyfr.solvers.acnavstokes.elements.ACNavierStokesElements, 52`  
`_property), 52`  
`_norm_pnorm_fpts(pyfr.solvers.euler.elements.EulerElements, 53`  
`_property), 53`  
`_norm_pnorm_fpts(pyfr.solvers.navstokes.elements.NavierStokesElements, 54`  
`_property), 54`  
`_norm_pnorm_fpts(pyfr.solvers.acnavstokes.system.ACNavierStokesElements, 54`  
`_property), 54`  
`_norm_pnorm_fpts(pyfr.solvers.euler.system.EulerElements, 54`  
`_property), 54`  
`_norm_pnorm_fpts(pyfr.solvers.navstokes.elements.NavierStokesElements, 54`  
`_property), 54`  
`_ploc_in_src_exprs(pyfr.solvers.acnavstokes.elements.ACNavierStokesElements, 52`  
`_property), 52`  
`_ploc_in_src_exprs(pyfr.solvers.euler.elements.EulerElements, 53`  
`_property), 53`  
`_ploc_in_src_exprs(pyfr.solvers.navstokes.elements.NavierStokesElements, 54`  
`_property), 54`  
`_pseudo_stepper_regidx(pyfr.integrators.dual.pseudo.pseudocontrollers.DualNonePseudoStepper, 34`  
`_method), 34`  
`_pseudo_stepper_regidx(pyfr.integrators.dual.pseudo.pseudocontrollers.DualPIPseudoController, 34`  
`_method), 34`  
`_pseudo_stepper_regidx(pyfr.integrators.dual.pseudo.pseudosteppers.DualDenseRKPseudoStepper, 42`  
`_method), 42`  
`_pseudo_stepper_regidx(pyfr.integrators.dual.pseudo.pseudosteppers.DualEulerPseudoStepper, 45`  
`_method), 45`  
`_pseudo_stepper_regidx(pyfr.integrators.dual.pseudo.pseudosteppers.DualRK34PseudoStepper, 54`  
`_method), 54`  
`_pseudo_stepper_regidx(pyfr.integrators.dual.pseudo.pseudosteppers.DualRK45PseudoStepper, 60`  
`_method), 60`  
`_pseudo_stepper_regidx(pyfr.integrators.dual.pseudo.pseudosteppers.DualRK4PseudoStepper, 61`  
`_method), 61`  
`_pseudo_stepper_regidx(pyfr.integrators.dual.pseudo.pseudosteppers.DualTVDRK3PseudoStepper, 63`  
`_method), 63`  
`_reject_step(pyfr.integrators.std.controllers.StdNoneController, 31`  
`_method), 31`  
`_reject_step(pyfr.integrators.std.controllers.StdPICController, 32`  
`_method), 32`  
`_render_body(pyfr.backends.cuda.generator.CUDAKernelGenerator, 64`  
`_method), 64`  
`_render_body(pyfr.backends.hip.generator.HIPKernelGenerator, 65`  
`_method), 65`  
`_render_body(pyfr.backends.opencl.generator.OpenCLKernelGenerator, 65`  
`_method), 65`  
`_render_body(pyfr.backends.openmp.generator.OpenMPKernelGenerator, 65`  
`_method), 65`

```
        method), 65
    _render_kernel() (pyfr.backends.cuda.provider.CUDAPointwiseKernelView(pyfr.solvers.euler.inters.EulerIntInters
        method), 62
    _render_kernel() (pyfr.backends.hip.provider.HIPPointwiseKernelView(pyfr.solvers.euler.inters.EulerMPIInters
        method), 63
    _render_kernel() (pyfr.backends.opencl.provider.OpenCLPointwiseKernelView(pyfr.solvers.navstokes.inters.NavierStokesIntInters
        method), 63
    _render_kernel() (pyfr.backends.openmp.provider.OpenMPPointwiseKernelView(pyfr.solvers.navstokes.inters.NavierStokesMPIInters
        method), 63
    _render_spec() (pyfr.backends.cuda.generator.CUDAKernelScratchbufs(pyfr.solvers.aceuler.elements.ACEulerElements
        method), 64
    _render_spec() (pyfr.backends.hip.generator.HIPKernelGeneratorScratchbufs(pyfr.solvers.acnavstokes.elements.ACNavierStokesElements
        method), 65
    _render_spec() (pyfr.backends.opencl.generator.OpenCLKernelScratchbufs(pyfr.solvers.euler.elements.EulerElements
        method), 65
    _resid() (pyfr.integrators.dual.pseudo.pseudocontrollers.DualNLPExtIntegrator(pyfr.solvers.navstokes.elements.NavierStokesElements
        method), 34
    _resid() (pyfr.integrators.dual.pseudo.pseudocontrollers.DualNLPExtIntegrator(pyfr.solvers.aceuler.inters.ACEulerIntInters
        method), 34
    _rhs_with_dts() (pyfr.integrators.dual.pseudo.pseudostep.set_DxExternalRK(pyfr.solvers.aceuler.inters.ACEulerMPIInters
        method), 42
    _rhs_with_dts() (pyfr.integrators.dual.pseudo.pseudostep.set_DxExternalRK(pyfr.solvers.acnavstokes.inters.ACNavierStokesIntInters
        method), 45
    _rhs_with_dts() (pyfr.integrators.dual.pseudo.pseudostep.set_DxExternalRK(pyfr.solvers.acnavstokes.inters.ACNavierStokesMPIInters
        method), 45
    _rhs_with_dts() (pyfr.integrators.dual.pseudo.pseudostep.set_DxExternalRK(pyfr.solvers.euler.inters.EulerIntInters
        method), 46
    _rhs_with_dts() (pyfr.integrators.dual.pseudo.pseudostep.set_DxExternalRK(pyfr.solvers.euler.inters.EulerMPIInters
        method), 43
    _rhs_with_dts() (pyfr.integrators.dual.pseudo.pseudostep.set_DxExternalRK(pyfr.solvers.navstokes.inters.NavierStokesIntInters
        method), 44
    _scal_view() (pyfr.solvers.aceuler.inters.ACEulerIntInters.set_external() (pyfr.solvers.navstokes.inters.NavierStokesMPIInters
        method), 56
    _scal_view() (pyfr.solvers.aceuler.inters.ACEulerMPIInters.slice_mat() (pyfr.solvers.aceuler.elements.ACEulerElements
        method), 57
    _scal_view() (pyfr.solvers.acnavstokes.inters.ACNavierStokesMPIInters.slice_mat() (pyfr.solvers.acnavstokes.elements.ACNavierStokesElements
        method), 57
    _scal_view() (pyfr.solvers.acnavstokes.inters.ACNavierStokesMPIInters.slice_mat() (pyfr.solvers.euler.elements.EulerElements
        method), 57
    _scal_view() (pyfr.solvers.euler.inters.EulerIntInters._slice_mat() (pyfr.solvers.navstokes.elements.NavierStokesElements
        method), 58
    _scal_view() (pyfr.solvers.euler.inters.EulerMPIInters._smats_djacs_mppts(pyfr.solvers.aceuler.elements.ACEulerElements
        method), 58
    _scal_view() (pyfr.solvers.navstokes.inters.NavierStokesIntInters._smats_djacs_mppts(pyfr.solvers.acnavstokes.elements.ACNavierStokesElements
        method), 58
    _scal_view() (pyfr.solvers.navstokes.inters.NavierStokesMPIInters._smats_djacs_mppts(pyfr.solvers.euler.elements.EulerElements
        method), 59
    _scal_xchg_view() (pyfr.solvers.aceuler.inters.ACEulerIntInters._smats_djacs_mppts(pyfr.solvers.navstokes.elements.NavierStokesElements
        method), 56
    _scal_xchg_view() (pyfr.solvers.aceuler.inters.ACEulerMPIInters._smats_djacs_mppts(pyfr.solvers.aceuler.elements.ACEulerElements
        method), 57
    _scal_xchg_view() (pyfr.solvers.acnavstokes.inters.ACNavierStokesMPIInters._smats_djacs_mppts(pyfr.solvers.acnavstokes.elements.ACNavierStokesElements
        method), 57
    _scal_xchg_view() (pyfr.solvers.acnavstokes.inters.ACNavierStokesMPIInters._smats_djacs_mppts(pyfr.solvers.euler.elements.EulerElements
        method), 57
```



```
        method), 58  
_vect_xchg_view() (pyfr.solvers.euler.inters.EulerMPIInters      ACEulerIntInters (class in pyfr.solvers.aceuler.inters),  
        method), 58 56  
_vect_xchg_view() (pyfr.solvers.navstokes.inters.NavierStokesIntInters  ACEulerMPIInters (class in pyfr.solvers.aceuler.inters),  
        method), 58 48  
_vect_xchg_view() (pyfr.solvers.navstokes.inters.NavierStokesIntInters  ACEulerSystem (class in pyfr.solvers.aceuler.system), 48  
        method), 59 ACNavierStokesElements (class in  
_view() (pyfr.solvers.aceuler.inters.AC EulerIntInters  pyfr.solvers.acnavstokes.elements), 52  
        method), 56 ACNavierStokesIntInters (class in  
_view() (pyfr.solvers.aceuler.inters.AC EulerMPIInters  pyfr.solvers.acnavstokes.inters), 57  
        method), 57 ACNavierStokesMPIInters (class in  
_view() (pyfr.solvers.acnavstokes.inters.AC NavierStokesIntInters  pyfr.solvers.acnavstokes.inters), 57  
        method), 57 ACNavierStokesSystem (class in  
_view() (pyfr.solvers.acnavstokes.inters.AC NavierStokesMPIInters  pyfr.solvers.acnavstokes.system), 48  
        method), 57 Advance_to() (pyfr.integrators.dual.phys.controllers.DualNoneController  
_view() (pyfr.solvers.euler.inters.EulerIntInters  method), 32  
        method), 58 advance_to() (pyfr.integrators.dual.phys.steppers.DualBackwardEulerStepper  
_view() (pyfr.solvers.euler.inters.EulerMPIInters  method), 41  
        method), 58 advance_to() (pyfr.integrators.dual.phys.steppers.DualBDF2Stepper  
_view() (pyfr.solvers.navstokes.inters.NavierStokesIntInters  method), 39  
        method), 58 advance_to() (pyfr.integrators.dual.phys.steppers.DualBDF3Stepper  
_view() (pyfr.solvers.navstokes.inters.NavierStokesMPIInters  method), 40  
        method), 59 advance_to() (pyfr.integrators.std.controllers.StdNoneController  
_xchg_view() (pyfr.solvers.aceuler.inters.AC EulerIntInters  method), 31  
        method), 56 advance_to() (pyfr.integrators.std.controllers.StdPIController  
_xchg_view() (pyfr.solvers.aceuler.inters.AC EulerMPIInters  method), 32  
        method), 57 advance_to() (pyfr.integrators.std.steppers.StdEulerStepper  
_xchg_view() (pyfr.solvers.acnavstokes.inters.AC NavierStokesIntInters  method), 35  
        method), 57 advance_to() (pyfr.integrators.std.steppers.StdRK34Stepper  
_xchg_view() (pyfr.solvers.acnavstokes.inters.AC NavierStokesMPIInters  method), 37  
        method), 57 advance_to() (pyfr.integrators.std.steppers.StdRK45Stepper  
_xchg_view() (pyfr.solvers.euler.inters.EulerIntInters  method), 38  
        method), 58 advance_to() (pyfr.integrators.std.steppers.StdRK4Stepper  
_xchg_view() (pyfr.solvers.euler.inters.EulerMPIInters  method), 36  
        method), 58 advance_to() (pyfr.integrators.std.steppers.StdTVDRK3Stepper  
_xchg_view() (pyfr.solvers.navstokes.inters.NavierStokesIntInters  method), 39  
        method), 58 alias() (pyfr.backends.cuda.base.CUDABackend  
_xchg_view() (pyfr.solvers.navstokes.inters.NavierStokesMPIInters  method), 60  
        method), 59 alias() (pyfr.backends.hip.base.HIPBackend method),  
A 60  
alias() (pyfr.backends.opencl.base.OpenCLBackend  
a (pyfr.integrators.dual.phys.steppers.DualBackwardEulerStepper  method), 61  
        attribute), 41 alias() (pyfr.backends.openmp.base.OpenMPBackend  
a (pyfr.integrators.dual.pseudo.pseudosteppers.DualRK34PseudoSteppe  method), 61  
        attribute), 45 argspec() (pyfr.backends.cuda.generator.CUDAKernelGenerator  
a (pyfr.integrators.dual.pseudo.pseudosteppers.DualRK45PseudoSteppe  method), 64  
        attribute), 46 argspec() (pyfr.backends.hip.generator.HIPKernelGenerator  
a (pyfr.integrators.std.steppers.StdRK34Stepper  method), 65  
        attribute), 37 at- argspec() (pyfr.backends.opencl.generator.OpenCLKernelGenerator  
a (pyfr.integrators.std.steppers.StdRK45Stepper  method), 65  
        attribute), 38 at- argspec() (pyfr.backends.openmp.generator.OpenMPKernelGenerator  
ACEulerElements (class  method), 65  
        in aux_nregs (pyfr.integrators.dual.pseudo.pseudocontrollers.DualNonePseudo  
pyfr.solvers.aceuler.elements), 51 attribute), 34  
aux_nregs (pyfr.integrators.dual.pseudo.pseudocontrollers.DualPIPpseudo
```

