
PyFR Documentation

Release 1.15.0

Imperial College London

Sep 30, 2022

CONTENTS

1 Installation	3
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
2 User Guide	7
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 Regions	29
2.2.7 Additional Information	30
3 Developer Guide	31
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	36
3.1.4 PseudoStepper	44
3.1.5 System	50
3.1.6 Elements	54
3.1.7 Interfaces	60
3.1.8 Backend	64
3.1.9 Pointwise Kernel Provider	67
3.1.10 Kernel Generator	70
3.2 PyFR-Mako	72
3.2.1 PyFR-Mako Kernels	72
3.2.2 PyFR-Mako Macros	73
3.2.3 Syntax	73
4 Performance Tuning	75
4.1 OpenMP Backend	75
4.1.1 AVX-512	75
4.1.2 Cores vs. threads	75

4.1.3	Loop Scheduling	75
4.1.4	MPI processes vs. OpenMP threads	76
4.2	CUDA Backend	76
4.2.1	CUDA-aware MPI	76
4.3	HIP Backend	76
4.3.1	HIP-aware MPI	76
4.4	Partitioning	76
4.4.1	METIS vs SCOTCH	76
4.4.2	Mixed grids	77
4.4.3	Detecting load imbalances	77
4.5	Scaling	78
4.6	Parallel I/O	78
4.7	Plugins	78
4.8	Start-up Time	78
5	Examples	79
5.1	Euler Equations	79
5.1.1	2D Euler Vortex	79
5.2	Compressible Navier–Stokes Equations	81
5.2.1	2D Couette Flow	81
5.3	Incompressible Navier–Stokes Equations	81
5.3.1	2D Incompressible Cylinder Flow	81
6	Indices and Tables	83
Index		85

PyFR 1.15.0 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 1.15.0 can be installed using `pip` and `virtualenv`, as shown in the quick-start guides below.

1.1.1 macOS

It is assumed that the Xcode Command Line Tools and Homebrew are already installed. Follow the steps below to setup the OpenMP backend on macOS:

1. Install MPI:

```
brew install mpi4py
```

2. Install METIS and set the library path:

```
brew install metis
export PYFR_METIS_LIBRARY_PATH=/opt/homebrew/lib/libmetis.dylib
```

3. Download and install libxsimm and set the library path:

```
git clone git@github.com:libxsimm/libxsimm.git
cd libxsimm
make -j4 STATIC=0 BLAS=0
export PYFR_XSMM_LIBRARY_PATH=`pwd`/lib/libxsimm.dylib
```

4. Make a venv and activate it:

```
python3.10 -m venv pyfr-venv
source pyfr-venv/bin/activate
```

5. Install PyFR:

```
pip install pyfr
```

6. Add the following to your *Configuration File (.ini)*:

```
[backend-openmp]
cc = gcc-12
```

Note the version of the compiler which must support the `openmp` flag. This has been tested on macOS 12.5 with an Apple M1 Max.

1.1.2 Ubuntu

Follow the steps below to setup the OpenMP backend on Ubuntu:

1. Install Python and MPI:

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

2. Install METIS:

```
sudo apt install metis libmetis-dev
```

3. Download and install libxsmm and set the library path:

```
git clone git@github.com:libxsmm/libxsmm.git  
cd libxsmm  
make -j4 STATIC=0 BLAS=0  
export PYFR_XSMM_LIBRARY_PATH=`pwd`/lib/libxsmm.so
```

4. Make a virtualenv and activate it:

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

5. Install PyFR:

```
pip install pyfr
```

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

1. `gimmik` >= 3.0
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` ≥ 11.4

1.2.1.2 HIP Backend

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

1. `ROCM` $\geq 5.2.0$
2. `rocBLAS` $\geq 2.41.0$

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` ≥ 2.1
2. Optionally `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 x86-64 and ARM CPUs. The backend requires:

1. `GCC` ≥ 12.0 or another C compiler with OpenMP 5.1 support
2. `libxsmm` \geq commit 0db15a0da13e3d9b9e3d57b992ecb3384d2e15ea compiled as a shared library (`STATIC=0`) with `BLAS=0`.

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 1.15.0 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. Additionally, all parameter values support environment variable expansion.