*attribute), 34*  
**aux\_nregs** (*pyfr.integrators.dual.pseudo.pseudosteppers.DualRK34Stepper*.*attribute*), 42  
**aux\_nregs** (*pyfr.integrators.dual.pseudo.pseudosteppers.DualEulerStepper*.*attribute*), 45  
**aux\_nregs** (*pyfr.integrators.dual.pseudo.pseudosteppers.DualRK34Stepper*.*attribute*), 46  
**aux\_nregs** (*pyfr.integrators.dual.pseudo.pseudosteppers.DualRK45Stepper*.*attribute*), 46  
**aux\_nregs** (*pyfr.integrators.dual.pseudo.pseudosteppers.DualRK45Stepper*.*attribute*), 43  
**aux\_nregs** (*pyfr.integrators.dual.pseudo.pseudosteppers.DualTVDRK3Stepper*.*attribute*), 44

*method), 33*  
*method), 41*  
*method), 39*  
*method), 40*  
*method), 31*  
*method), 32*  
*method), 35*  
*call\_plugin\_dt()* (*pyfr.integrators.std.steppers.StdRK34Stepper*.*method*), 37

**B**

**b** (*pyfr.integrators.dual.pseudo.pseudosteppers.DualRK34PseudoStepper*.*attribute*), 46  
**b** (*pyfr.integrators.dual.pseudo.pseudosteppers.DualRK45PseudoStepper*.*attribute*), 46  
**b** (*pyfr.integrators.std.steppers.StdRK34Stepper*.*attribute*), 37  
**b** (*pyfr.integrators.std.steppers.StdRK45Stepper*.*attribute*), 38  
**bbcinterscls** (*pyfr.solvers.aceuler.system.ACEulerSystem*.*cfgmeta* (*pyfr.integrators.dual.phys.steppers.DualBackwardEulerStepper*.*attribute*), 48  
**bbcinterscls** (*pyfr.solvers.acnavstokes.system.ACNavierStokesSystem*.*cfgmeta* (*pyfr.integrators.dual.phys.steppers.DualBDF2Stepper*.*attribute*), 49  
**bbcinterscls** (*pyfr.solvers.euler.system.EulerSystem*.*attribute*), 49  
**bbcinterscls** (*pyfr.solvers.navstokes.system.NavierStokesSystem*.*cfgmeta* (*pyfr.integrators.std.controllers.StdNoneController*.*attribute*), 50  
**bhat** (*pyfr.integrators.dual.pseudo.pseudosteppers.DualRK34Stepper*.*attribute*), 46  
**bhat** (*pyfr.integrators.dual.pseudo.pseudosteppers.DualRK45Stepper*.*attribute*), 46  
**bhat** (*pyfr.integrators.std.steppers.StdRK34Stepper*.*attribute*), 37  
**bhat** (*pyfr.integrators.std.steppers.StdRK45Stepper*.*attribute*), 38  
**block1d** (*pyfr.backends.hip.generator.HIPKernelGenerator*.*cfgmeta* (*pyfr.integrators.std.steppers.StdRK4Stepper*.*attribute*), 65  
**block2d** (*pyfr.backends.hip.generator.HIPKernelGenerator*.*cfgmeta* (*pyfr.integrators.std.steppers.StdTVDRK3Stepper*.*attribute*), 65  
**blocks** (*pyfr.backends.cuda.base.CUDABackend*.*attribute*), 60  
**blocks** (*pyfr.backends.hip.base.HIPBackend*.*attribute*), 60  
**blocks** (*pyfr.backends.opencl.base.OpenCLBackend*.*attribute*), 61  
**blocks** (*pyfr.backends.openmp.base.OpenMPBackend*.*attribute*), 61

*cfgmeta* (*pyfr.integrators.std.controllers.StdNoneController*.*attribute*), 32  
*cfgmeta* (*pyfr.integrators.dual.phys.steppers.DualBDF3Stepper*.*attribute*), 40  
*cfgmeta* (*pyfr.integrators.std.steppers.StdRK34Stepper*.*attribute*), 37  
*cfgmeta* (*pyfr.integrators.std.steppers.StdRK45Stepper*.*attribute*), 38  
*cfgmeta* (*pyfr.integrators.std.steppers.StdRK4Stepper*.*attribute*), 36  
*cfgmeta* (*pyfr.integrators.std.steppers.StdTVDRK3Stepper*.*attribute*), 39  
*collect\_stats()* (*pyfr.integrators.dual.phys.controllers.DualNoneController*.*method*), 33  
*collect\_stats()* (*pyfr.integrators.dual.phys.steppers.DualBackwardEulerStepper*.*method*), 41  
*collect\_stats()* (*pyfr.integrators.dual.phys.steppers.DualBDF2Stepper*.*method*), 39  
*collect\_stats()* (*pyfr.integrators.dual.phys.steppers.DualBDF3Stepper*.*method*), 40  
*collect\_stats()* (*pyfr.integrators.dual.pseudo.pseudosteppers.DualDenseStepper*.*method*), 43

**C**

**call\_plugin\_dt()** (*pyfr.integrators.dual.phys.controllers.DualNoneController*.*method*)

`method), 45`  
`collect_stats() (pyfr.integrators.dual.pseudo.pseudosteppecontontrollecontrs.DualNoneController method), 46`  
`collect_stats() (pyfr.integrators.dual.pseudo.pseudosteppecontontrollecontrs.StdNoneController method), 47`  
`collect_stats() (pyfr.integrators.dual.pseudo.pseudosteppecontontrollecontrs.StdPIController method), 43`  
`collect_stats() (pyfr.integrators.dual.pseudo.pseudosteppecontontrollecontrs.StdRK3Stepper method), 44`  
`collect_stats() (pyfr.integrators.std.controllers.StdNoneController property), 31`  
`method), 31`  
`controller_needs_errest`  
`collect_stats() (pyfr.integrators.std.controllers.StdPIController method), 32`  
`property), 32`  
`collect_stats() (pyfr.integrators.std.steppers.StdEulerStepper controller_needs_errest method), 36`  
`collect_stats() (pyfr.integrators.std.steppers.StdRK34Stepper controller_needs_errest method), 37`  
`collect_stats() (pyfr.integrators.std.steppers.StdRK45Stepper controller_needs_errest method), 38`  
`property), 37`  
`collect_stats() (pyfr.integrators.std.steppers.StdRK45Stepper controller_needs_errest method), 36`  
`property), 36`  
`collect_stats() (pyfr.integrators.std.steppers.StdTVDRK3Stepper controller_needs_errest method), 39`  
`property), 39`  
`commit() (pyfr.backends.cuda.base.CUDABackend controller_needs_errest method), 60`  
`commit() (pyfr.backends.hip.base.HIPBackend controller_needs_errest method), 60`  
`commit() (pyfr.backends.opencl.base.OpenCLBackend controller_needs_errest method), 61`  
`commit() (pyfr.backends.openmp.base.OpenMPBackend controller_needs_errest method), 61`  
`compute_grads() (pyfr.solvers.aceuler.system.ACEulerSystem convvarmap (pyfr.solvers.euler.elements.EulerElements method), 48`  
`attribute), 52`  
`compute_grads() (pyfr.solvers.acnavstokes.system.ACNavierStokesSystem convvarmap (pyfr.solvers.navstokes.elements.NavierStokesElements method), 49`  
`attribute), 53`  
`compute_grads() (pyfr.solvers.euler.system.EulerSystem convvarmap (pyfr.solvers.navstokes.elements.NavierStokesElements method), 49`  
`attribute), 55`  
`compute_grads() (pyfr.solvers.navstokes.system.NavierStokesSystem convmon() (pyfr.integrators.dual.pseudo.pseudocontrollers.DualNonePseudo method), 34`  
`method), 50`  
`convmon() (pyfr.integrators.dual.pseudo.pseudocontrollers.DualPIPpseudo method), 34`  
`con_to_pri() (pyfr.solvers.aceuler.elements.ACEulerElements CUDABackend (class in pyfr.backends.cuda.base), 60`  
`static method), 51`  
`con_to_pri() (pyfr.solvers.acnavstokes.elements.ACNavierStokesSystem CUDAKernelGenerator (class in pyfr.backends.cuda.generator), 64`  
`static method), 52`  
`con_to_pri() (pyfr.solvers.euler.elements.EulerElements CUDAPointwiseKernelProvider (class in pyfr.backends.cuda.provider), 62`  
`static method), 53`  
`con_to_pri() (pyfr.solvers.navstokes.elements.NavierStokesElements D`  
`static method), 55`  
`const_matrix() (pyfr.backends.cuda.base.CUDABackend discard_oldest_source() method), 60`  
`const_matrix() (pyfr.backends.hip.base.HIPBackend discard_oldest_source() method), 34`  
`const_matrix() (pyfr.backends.opencl.base.OpenCLBackend discard_oldest_source() method), 61`  
`const_matrix() (pyfr.backends.openmp.base.OpenMPBackend disregard_oldest_source() method), 34`

D

(`pyfr.integrators.dual.pseudo.pseudostepers.DualRK34PseudoStepper`) (class in `pyfr.integrators.dual.pseudo.pseudostepers`), 44

`discard_oldest_source()` (`pyfr.integrators.dual.pseudo.pseudostepers.DualEulerPseudoStepper`)

`discard_oldest_source()` (`pyfr.integrators.dual.pseudo.pseudostepers.DualRK34PseudoStepper`)

`discard_oldest_source()` (`pyfr.integrators.dual.pseudo.pseudostepers.DualRK45PseudoStepper`)

`discard_oldest_source()` (`pyfr.integrators.dual.pseudo.pseudostepers.DualRK45PseudoStepper`)

`discard_oldest_source()` (`pyfr.integrators.dual.pseudo.pseudostepers.DualRK4PseudoStepper`)

`discard_oldest_source()` (`pyfr.integrators.dual.pseudo.pseudostepers.DualRK4PseudoStepper`)

`DualBackwardEulerStepper` (class in `pyfr.integrators.dual.phys.steppers`), 40

`DualBDF2Stepper` (class in `pyfr.integrators.dual.phys.steppers`), 39

`DualBDF3Stepper` (class in `pyfr.integrators.dual.phys.steppers`), 40

`dualcoeffs` (`pyfr.solvers.aceuler.elements.ACEulerElements`)

`dualcoeffs` (`pyfr.solvers.acnavstokes.elements.ACNavierStokesElements`)

`dualcoeffs` (`pyfr.solvers.euler.elements.EulerElements`)

`dualcoeffs` (`pyfr.solvers.navstokes.elements.NavierStokesElements`)

`DualDenseRKPseudoStepper` (class in `pyfr.integrators.dual.pseudo.pseudostepers`), 42

`DualEulerPseudoStepper` (class in `pyfr.integrators.dual.pseudo.pseudostepers`), 44

`DualNoneController` (class in `pyfr.integrators.dual.phys.controllers`), 32

`DualNonePseudoController` (class in `pyfr.integrators.dual.pseudo.pseudocontrollers`), 33

`DualPIPpseudoController` (class in `pyfr.integrators.dual.pseudo.pseudocontrollers`), 34

`DualRK34PseudoStepper` (class in `pyfr.integrators.dual.pseudo.pseudostepers`), 45

`DualRK45PseudoStepper` (class in `pyfr.integrators.dual.pseudo.pseudostepers`), 46

`DualRK4PseudoStepper` (class in `pyfr.integrators.dual.pseudo.pseudostepers`), 43

`elementscls` (`pyfr.solvers.aceuler.system.ACEulerSystem` attribute), 48

`elementscls` (`pyfr.solvers.acnavstokes.system.ACNavierStokesSystem` attribute), 49