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`

3. collect-wait-times — If to track MPI request wait times or not:

`True | False`

4. collect-wait-times-len — Size of the wait time history buffer:

`int`

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 | uuid

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 | uuid

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 | uuid

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. schedule — OpenMP loop scheduling scheme:

static | dynamic | dynamic, n | guided | guided, n

where *n* is a positive integer.

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
```

(continues on next page)

(continued from previous page)

```
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|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
- **controller** — time-step controller
`none`
- **pseudo-dt** — pseudo time-step
float
- **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

float | string

- **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

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 combination of the geometric shapes specified in [Regions](#), or a sub-region of elements that have faces on a specific domain boundary via the name of the domain boundary:

`* | shape(args, ...) | 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 = box((-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 written, specified as either the entire domain using *, a combination of the geometric shapes specified in *Regions*, or a sub-region of elements that have faces on a specific domain boundary via the name of the domain boundary:

* | shape(args, ...) | *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. `region` — region to integrate, specified as either the entire domain using `*` or a combination of the geometric shapes specified in *Regions*:

`* | shape(args, ...)`

7. `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 Regions

Certain plugins are capable of performing operations on a subset of the elements inside the domain. One means of constructing these element subsets is through parameterised regions. Note that an element is considered part of a region if *any* of its nodes are found to be contained within the region. Supported regions:

Rectangular cuboid box(`x0`, `x1`)

A rectangular cuboid defined by two diametrically opposed vertices. Valid in both 2D and 3D.

Conical frustum `conical_frustum(x0, x1, r0, r1)`

A conical frustum whose end caps are at $x0$ and $x1$ with radii $r0$ and $r1$, respectively. Only valid in 3D.

Cone `cone(x0, x1, r)`

A cone of radius r whose centre-line is defined by $x0$ and $x1$. Equivalent to `conical_frustum(x0, x1, r, 0)`. Only valid in 3D.

Cylinder cylinder(x_0 , x_1 , r)

A circular cylinder of radius r whose centre-line is defined by x_0 and x_1 . Equivalent to `conical_frustum(x0, x1, r, r)`. Only valid in 3D.

Cartesian ellipsoid ellipsoid(x_0 , a , b , c)

An ellipsoid centred at x_0 with Cartesian coordinate axes whose extents in the x , y , and z directions are given by a , b , and c , respectively. Only valid in 3D.

Sphere sphere(x_0 , r)

A sphere centred at x_0 with a radius of r . Equivalent to `ellipsoid(x0, r, r, r)`. Only valid in 3D.

Region expressions can also be added and subtracted together arbitrarily. For example `box((-10, -10, -10), (10, 10, 10)) - sphere((0, 0, 0), 3)` will result in a cube-shaped region with a sphere cut out of the middle.

2.2.7 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 1.15.0:

`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_axnpy_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_errest
formulation = 'std'
static get_plugin_data_prefix(name, suffix)
property grad_soln
property nsteps
run()
property soln
step(t, dt)
```

StdPIController [Click to show](#)

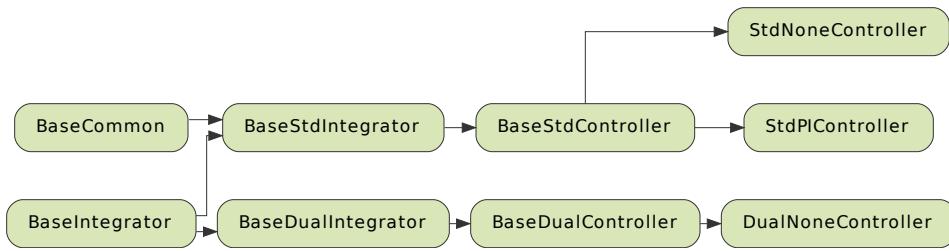
```
class pyfr.integrators.std.controllers.StdPIController(*args, **kwargs)
    _accept_step(dt, idxcurr, err=None)
    _add(*args, subdims=None)
    _addv(consts, regidxs, subdims=None)
    _check_abort()
    _errest(rcurr, rprev, rerr)
    _get_axnpby_kerns(*rs, subdims=None)
    _get_gndofs()
    _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 1.15.0:

`DualNonePseudoController` [Click to show](#)

```
class pyfr.integrators.dual.pseudo.pseudocontrollers.DualNonePseudoController(*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
    _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)
    formulation = 'dual'
    init_stage(currstg, stepper_coeffs, dt)
    obtain_solution(bcoeffs)
```

```

pseudo_advance(tcurr)
pseudo_controller_name = 'none'
pseudo_controller_needs_lerrest = False
store_current_soln()