`elementscls` (`pyfr.solvers.euler.system.EulerSystem` attribute), 49

`elementscls` (`pyfr.solvers.navstokes.system.NavierStokesSystem` attribute), 49

`EulerElements` (class in `pyfr.solvers.euler.elements`), 53

`EulerIntInters` (class in `pyfr.solvers.euler.inters`), 57

`EulerMPIInters` (class in `pyfr.solvers.euler.inters`), 58

`EulerSystem` (class in `pyfr.solvers.euler.system`), 49

`filt()` (`pyfr.solvers.aceuler.system.ACEulerSystem` method), 48

`filt()` (`pyfr.solvers.acnavstokes.system.ACNavierStokesSystem` method), 49

`filt()` (`pyfr.solvers.euler.system.EulerSystem` method), 49

`filt()` (`pyfr.solvers.navstokes.system.NavierStokesSystem` method), 50

`formulation` (`pyfr.integrators.dual.phys.controllers.DualNoneController` attribute), 33

`formulation` (`pyfr.integrators.dual.phys.steppers.DualBackwardEulerStepper` attribute), 41

`formulation` (`pyfr.integrators.dual.phys.steppers.DualBDF2Stepper` attribute), 39

`formulation` (`pyfr.integrators.dual.phys.steppers.DualBDF3Stepper` attribute), 40

`formulation` (`pyfr.integrators.dual.pseudo.pseudocontrollers.DualNonePseudoController` attribute), 34

`formulation` (`pyfr.integrators.dual.pseudo.pseudocontrollers.DualPIPpseudoController` attribute), 34

`formulation` (`pyfr.integrators.dual.pseudo.pseudostepers.DualDenseRKPseudoStepper` attribute), 43

`formulation` (`pyfr.integrators.dual.pseudo.pseudostepers.DualEulerPseudoStepper` attribute), 45

`formulation` (`pyfr.integrators.dual.pseudo.pseudostepers.DualRK34PseudoStepper` attribute), 46

```
formulation(pyfr.integrators.dual.pseudo.pseudosteppers.DualRK45RshdyStepper
           attribute), 47
formulation(pyfr.integrators.dual.pseudo.pseudosteppers.DualRK45RshdyStepper
           attribute), 43
formulation(pyfr.integrators.dual.pseudo.pseudosteppers.DualRK45RshdyStepper
           attribute), 44
formulation(pyfr.integrators.std.controllers.StdNoneController
           attribute), 31
formulation(pyfr.integrators.std.controllers.StdPIController
           attribute), 32
formulation(pyfr.integrators.std.steppers.StdEulerStepper
           attribute), 36
formulation(pyfr.integrators.std.steppers.StdRK34Stepper
           attribute), 37
formulation(pyfr.integrators.std.steppers.StdRK45Stepper
           attribute), 38
formulation(pyfr.integrators.std.steppers.StdRK4Stepper
           attribute), 36
formulation(pyfr.integrators.std.steppers.StdTVDRK3Stepper
           attribute), 39
formulations(pyfr.solvers.aceuler.elements.ACEulerElements
             attribute), 51
formulations(pyfr.solvers.acnavstokes.elements.ACNavierStokesElements
             attribute), 52
formulations(pyfr.solvers.euler.elements.EulerElements
             attribute), 53
formulations(pyfr.solvers.navstokes.elements.NavierStokesElements
             attribute), 54
fsal(pyfr.integrators.dual.phys.steppers.DualBackwardEulerStepper
      attribute), 41
G
get_artvisc_fpts_for_inter()
  (pyfr.solvers.acnavstokes.elements.ACNavierStokesElements
   method), 52
get_artvisc_fpts_for_inter()
  (pyfr.solvers.navstokes.elements.NavierStokesElements
   method), 55
get_mag_pnorms()
  (pyfr.solvers.aceuler.elements.ACEulerElements static method), 33
  (pyfr.solvers.acnavstokes.elements.ACNavierStokesElements
   method), 51
get_mag_pnorms()
  (pyfr.solvers.acnavstokes.elements.ACNavierStokesElements
   method), 52
get_mag_pnorms()
  (pyfr.solvers.euler.elements.EulerElements
   method), 53
get_mag_pnorms()
  (pyfr.solvers.navstokes.elements.NavierStokesElements
   method), 39
  (pyfr.integrators.dual.phys.steppers.DualBackwardEulerStepper
   static method), 41
get_mag_pnorms()
  (pyfr.solvers.euler.elements.EulerElements
   method), 53
get_mag_pnorms()
  (pyfr.solvers.navstokes.elements.NavierStokesElements
   method), 55
get_mag_pnorms_for_inter()
  (pyfr.solvers.aceuler.elements.ACEulerElements
   method), 51
get_mag_pnorms_for_inter()
  (pyfr.solvers.acnavstokes.elements.ACNavierStokesElements
   static method), 32
  (pyfr.solvers.acnavstokes.elements.ACNavierStokesElements
   method), 52
get_mag_pnorms_for_inter()
  (pyfr.solvers.euler.elements.EulerElements
   method), 32
```

```

get_plugin_data_prefix()
    (pyfr.integrators.std.steppers.StdEulerStepper
     static method), 36
get_plugin_data_prefix()
    (pyfr.integrators.std.steppers.StdRK34Stepper
     static method), 37
get_plugin_data_prefix()
    (pyfr.integrators.std.steppers.StdRK45Stepper
     static method), 38
get_plugin_data_prefix()
    (pyfr.integrators.std.steppers.StdRK4Stepper
     static method), 36
get_plugin_data_prefix()
    (pyfr.integrators.std.steppers.StdTVDRK3Stepper
     static method), 39
get_scal_fpts_for_inter()
    (pyfr.solvers.aceuler.elements.ACEulerElements
     method), 51
get_scal_fpts_for_inter()
    (pyfr.solvers.acnavstokes.elements.ACNavierStokesElements
     method), 52
get_scal_fpts_for_inter()
    (pyfr.solvers.euler.elements.EulerElements
     method), 54
get_scal_fpts_for_inter()
    (pyfr.solvers.navstokes.elements.NavierStokesElements
     method), 55
get_vect_fpts_for_inter()
    (pyfr.solvers.aceuler.elements.ACEulerElements
     method), 51
get_vect_fpts_for_inter()
    (pyfr.solvers.acnavstokes.elements.ACNavierStokesElements
     method), 52
get_vect_fpts_for_inter()
    (pyfr.solvers.euler.elements.EulerElements
     method), 54
get_vect_fpts_for_inter()
    (pyfr.solvers.navstokes.elements.NavierStokesElements
     method), 55
grad_con_to_pri()
    (pyfr.solvers.acnavstokes.elements.ACNavierStokesElements
     static method), 52
grad_con_to_pri()
    (pyfr.solvers.navstokes.elements.NavierStokesElements
     static method), 55
grad_soln(pyfr.integrators.dual.phys.controllers.DualNoneController
          property), 33
grad_soln(pyfr.integrators.dual.phys.steppers.DualBackwardEulerStepper
          property), 41
grad_soln(pyfr.integrators.dual.phys.steppers.DualBDF2Stepper
          property), 39
grad_soln(pyfr.integrators.dual.phys.steppers.DualBDF3Stepper
          property), 40
grad_soln(pyfr.integrators.std.controllers.StdNoneController
          property), 32
grad_soln(pyfr.integrators.std.controllers.StdPICController
          property), 32
grad_soln(pyfr.integrators.std.steppers.StdEulerStepper
          property), 36
grad_soln(pyfr.integrators.std.steppers.StdRK34Stepper
          property), 37
grad_soln(pyfr.integrators.std.steppers.StdRK45Stepper
          property), 38
grad_soln(pyfr.integrators.std.steppers.StdRK4Stepper
          property), 36
grad_soln(pyfr.integrators.std.steppers.StdTVDRK3Stepper
          property), 39

H
HIPBackend (class in pyfr.backends.hip.base), 60
HIPKernelGenerator (class in pyfr.backends.hip.generator), 65
HIPPointwiseKernelProvider (class in pyfr.backends.hip.provider), 63

I
init_stage() (pyfr.integrators.dual.pseudo.pseudocontrollers.DualNonePseudoController
               method), 34
init_stage() (pyfr.integrators.dual.pseudo.pseudocontrollers.DualPIPse
               method), 34
init_stage() (pyfr.integrators.dual.pseudo.pseudosteppers.DualDenseRK3Pse
               method), 43
init_stage() (pyfr.integrators.dual.pseudo.pseudosteppers.DualEulerPse
               method), 45
init_stage() (pyfr.integrators.dual.pseudo.pseudosteppers.DualRK34Pse
               method), 46
init_stage() (pyfr.integrators.dual.pseudo.pseudosteppers.DualRK45Pse
               method), 47
init_stage() (pyfr.integrators.dual.pseudo.pseudosteppers.DualRK4Pse
               method), 43
init_stage() (pyfr.integrators.dual.pseudo.pseudosteppers.DualTVDRK3Pse
               method), 44
intinterscls (pyfr.solvers.aceuler.system.ACEulerSystem
               attribute), 48
intinterscls (pyfr.solvers.acnavstokes.system.ACNavierStokesSystem
               attribute), 49
intinterscls (pyfr.solvers.euler.system.EulerSystem attribute), 49
intinterscls (pyfr.solvers.navstokes.system.NavierStokesSystem
               attribute), 50

K
kernel() (pyfr.backends.cuda.base.CUDABackend
            method), 60
kernel() (pyfr.backends.hip.base.HIPBackend method),
            60
kernel() (pyfr.backends.opencl.base.OpenCLBackend
            method), 61
kernel() (pyfr.backends.openmp.base.OpenMPBackend
            method), 61

```

kernel\_generator\_cls  
attribute), 62

kernel\_generator\_cls  
attribute), 63

kernel\_generator\_cls  
attribute), 63

kernel\_generator\_cls  
attribute), 63

L

ldim\_size() (pyfr.backends.cuda.generator.CUDAKernelGenerator method), 64

ldim\_size() (pyfr.backends.hip.generator.HIPKernelGenerator method), 65

ldim\_size() (pyfr.backends.opencl.generator.OpenCLKernelGenerator method), 65

ldim\_size() (pyfr.backends.openmp.generator.OpenMPKernelGenerator method), 65

localerrest() (pyfr.integrators.dual.pseudo.pseudocontraint method), 35

lookup (pyfr.backends.cuda.base.CUDABackend property), 60

lookup (pyfr.backends.hip.base.HIPBackend property), 60

lookup (pyfr.backends.opencl.base.OpenCLBackend property), 61

lookup (pyfr.backends.openmp.base.OpenMPBackend property), 61

M

malloc() (pyfr.backends.cuda.base.CUDABackend method), 60

malloc() (pyfr.backends.hip.base.HIPBackend method), 60

malloc() (pyfr.backends.opencl.base.OpenCLBackend method), 61

malloc() (pyfr.backends.openmp.base.OpenMPBackend method), 61

matrix() (pyfr.backends.cuda.base.CUDABackend method), 60

matrix() (pyfr.backends.hip.base.HIPBackend method), 60

matrix() (pyfr.backends.opencl.base.OpenCLBackend method), 61

matrix() (pyfr.backends.openmp.base.OpenMPBackend method), 61

matrix\_slice() (pyfr.backends.cuda.base.CUDABackend method), 60

matrix\_slice() (pyfr.backends.hip.base.HIPBackend method), 60

matrix\_slice() (pyfr.backends.opencl.base.OpenCLBackend method), 61

matrix\_slice() (pyfr.backends.openmp.base.OpenMPBackend method), 61

matrix\_slice() (pyfr.backends.openmp.base.OpenMPBackend method), 61

MPI\_TAG (pyfr.solvers.aceuler.inters.AC EulerMPIInters attribute), 56

MPI\_TAG (pyfr.solvers.acnavstokes.inters.AC NavierStokesMPIInters attribute), 57

MPI\_TAG (pyfr.solvers.euler.inters.EulerMPIInters attribute), 58

MPI\_TAG (pyfr.solvers.navstokes.inters.NavierStokesMPIInters attribute), 59

mpiinterscls (pyfr.solvers.aceuler.system.AC EulerSystem attribute), 48

mpiinterscls (pyfr.solvers.acnavstokes.system.AC NavierStokesSystem attribute), 49

mpiinterscls (pyfr.solvers.euler.system.EulerSystem attribute), 49

mpiinterscls (pyfr.solvers.navstokes.system.NavierStokesSystem attribute), 50

N

name (pyfr.backends.cuda.base.CUDABackend attribute), 60

name (pyfr.backends.hip.base.HIPBackend attribute), 60

name (pyfr.backends.opencl.base.OpenCLBackend attribute), 61

name (pyfr.backends.openmp.base.OpenMPBackend attribute), 61

name (pyfr.solvers.aceuler.system.AC EulerSystem attribute), 48

name (pyfr.solvers.acnavstokes.system.AC NavierStokesSystem attribute), 49

name (pyfr.solvers.euler.system.EulerSystem attribute), 49

NavierStokesElements (class pyfr.solvers.navstokes.elements), 54

NavierStokesIntInters (class pyfr.solvers.navstokes.inters), 58

NavierStokesMPIInters (class pyfr.solvers.navstokes.inters), 59

NavierStokesSystem (class pyfr.solvers.navstokes.system), 50

needs\_ldim() (pyfr.backends.cuda.generator.CUDAKernelGenerator method), 64

needs\_ldim() (pyfr.backends.hip.generator.HIPKernelGenerator method), 65

needs\_ldim() (pyfr.backends.opencl.generator.OpenCLKernelGenerator method), 65

needs\_ldim() (pyfr.backends.openmp.generator.OpenMPKernelGenerator method), 65

nstages (pyfr.integrators.dual.phys.steppers.DualBackwardEulerStepper attribute), 41