```

DualPIPpseudoController [Click to show](#)

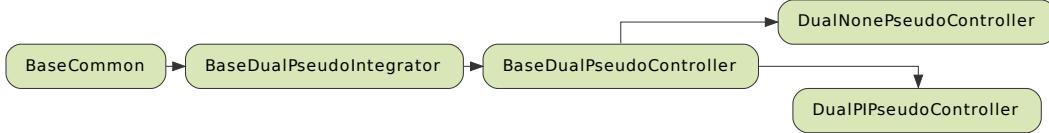
```

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 stepper_regidx
    property _stage stepper_regidx
    property _stepper stepper_regidx
    _update_pseudostepinfo(niters, resid)
    aux_nregs = 0
    convmon(i, minniters, dt_fac=1)
    formulation = 'dual'
    init_stage(currstg, stepper_coeffs, dt)
    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 1.15.0:

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_erreest
    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)

    _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 = [0.22502245872571303, 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, initsofn, cfg)
    _add(*args, subdims=None)
    _addv(consts, regidxs, subdims=None)
    _check_abort()
    _get_axnpby_kerns(*rs, subdims=None)
    _get_gndofs()
    _get_plugins(initsofn)
    _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
```

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_name = 'backward-euler'
    stepper_nregs = 1
    property system

SDIRK33Stepper Click to show
class pyfr.integrators.dual.phys.steppers.SDIRK33Stepper(*args, **kwargs)
    _al = 0.43586652150845906
    _at = 0.11327896981804066
    _check_abort()
    _finalize_step()
    _get_plugins(initsoln)

```

```
a = [[0.43586652150845906], [0.28206673924577047, 0.43586652150845906],  
[1.2084966491760103, -0.6443631706844692, 0.43586652150845906]]  
  
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 = 3  
  
property nsteps  
  
property pseudostepinfo  
  
run()  
  
property soln  
  
property stage_nregs  
  
step(t, dt)  
  
stepper_name = 'sdirk33'  
  
stepper_nregs = 1  
  
property system
```

SDIRK43Stepper Click to show

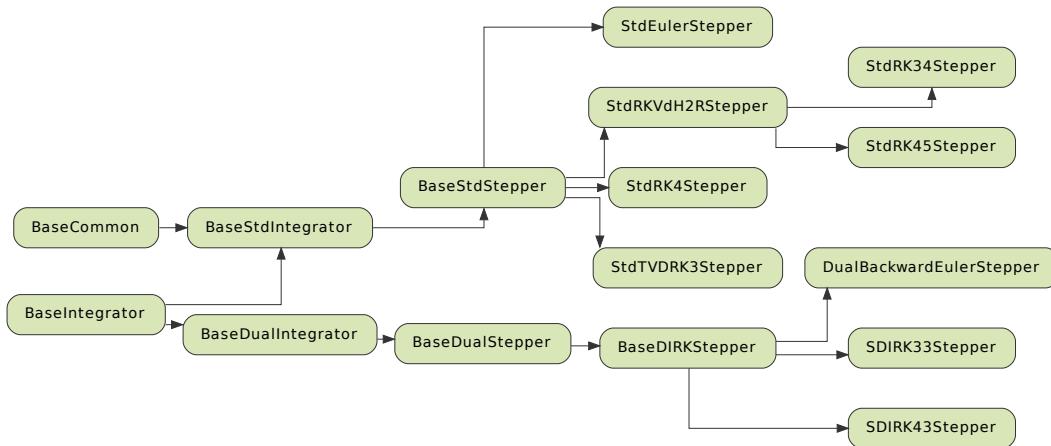
```
class pyfr.integrators.dual.phys.steppers.SDIRK43Stepper(*args, **kwargs)  
  
_a_lam = 1.0685790213016289  
  
_b_rlam = 0.1288864005157204  
  
_check_abort()  
  
_finalize_step()  
  
_get_plugins(initsoln)  
  
a = [[1.0685790213016289], [-0.5685790213016289, 1.0685790213016289],  
[2.1371580426032577, -3.2743160852065154, 1.0685790213016289]]  
  
advance_to(t)  
  
b = [0.1288864005157204, 0.7422271989685592, 0.1288864005157204]  
  
call_plugin_dt(dt)
```

```

property cfgmeta
collect_stats(stats)
formulation = 'dual'
fsal = False
static get_plugin_data_prefix(name, suffix)
property grad_soln
nstages = 3
property nsteps
property pseudostepinfo
run()
property soln
property stage_nregs
step(t, dt)
stepper_name = 'sdirk43'
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 1.15.0:

*DualDenseRK**PseudoStepper* [Click to show](#)

```
class pyfr.integrators.dual.pseudo.pseudosteppers.DualDenseRKpseudoStepper(*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
    _rhs_with_dts(t, uin, fout)
    property _source_regidx
    property _stage_regidx
    property _stepper_regidx
    aux_nregs = 0
    collect_stats(stats)
    formulation = 'dual'
    init_stage(currstg, stepper_coeffs, dt)
    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)
formulation = 'dual'
init_stage(currstg, stepper_coeffs, dt)
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)
formulation = 'dual'
init_stage(currstg, stepper_coeffs, dt)
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)
formulation = 'dual'
init_stage(currstg, stepper_coeffs, dt)
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)

formulation = 'dual'

init_stage(currstg, stepper_coeffs, dt)

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)

formulation = 'dual'

init_stage(currstg, stepper_coeffs, dt)

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)

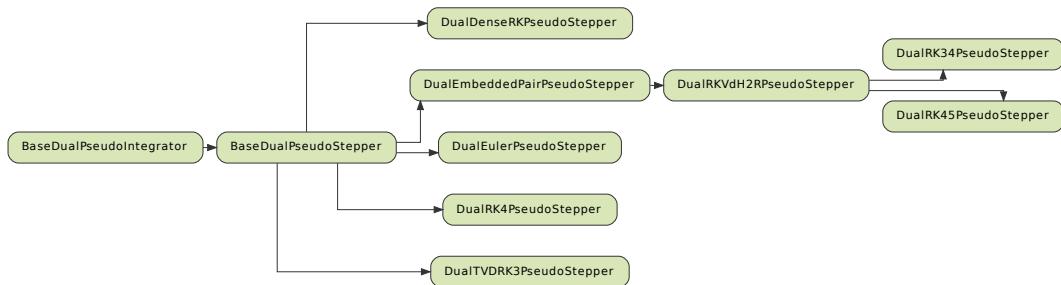
store_current_soln()

```

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 1.15.0 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 1.15.0:

ACEulerSystem [Click to show](#)

```
class pyfr.solvers.aceuler.system.ACEulerSystem(backend, rallocs, mesh, initsoln, nregs, cfg)

    _compute_grads_graph(t, uinbank)

    _gen_kernels(nregs, eles, iint, mpiint, bcint)

    _gen_mpireqs(mpiint)

    _get_kernels(uinbank, foutbank)

    _kdeps(kdict, kern, *dnames)

    _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)

    _prepare_kernels(t, uinbank, foutbank)

    _rhs_graphs(uinbank, foutbank)

bbcinterscls
    alias of ACEulerBaseBCInters

compute_grads(t, uinbank)

ele_scal_upts(idx)

elementscls
    alias of ACEulerElements

filt(uinoutbank)

intinterscls
    alias of ACEulerIntInters

mpiinterscls
    alias of ACEulerMPIInters

name = 'ac-euler'
```

rhs(*t*, *uinbank*, *foutbank*)

rhs_wait_times()

ACNavierStokesSystem [Click to show](#)

```
class pyfr.solvers.acnavstokes.system.ACNavierStokesSystem(backend, rallocs, mesh, initsohn, nregs, cfg)
```

_compute_grads_graph(*uinbank*)

_gen_kernels(*nregs*, *eles*, *iint*, *mpiint*, *bcint*)

_gen_mpireqs(*mpiint*)

_get_kernels(*uinbank*, *foutbank*)

_kdeps(*kdict*, *kern*, **dnames*)

_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)

_prepare_kernels(*t*, *uinbank*, *foutbank*)

_rhs_graphs(*uinbank*, *foutbank*)

bbcinterscls

alias of *ACNavierStokesBaseBCInters*

compute_grads(*t*, *uinbank*)

ele_scal_upts(*idx*)

elementscls

alias of *ACNavierStokesElements*

filt(*uinoutbank*)

intinterscls

alias of *ACNavierStokesIntInters*

mpiinterscls

alias of *ACNavierStokesMPIInters*

name = 'ac-navier-stokes'

rhs(*t*, *uinbank*, *foutbank*)

rhs_wait_times()