**nstages** (`pyfr.integrators.dual.phys.steppers.DualBDF2Stepper` attribute), 39  
**obtain\_solution()** (`pyfr.integrators.dual.pseudo.pseudosteppers.DualTVDRK3PseudoStepper` method), 44  
**nstages** (`pyfr.integrators.dual.phys.steppers.DualBDF3Stepper` attribute), 40  
**openCLBackend** (class in `pyfr.backends.opencl.base`), 61  
**OpenCLKernelGenerator** (class in `pyfr.backends.opencl.generator`), 65  
**OpenCLPointwiseKernelProvider** (class in `pyfr.backends.opencl.provider`), 63  
**nsteps** (`pyfr.integrators.dual.phys.controllers.DualNoneController` property), 33  
**nsteps** (`pyfr.integrators.dual.phys.steppers.DualBackwardEulerStepper` property), 41  
**OpenMPBackend** (class in `pyfr.backends.openmp.base`), 61  
**nsteps** (`pyfr.integrators.dual.phys.steppers.DualBDF2Stepper` property), 39  
**nsteps** (`pyfr.integrators.dual.phys.steppers.DualBDF3Stepper` property), 40  
**nsteps** (`pyfr.integrators.std.controllers.StdNoneController` property), 32  
**nsteps** (`pyfr.integrators.std.controllers.StdPICController` property), 32  
**nsteps** (`pyfr.integrators.std.steppers.StdEulerStepper` property), 36  
**nsteps** (`pyfr.integrators.std.steppers.StdRK34Stepper` property), 37  
**nsteps** (`pyfr.integrators.std.steppers.StdRK45Stepper` property), 38  
**nsteps** (`pyfr.integrators.std.steppers.StdRK4Stepper` property), 36  
**nsteps** (`pyfr.integrators.std.steppers.StdTVDRK3Stepper` property), 39  
**ntotiters** (`pyfr.integrators.dual.pseudo.pseudosteppers.DualDenseRK3PseudoStepper` property), 43  
**ntotiters** (`pyfr.integrators.dual.pseudo.pseudosteppers.DualEulerPseudoStepper` property), 45  
**ntotiters** (`pyfr.integrators.dual.pseudo.pseudosteppers.DualRK34PseudoStepper` property), 46  
**ntotiters** (`pyfr.integrators.dual.pseudo.pseudosteppers.DualRK45PseudoStepper` property), 47  
**ntotiters** (`pyfr.integrators.dual.pseudo.pseudosteppers.DualRK4PseudoStepper` property), 43  
**ntotiters** (`pyfr.integrators.dual.pseudo.pseudosteppers.DualTVDRK3PseudoStepper` property), 44  
**P**  
**ploc\_at()** (`pyfr.solvers.aceuler.elements.ACEulerElements` method), 51  
**ploc\_at()** (`pyfr.solvers.acnavstokes.elements.ACNavierStokesElements` method), 53  
**ploc\_at()** (`pyfr.solvers.euler.elements.EulerElements` method), 54  
**ploc\_at()** (`pyfr.solvers.navstokes.elements.NavierStokesElements` method), 55  
**ploc\_at\_np()** (`pyfr.solvers.aceuler.elements.ACEulerElements` method), 53  
**ploc\_at\_np()** (`pyfr.solvers.acnavstokes.elements.ACNavierStokesElements` method), 55  
**ploc\_at\_np()** (`pyfr.solvers.euler.elements.EulerElements` method), 54  
**ploc\_at\_np()** (`pyfr.solvers.navstokes.elements.NavierStokesElements` method), 55  
**plocfpts** (`pyfr.solvers.aceuler.elements.ACEulerElements`)  
**obtain\_solution()** (`pyfr.integrators.dual.pseudo.pseudocontrollers.DualNonePseudoController` method), 34  
**plocfpts** (`pyfr.solvers.acnavstokes.elements.ACNavierStokesElements`)  
**obtain\_solution()** (`pyfr.integrators.dual.pseudo.pseudocontrollers.DualHPSpseudoController` method), 35  
**plocfpts** (`pyfr.solvers.euler.elements.EulerElements`)  
**obtain\_solution()** (`pyfr.integrators.dual.pseudo.pseudosteppers.DualRK34PseudoStepper` method), 43  
**plocfpts** (`pyfr.solvers.navstokes.elements.NavierStokesElements`)  
**obtain\_solution()** (`pyfr.integrators.dual.pseudo.pseudosteppers.DualRK45PseudoStepper` method), 45  
**plocfpts** (`pyfr.solvers.euler.elements.EulerElements`)  
**obtain\_solution()** (`pyfr.integrators.dual.pseudo.pseudosteppers.DualRK4PseudoStepper` method), 46  
**prepare()** (`pyfr.solvers.aceuler.inters.ACEulerIntInters`)  
**obtain\_solution()** (`pyfr.integrators.dual.pseudo.pseudosteppers.DualRK45PseudoStepper` method), 47  
**prepare()** (`pyfr.solvers.aceuler.inters.ACEulerMPIInters`)  
**obtain\_solution()** (`pyfr.integrators.dual.pseudo.pseudosteppers.DualRK4PseudoStepper` method), 43  
**O**  
**obtain\_solution()** (`pyfr.integrators.dual.pseudo.pseudocontrollers.DualNonePseudoController` method), 34  
**plocfpts** (`pyfr.solvers.acnavstokes.elements.ACNavierStokesElements`)  
**obtain\_solution()** (`pyfr.integrators.dual.pseudo.pseudocontrollers.DualHPSpseudoController` method), 35  
**plocfpts** (`pyfr.solvers.euler.elements.EulerElements`)  
**obtain\_solution()** (`pyfr.integrators.dual.pseudo.pseudosteppers.DualRK34PseudoStepper` method), 43  
**plocfpts** (`pyfr.solvers.navstokes.elements.NavierStokesElements`)  
**obtain\_solution()** (`pyfr.integrators.dual.pseudo.pseudosteppers.DualRK45PseudoStepper` method), 45  
**prepare()** (`pyfr.solvers.aceuler.inters.ACEulerIntInters`)  
**obtain\_solution()** (`pyfr.integrators.dual.pseudo.pseudosteppers.DualRK45PseudoStepper` method), 46  
**prepare()** (`pyfr.solvers.aceuler.inters.ACEulerMPIInters`)  
**obtain\_solution()** (`pyfr.integrators.dual.pseudo.pseudosteppers.DualRK4PseudoStepper` method), 47  
**prepare()** (`pyfr.solvers.acnavstokes.inters.ACNavierStokesIntInters`)  
**obtain\_solution()** (`pyfr.integrators.dual.pseudo.pseudosteppers.DualRK4PseudoStepper` method), 43

```
prepare() (pyfr.solvers.acnavstokes.inters.ACNavierStokesMethodstepper_has_lerrest
          method), 57
prepare() (pyfr.solvers.euler.inters.EulerIntInters
          method), 58
prepare() (pyfr.solvers.euler.inters.EulerMPIInters
          method), 58
prepare() (pyfr.solvers.navstokes.inters.NavierStokesIntIntpseudo_stepper_name
          method), 58
prepare() (pyfr.solvers.navstokes.inters.NavierStokesMPIInters
          method), 59
pri_to_con() (pyfr.solvers.aceuler.elements.ACEulerElements
              static method), 51
pri_to_con() (pyfr.solvers.acnavstokes.elements.ACNavierStokesElements
              static method), 53
pri_to_con() (pyfr.solvers.euler.elements.EulerElements
              static method), 54
pri_to_con() (pyfr.solvers.navstokes.elements.NavierStokesElements
              static method), 55
privarmap (pyfr.solvers.aceuler.elements.ACEulerElementspseudo_stepper_nfevals
            attribute), 51
privarmap (pyfr.solvers.acnavstokes.elements.ACNavierStokesElementsproperty),
            attribute), 53
privarmap (pyfr.solvers.euler.elements.EulerElements
            attribute), 54
privarmap (pyfr.solvers.navstokes.elements.NavierStokesElementspseudo_stepper_nfevals
            attribute), 55
pseudo_advance() (pyfr.integrators.dual.pseudo.pseudocontrollers.DualNonePseudoController
                  method), 34
pseudo_advance() (pyfr.integrators.dual.pseudo.pseudocontrollers.DualPIPSe
                  method), 35
pseudo_controller_name (pyfr.integrators.dual.pseudo.pseudocontrollers.DualNonePseudoController
                        attribute), 34
pseudo_controller_name (pyfr.integrators.dual.pseudo.pseudocontrollers.DualPIPSe
                        attribute), 35
pseudo_controller_needs_lerrest (pyfr.integrators.dual.pseudo.pseudocontrollers.DualNonePseudoController
                                 attribute), 34
pseudo_controller_needs_lerrest (pyfr.integrators.dual.pseudo.pseudocontrollers.DualPIPSe
                                 attribute), 35
pseudo stepper_has_lerrest (pyfr.integrators.dual.pseudo.pseudosteppers.DualEulerPseudoStepper
                           attribute), 45
pseudo stepper_has_lerrest (pyfr.integrators.dual.pseudo.pseudosteppers.DualRK34PseudoStepper
                           attribute), 46
pseudo stepper_has_lerrest (pyfr.integrators.dual.pseudo.pseudosteppers.DualRK45PseudoStepper
                           attribute), 47
pseudo stepper_has_lerrest (pyfr.integrators.dual.pseudo.pseudosteppers.DualRK4PseudoStepper
                           attribute), 43
pseudo stepper_has_lerrest (pyfr.integrators.dual.pseudo.pseudosteppers.DualRK45PseudoStepper
                           attribute), 44
pseudo stepper_has_lerrest (pyfr.integrators.dual.pseudo.pseudosteppers.DualRK4PseudoStepper
                           attribute), 45
```

pseudo stepper\_order `rcpdjac_at_np()` (*pyfr.solvers.navstokes.elements.NavierStokesElements*  
`(pyfr.integrators.dual.pseudo.pseudosteppers.DualRK34PseudoStep)`, 45  
`attribute), 46`

pseudo stepper\_order `register()` (*pyfr.backends.cuda.provider.CUDAPointwiseKernelProvider*  
`(method), 62`

pseudo stepper\_order `RK15PseudoStep()` (*pyfr.backends.hip.provider.HIPPointwiseKernelProvider*  
`(method), 63`

pseudo stepper\_order `register()` (*pyfr.backends.opencl.provider.OpenCLPointwiseKernelProvider*  
`(method), 63`

pseudo stepper\_order `register()` (*pyfr.backends.openmp.provider.OpenMPPointwiseKernelProvider*  
`(method), 63`

pseudo stepper\_order `RK4PseudoStep()` (*pyfr.backends.cuda.generator.CUDAKernelGenerator*  
`(method), 65`

pseudostepinfo (*pyfr.integrators.dual.phys.controllers.DualNoneController*  
`(pyfr.backends.hip.generator.HIPKernelGenerator`, 65  
`property), 33`

pseudostepinfo (*pyfr.integrators.dual.phys.steppers.DualBDF2Stepper*  
`(pyfr.backends.opencl.generator.OpenCLKernelGenerator`, 65  
`property), 41`

pseudostepinfo (*pyfr.integrators.dual.phys.steppers.DualBDF3Stepper*  
`(pyfr.backends.openmp.generator.OpenMPKernelGenerator`, 65  
`property), 39`

pseudostepinfo (*pyfr.integrators.dual.phys.steppers.DualEulerStepper* (*pyfr.solvers.aceuler.system.ACEulerSystem*  
`(method), 48`