EulerSystem [Click to show](#)

```
class pyfr.solvers.euler.system.EulerSystem(backend, rallocs, mesh, initsohn, nregs, cfg)
```

```
_compute_grads_graph(t, uinbank)
_gen_kernels(nregs, eles, iint, mpiint, bcint)
_gen_mpireqs(mpiint)
_get_kernels(uinbank, foutbank)
_kdeps(kdict, kern, *dnames)
_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)
_prepare_kernels(t, uinbank, foutbank)
_rhs_graphs(uinbank, foutbank)
bbcinterscls
    alias of EulerBaseBCInters
compute_grads(t, uinbank)
ele_scal_upts(idx)
elementscls
    alias of EulerElements
filt(uinoutbank)
intinterscls
    alias of EulerIntInters
mpiinterscls
    alias of EulerMPIInters
name = 'euler'
rhs(t, uinbank, foutbank)
rhs_wait_times()
```

NavierStokesSystem [Click to show](#)

```
class pyfr.solvers.navstokes.system.NavierStokesSystem(backend, rallocs, mesh, initsoln, nregs, cfg)
    _compute_grads_graph(uinbank)
    _gen_kernels(nregs, eles, iint, mpiint, bcint)
    _gen_mpireqs(mpiint)
    _get_kernels(uinbank, foutbank)
    _kdeps(kdict, kern, *dnames)
```

```

_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)
_prepare_kernels(t, uinbank, foutbank)
_rhs_graphs(uinbank, foutbank)

bbcinterscls
    alias of NavierStokesBaseBCInters
compute_grads(t, uinbank)
ele_scal_upts(idx)

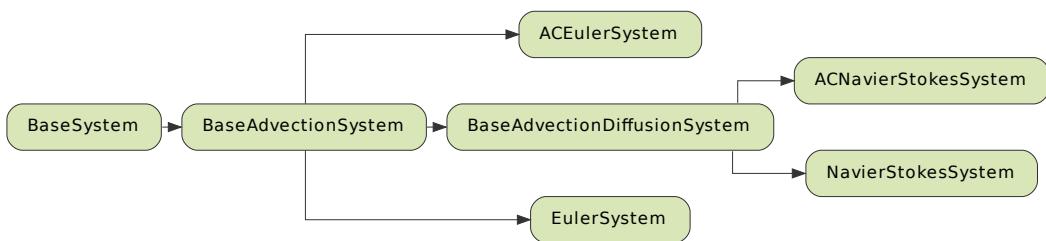
elementscls
    alias of NavierStokesElements
filt(uinoutbank)

intinterscls
    alias of NavierStokesIntInters

mpiinterscls
    alias of NavierStokesMPIInters
name = 'navier-stokes'
rhs(t, uinbank, foutbank)
rhs_wait_times()

```

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 1.15.0:

ACEulerElements [Click to show](#)

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

    property _mesh_regions
    property _ploc_in_src_exprs
    property _pnorm_fpts
    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']}
    curved_smat_at(name)
    dualcoeffs = {2: ['u', 'v'], 3: ['u', 'v', 'w']}
    formulations = ['dual']
    get_ploc_for_inter(eidx, fidx)
    get_pnorms(eidx, fidx)
    get_pnorms_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_np(name)
property upto
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)
property _mesh_regions
property _ploc_in_src_expressions
property _pnorm_fpts
property _scratch_bufs
_slice_mat(mat, region, ra=None, rb=None)
property _smats_djacs_mpts
property _soln_in_src_expressions
property _src_expressions
property _srtd_face_fpts
static con_to_pri(convs, cfg)
convvarmap = {2: ['p', 'u', 'v'], 3: ['p', 'u', 'v', 'w']}
curved_smat_at(name)
dualcoeffs = {2: ['u', 'v'], 3: ['u', 'v', 'w']}
formulations = ['dual']
get_artvisc_fpts_for_inter(eidx, fidx)
get_ploc_for_inter(eidx, fidx)
get_pnorms(eidx, fidx)
get_pnorms_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_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)
    property _mesh_regions
    property _ploc_in_src_expressions
    property _pnorm_fpts
    property _scratch_bufs
    _slice_mat(mat, region, ra=None, rb=None)
    property _smats_djacs_mpts
    property _soln_in_src_expressions
    property _src_expressions
    property _srtd_face_fpts
```

```

static con_to_pri(cons, cfg)

convarmap = {2: ['rho', 'rhou', 'rhov', 'E'], 3: ['rho', 'rhou', 'rhov', 'rhow',
'E']}

curved_smat_at(name)

dualcoeffs = {2: ['rho', 'rhou', 'rhov', 'E'], 3: ['rho', 'rhou', 'rhov', 'rhow',
'E']}

formulations = ['std', 'dual']

get_ploc_for_inter(eidx, fidx)

get_pnorms(eidx, fidx)

get_pnorms_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: ['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_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)
```

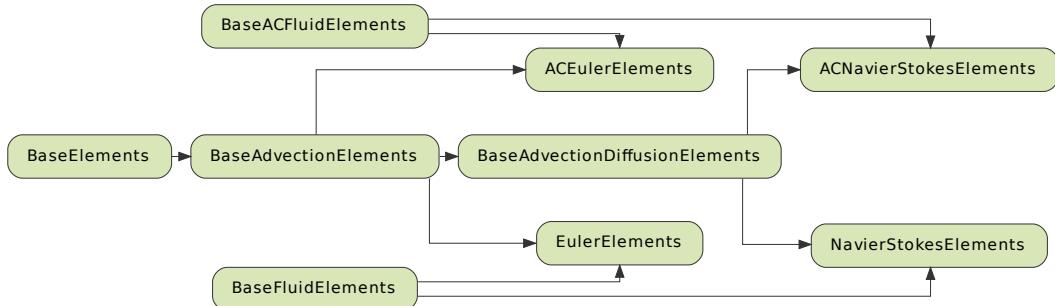
```
property _mesh_regions
property _ploc_in_src_exprs
property _pnorm_fpts
property _scratch_bufs
_slice_mat(mat, region, ra=None, rb=None)
property _smats_djacs_mpts
property _soln_in_src_exprs
property _src_exprs
property _srtd_face_fpts
static con_to_pri(cons, cfg)
convarmap = {2: ['rho', 'rhou', 'rhov', 'E'], 3: ['rho', 'rhou', 'rhov', 'rhow', 'E']}
curved_smat_at(name)
dualcoeffs = {2: ['rho', 'rhou', 'rhov', 'E'], 3: ['rho', 'rhou', 'rhov', 'rhow', 'E']}
formulations = ['std', 'dual']
get_artvisc_fpts_for_inter(eidx, fidx)
get_ploc_for_inter(eidx, fidx)
get_pnorms(eidx, fidx)
get_pnorms_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_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 1.15.0:

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)

    BASE_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)
```

```

BASE_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)

BASE_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)

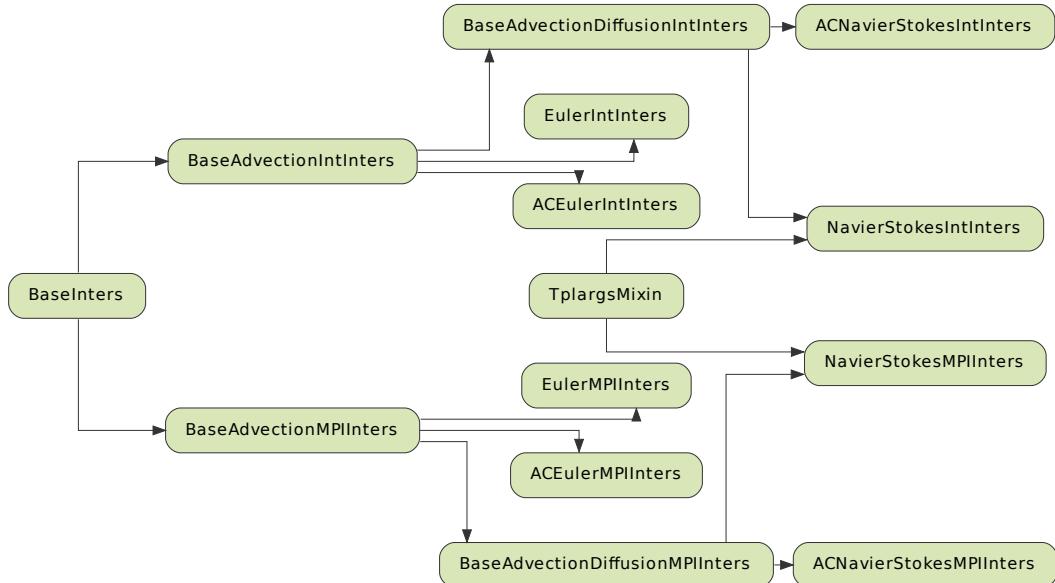
_const_mat(inter, meth)
_gen_perm(lhs, rhs)
_get_perm_for_view(inter, meth)
scal_view(inter, meth)
scal_xchg_view(inter, meth)
_set_external(name, spec, value=None)
_vect_view(inter, meth)
_vect_xchg_view(inter, meth)
_view(inter, meth, vshape=())
_xchg_view(inter, meth, vshape=())
```