**Q**

qpts (*pyfr.solvers.aceuler.elements.ACEulerElements*  
`(property), 51`

qpts (*pyfr.solvers.acnavstokes.elements.ACNavierStokesElements*  
`(property), 53`

qpts (*pyfr.solvers.euler.elements.EulerElements*  
`(property), 54`

qpts (*pyfr.solvers.navstokes.elements.NavierStokesElements*  
`(property), 55`

queue() (*pyfr.backends.cuda.base.CUDABackend*  
`(method), 60`

queue() (*pyfr.backends.hip.base.HIPBackend*  
`(method), 60`

queue() (*pyfr.backends.opencl.base.OpenCLBackend*  
`(method), 61`

queue() (*pyfr.backends.openmp.base.OpenMPBackend*  
`(method), 61`

**R**

rcpdjac\_at() (*pyfr.solvers.aceuler.elements.ACEulerElements*  
`(method), 51`

rcpdjac\_at() (*pyfr.solvers.acnavstokes.elements.ACNavierStokesElements*  
`(method), 53`

rcpdjac\_at() (*pyfr.solvers.euler.elements.EulerElements*  
`(method), 54`

rcpdjac\_at() (*pyfr.solvers.navstokes.elements.NavierStokesElements*  
`(method), 55`

rcpdjac\_at\_np() (*pyfr.solvers.aceuler.elements.ACEulerElements*  
`(method), 51`

rcpdjac\_at\_np() (*pyfr.solvers.acnavstokes.elements.ACNavierStokesElements*  
`(method), 53`

rcpdjac\_at\_np() (*pyfr.solvers.euler.elements.EulerElements*  
`(method), 54`

rhs() (*pyfr.solvers.acnavstokes.system.ACNavierStokesSystem*  
`(method), 49`

rhs() (*pyfr.solvers.euler.system.EulerSystem*  
`(method), 49`

rhs() (*pyfr.solvers.navstokes.system.NavierStokesSystem*  
`(method), 50`

run() (*pyfr.integrators.dual.phys.controllers.DualNoneController*  
`(method), 33`

run() (*pyfr.integrators.dual.phys.steppers.DualBackwardEulerStepper*  
`(method), 41`