```
prepare(t)
```

NavierStokesMPIInters [Click to show](#)

```
class pyfr.solvers.navstokes.inters.NavierStokesMPIInters(be, lhs, rhsrank, rallocs, elemap, cfg)
    BASE_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 1.15.0:

CUDABackend [Click to show](#)

```
class pyfr.backends.cuda.base.CUDABackend(cfg)

    _malloc_impl(nbytes)
    alias(obj, aobj)
    blocks = False
    commit()
    const_matrix(initval, dtype=None, tags={})
    graph()
    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'
    ordered_meta_kernel(kerns)
    run_graph(graph, wait=False)
    run_kernels(kernels, wait=False)
    unordered_meta_kernel(kerns)
    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, dtype=None, tags={})
graph()
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'
ordered_meta_kernel(kerns)
run_graph(graph, wait=False)
run_kernels(kernels, wait=False)
unordered_meta_kernel(kerns)
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, dtype=None, tags={})
    graph()
    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'

```

```
ordered_meta_kernel(kerns)
run_graph(graph, wait=False)
run_kernels(kernels, wait=False)
unordered_meta_kernel(kerns)

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

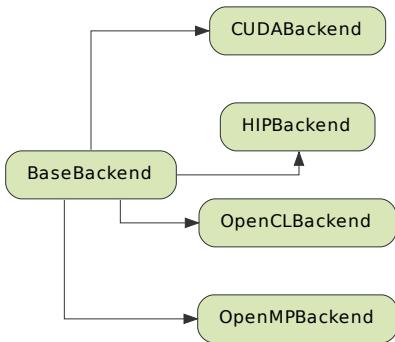
```
class pyfr.backends.openmp.base.OpenMPBackend(cfg)

_malloc_impl(nbytes)
alias(obj, aobj)
blocks = True
commit()
const_matrix(initval, dtype=None, tags={})
graph()
kernel(name, *args, **kwargs)
property krunner
property lookup
malloc(obj, extent)
matrix(ioshape, initval=None, extent=None, aliases=None, tags={})
matrix_slice(mat, ra, rb, ca, cb)
name = 'openmp'
ordered_meta_kernel(kerns)
run_graph(graph, wait=False)
run_kernels(kernels, wait=False)
unordered_meta_kernel(kerns)

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 1.15.0:

`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, tplsargs)
```

kernel_generator_clsalias of [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 OpenCLKernelGenerator

register(mod)
```

OpenMPPointwiseKernelProvider [Click to show](#)