run() (*pyfr.integrators.dual.phys.steppers.DualBDF2Stepper*  
`(method), 39`

run() (*pyfr.integrators.dual.phys.steppers.DualBDF3Stepper*  
`(method), 40`

run() (*pyfr.integrators.std.controllers.StdNoneController*  
`(method), 32`

run() (*pyfr.integrators.std.controllers.StdPICController*  
`(method), 32`

run() (*pyfr.integrators.std.steppers.StdEulerStepper*  
`(method), 36`

run() (*pyfr.integrators.std.steppers.StdRK34Stepper*  
`(method), 37`

run() (*pyfr.integrators.std.steppers.StdRK45Stepper*  
`(method), 38`

run() (*pyfr.integrators.std.steppers.StdRK4Stepper*  
`(method), 36`

run() (*pyfr.integrators.std.steppers.StdTVDRK3Stepper*  
`(method), 39`

**S**

set\_backend() (*pyfr.solvers.aceuler.elements.ACEulerElements*  
`(method), 52`

set\_backend() (*pyfr.solvers.acnavstokes.elements.ACNavierStokesElements*  
`(method), 53`

```
set_backend() (pyfr.solvers.euler.elements.EulerElements soln (pyfr.integrators.dual.phys.steppers.DualBDF2Stepper
    method), 54
    property), 39
set_backend() (pyfr.solvers.navstokes.elements.NavierStokesElements soln (pyfr.integrators.dual.phys.steppers.DualBDF3Stepper
    method), 55
    property), 40
set_ics_from_cfg() (pyfr.solvers.aceuler.elements.ACEulerElements soln (pyfr.integrators.std.controllers.StdNoneController
    method), 52
    property), 32
set_ics_from_cfg() (pyfr.solvers.acnavstokes.elements.ACNavierStokesElements soln (pyfr.integrators.std.controllers.StdPICController
    method), 53
    property), 32
set_ics_from_cfg() (pyfr.solvers.euler.elements.EulerElements soln (pyfr.integrators.std.steppers.StdEulerStepper prop-
    erty), 36
    method), 54
set_ics_from_cfg() (pyfr.solvers.navstokes.elements.NavierStokesElements soln (pyfr.integrators.std.steppers.StdRK34Stepper prop-
    erty), 37
    method), 55
set_ics_from_soln() (pyfr.solvers.aceuler.elements.ACEulerElements soln (pyfr.integrators.std.steppers.StdRK45Stepper prop-
    erty), 38
    method), 52
set_ics_from_soln() (pyfr.solvers.acnavstokes.elements.ACNavierStokesElements soln (pyfr.integrators.std.steppers.StdTVDRK3Stepper
    method), 53
    property), 39
set_ics_from_soln() (pyfr.solvers.euler.elements.EulerElements stage_nregs (pyfr.integrators.dual.phys.steppers.DualBackwardEulerStep-
    method), 54
    property), 41
set_ics_from_soln() (pyfr.solvers.navstokes.elements.NavierStokesElements stage_nregs (pyfr.integrators.dual.phys.steppers.DualBDF2Stepper
    method), 55
    attribute), 40
stage_nregs (pyfr.integrators.dual.phys.steppers.DualBDF3Stepper
attribute), 40
shockvar (pyfr.solvers.navstokes.elements.NavierStokesElements StdEulerStepper (class
attribute), 55
    pyfr.integrators.std.steppers), 35
sliceat() (pyfr.solvers.aceuler.elements.ACEulerElements StdNoneController (class
method), 52
    pyfr.integrators.std.controllers), 31
sliceat() (pyfr.solvers.acnavstokes.elements.ACNavierStokesElements StdPICController (class
method), 53
    pyfr.integrators.std.controllers), 32
sliceat() (pyfr.solvers.euler.elements.EulerElements StdRK34Stepper (class in pyfr.integrators.std.steppers),
method), 37
sliceat() (pyfr.solvers.navstokes.elements.NavierStokesElements StdRK45Stepper (class in pyfr.integrators.std.steppers),
method), 37
smat_at() (pyfr.solvers.aceuler.elements.ACEulerElements StdRK4Stepper (class in pyfr.integrators.std.steppers),
method), 52
smat_at() (pyfr.solvers.acnavstokes.elements.ACNavierStokesElements StdTVDRK3Stepper (class
method), 53
    pyfr.integrators.std.steppers), 38
smat_at() (pyfr.solvers.euler.elements.EulerElements step() (pyfr.integrators.dual.phys.controllers.DualNoneController
method), 54
    method), 33
smat_at() (pyfr.solvers.navstokes.elements.NavierStokesElements step() (pyfr.integrators.dual.phys.steppers.DualBackwardEulerStepper
method), 55
    method), 41
smat_at_np() (pyfr.solvers.aceuler.elements.ACEulerElements step() (pyfr.integrators.dual.phys.steppers.DualBDF2Stepper
method), 52
    method), 40
smat_at_np() (pyfr.solvers.acnavstokes.elements.ACNavierStokesElements step() (pyfr.integrators.dual.phys.steppers.DualBDF3Stepper
method), 53
    method), 40
smat_at_np() (pyfr.solvers.euler.elements.EulerElements step() (pyfr.integrators.dual.pseudo.pseudosteppers.DualDenseRKPseudo
method), 54
    method), 43
smat_at_np() (pyfr.solvers.navstokes.elements.NavierStokesElements step() (pyfr.integrators.dual.pseudo.pseudosteppers.DualEulerPseudoStep-
method), 55
    method), 45
soln (pyfr.integrators.dual.phys.controllers.DualNoneController step() (pyfr.integrators.dual.pseudo.pseudosteppers.DualRK34PseudoStep-
property), 33
    method), 46
soln (pyfr.integrators.dual.phys.steppers.DualBackwardEulerStepper step() (pyfr.integrators.dual.pseudo.pseudosteppers.DualRK45PseudoStep-
property), 41
    method), 47
```

```

step() (pyfr.integrators.dual.pseudo.pseudosteppers.DualRK34Stepper
       stepper_orderstepper_nregs(pyfr.integrators.dual.phys.steppers.DualBDF3Stepper
       method), 43
step() (pyfr.integrators.dual.pseudo.pseudosteppers.DualTVDRK3Stepper
       stepper_orderstepper_nregs(pyfr.integrators.std.steppers.StdEulerStepper
       method), 44
step() (pyfr.integrators.std.controllers.StdNoneController stepper_nregs (pyfr.integrators.std.steppers.StdRK34Stepper
       method), 32
step() (pyfr.integrators.std.controllers.StdPIController stepper_nregs (pyfr.integrators.std.steppers.StdRK45Stepper
       method), 32
step() (pyfr.integrators.std.steppers.StdEulerStepper stepper_nregs (pyfr.integrators.std.steppers.StdRK4Stepper
       method), 36
step() (pyfr.integrators.std.steppers.StdRK34Stepper stepper_nregs (pyfr.integrators.std.steppers.StdTVDRK3Stepper
       method), 37
step() (pyfr.integrators.std.steppers.StdRK45Stepper stepper_order (pyfr.integrators.dual.phys.steppers.DualBDF2Stepper
       method), 38
step() (pyfr.integrators.std.steppers.StdRK4Stepper stepper_order (pyfr.integrators.dual.phys.steppers.DualBDF3Stepper
       method), 36
step() (pyfr.integrators.std.steppers.StdTVDRK3Stepper stepper_order (pyfr.integrators.std.steppers.StdEulerStepper
       method), 39
stepper_coeffs() (pyfr.integrators.dual.phys.steppers.DualRK34Stepper
       stepper_orderstepper_nregs(pyfr.integrators.std.steppers.StdRK34Stepper
       method), 41
stepper_coeffs() (pyfr.integrators.dual.phys.steppers.DualRK45Stepper
       stepper_orderstepper_nregs(pyfr.integrators.std.steppers.StdRK45Stepper
       method), 40
stepper_coeffs() (pyfr.integrators.dual.phys.steppers.DualRK4Stepper
       stepper_orderstepper_nregs(pyfr.integrators.std.steppers.StdRK4Stepper
       method), 40
stepper_has_errest(pyfr.integrators.std.steppers.StdEulerStepper
       stepper_orderstepper_nregs(pyfr.integrators.std.steppers.StdTVDRK3Stepper
       attribute), 36
stepper_has_errest(pyfr.integrators.std.steppers.StdRK34Stepper
       stepper_orderstepper_nregs(pyfr.integrators.dual.phys.steppers.DualBDF2Stepper
       attribute), 37
stepper_has_errest(pyfr.integrators.std.steppers.StdRK45Stepper
       stepper_orderstepper_nregs(pyfr.integrators.std.steppers.DualBDF3Stepper
       attribute), 40
stepper_has_errest(pyfr.integrators.std.steppers.StdTVDRK3Stepper
       stepper_orderstepper_nregs(pyfr.integrators.dual.phys.steppers.DualBDF2Stepper
       attribute), 39
stepper_has_errest(pyfr.integrators.std.steppers.StdRK4Stepper
       stepper_orderstepper_nregs(pyfr.integrators.dual.phys.steppers.DualBDF3Stepper
       attribute), 36
stepper_has_errest(pyfr.integrators.std.steppers.StdTVDRK3Stepper
       stepper_orderstepper_nregs(pyfr.integrators.dual.phys.steppers.DualBDF2Stepper
       attribute), 39
stepper_name(pyfr.integrators.dual.phys.steppers.DualBackwardEulerStepper
       stepper_orderstepper_nregs(pyfr.integrators.dual.pseudo.pseudocontrollers.DualNonePseudo
       attribute), 41
store_current_soln()
stepper_name(pyfr.integrators.dual.phys.steppers.DualBDF2Stepper
       stepper_orderstepper_nregs(pyfr.integrators.dual.pseudo.pseudocontrollers.DualPIPpseudoCo
       attribute), 40
store_current_soln()
stepper_name(pyfr.integrators.dual.phys.steppers.DualBDF3Stepper
       stepper_orderstepper_nregs(pyfr.integrators.dual.pseudo.pseudocontrollers.DualDenseRKPseu
       attribute), 40
store_current_soln()
stepper_name(pyfr.integrators.std.steppers.StdEulerStepper
       stepper_orderstepper_nregs(pyfr.integrators.dual.pseudo.pseudocontrollers.DualDenseRKPseu
       attribute), 36
store_current_soln()
stepper_name(pyfr.integrators.std.steppers.StdRK34Stepper
       stepper_orderstepper_nregs(pyfr.integrators.dual.pseudo.pseudosteppers.DualEulerPseudoSt
       attribute), 37
store_current_soln()
stepper_name(pyfr.integrators.std.steppers.StdRK45Stepper
       stepper_orderstepper_nregs(pyfr.integrators.dual.pseudo.pseudosteppers.DualRK34PseudoSt
       attribute), 38
store_current_soln()
stepper_name(pyfr.integrators.std.steppers.StdRK4Stepper
       stepper_orderstepper_nregs(pyfr.integrators.dual.pseudo.pseudocontrollers.DualRK45PseudoSt
       attribute), 37
store_current_soln()
stepper_name(pyfr.integrators.std.steppers.StdTVDRK3Stepper
       stepper_orderstepper_nregs(pyfr.integrators.dual.pseudo.pseudosteppers.DualRK45PseudoSt
       attribute), 39
store_current_soln()
stepper_nregs(pyfr.integrators.dual.phys.steppers.DualBDF2Stepper
       stepper_orderstepper_nregs(pyfr.integrators.dual.pseudo.pseudosteppers.DualRK4PseudoSt
       attribute), 41
store_current_soln()
stepper_nregs(pyfr.integrators.dual.phys.steppers.DualBDF3Stepper
       stepper_orderstepper_nregs(pyfr.integrators.dual.pseudo.pseudosteppers.DualRK4PseudoSt
       attribute), 44
store_current_soln()

```

(*pyfr.integrators.dual.pseudo.pseudostepers.DualNoneController*.**xchg\_matrix\_for\_view()**)  
method), 44

**system**(*pyfr.integrators.dual.phys.controllers.DualNoneController* method), 61  
property), 33

**xchg\_matrix\_for\_view()**

**system**(*pyfr.integrators.dual.phys.steppers.DualBackwardEulerStepper*.**xchg\_view()**) (*pyfr.backends.openmp.base.OpenMPBackend* method), 61

**system**(*pyfr.integrators.dual.phys.steppers.DualBDF2Stepper*.**xchg\_view()**) (*pyfr.backends.cuda.base.CUDABackend* method), 60

**system**(*pyfr.integrators.dual.phys.steppers.DualBDF3Stepper*.**xchg\_view()**) (*pyfr.backends.hip.base.HIPBackend* method), 61

**xchg\_view()** (*pyfr.backends.opencl.base.OpenCLBackend* method), 61

**U**

**upts** (*pyfr.solvers.aceuler.elements.ACEulerElements* **xchg\_view()**) (*pyfr.backends.openmp.base.OpenMPBackend* method), 61  
property), 52

**upts** (*pyfr.solvers.acnavstokes.elements.ACNavierStokesElements* property), 53

**upts** (*pyfr.solvers.euler.elements.EulerElements* property), 54

**upts** (*pyfr.solvers.navstokes.elements.NavierStokesElements* property), 55

**V**

**view()** (*pyfr.backends.cuda.base.CUDABackend* method), 60

**view()** (*pyfr.backends.hip.base.HIPBackend* method), 60

**view()** (*pyfr.backends.opencl.base.OpenCLBackend* method), 61

**view()** (*pyfr.backends.openmp.base.OpenMPBackend* method), 61

**visvarmap** (*pyfr.solvers.aceuler.elements.ACEulerElements* attribute), 52

**visvarmap** (*pyfr.solvers.acnavstokes.elements.ACNavierStokesElements* attribute), 53

**visvarmap** (*pyfr.solvers.euler.elements.EulerElements* attribute), 54

**visvarmap** (*pyfr.solvers.navstokes.elements.NavierStokesElements* attribute), 55

**X**

**xchg\_matrix()** (*pyfr.backends.cuda.base.CUDABackend* method), 60

**xchg\_matrix()** (*pyfr.backends.hip.base.HIPBackend* method), 60

**xchg\_matrix()** (*pyfr.backends.opencl.base.OpenCLBackend* method), 61

**xchg\_matrix()** (*pyfr.backends.openmp.base.OpenMPBackend* method), 61

**xchg\_matrix\_for\_view()**  
(*pyfr.backends.cuda.base.CUDABackend* method), 60

**xchg\_matrix\_for\_view()**  
(*pyfr.backends.hip.base.HIPBackend* method), 60