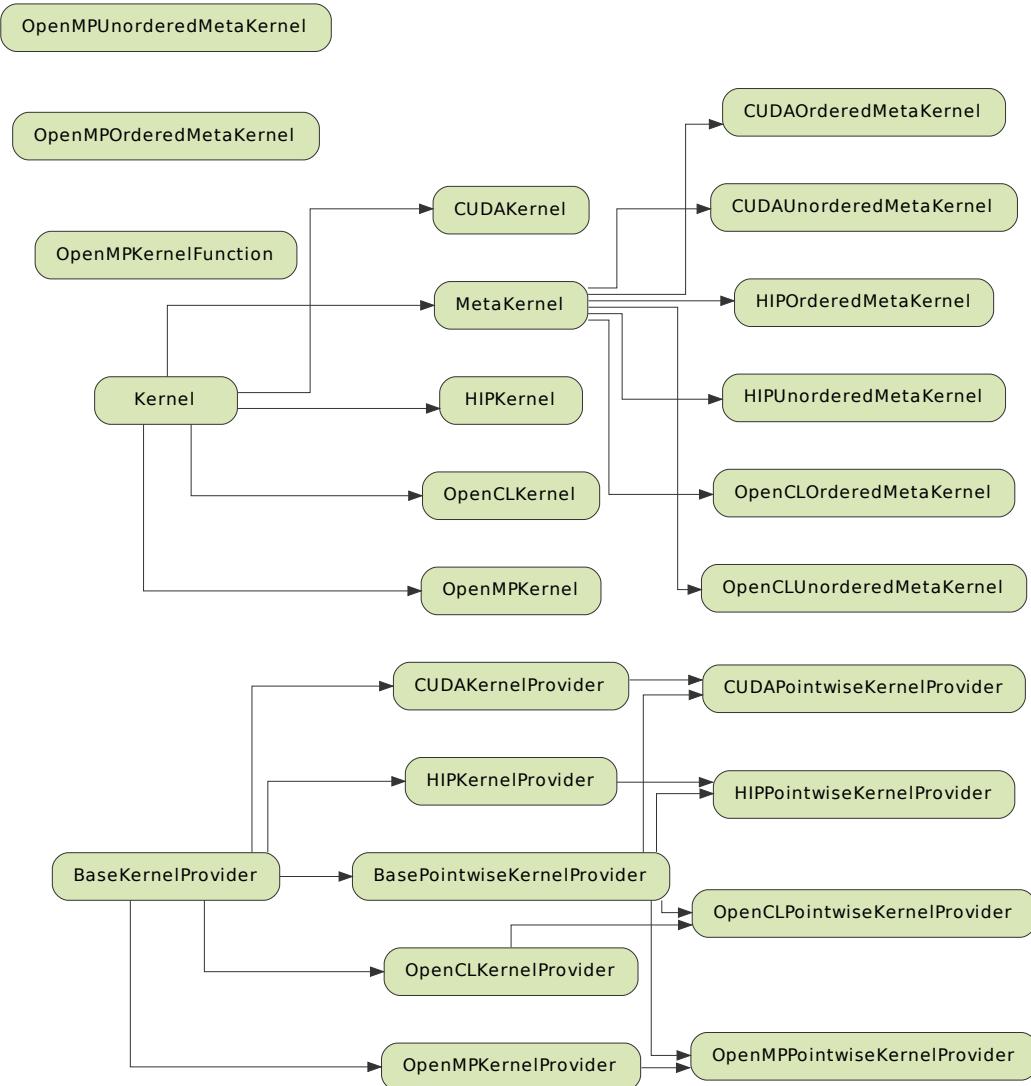
```
class pyfr.backends.openmp.provider.OpenMPPointwiseKernelProvider(*args, **kwargs)

    _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 = None
```

```
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 1.15.0:

CUDAKernelGenerator [Click to show](#)

```
class pyfr.backends.cuda.generator.CUDAKernelGenerator(*args, **kwargs)

    _deref_arg(arg)
    _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(arg)
    _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()
```

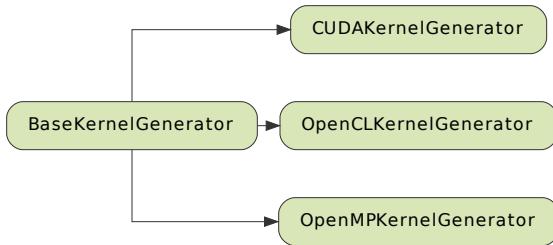
OpenCLKernelGenerator [Click to show](#)

```
class pyfr.backends.opencl.generator.OpenCLKernelGenerator(*args, **kwargs)
    _deref_arg(arg)
    _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(arg)
    _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 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
```

4.1.2 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.

4.1.3 Loop Scheduling

By default PyFR employs static scheduling for loops, with work being split evenly across cores. For systems with a single type of core this is usually the right choice. However, on heterogeneous systems it typically results in load imbalance. Here, it can be beneficial to experiment with the *guided* and *dynamic* loop schedules as:

```
[backend-openmp]
schedule = dynamic, 5
```

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 HIP Backend

4.3.1 HIP-aware MPI

PyFR is capable of taking advantage of HIP-aware MPI. This enables HIP 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 HIP-aware this functionality can be enabled through the `mpi-type` key as:

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

4.4 Partitioning

4.4.1 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.

4.4.2 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.

4.4.3 Detecting load imbalances

PyFR includes code for monitoring the amount of time each rank spends waiting for MPI transfers to complete. This can be used, among other things, to detect load imbalances. Such imbalances are typically observed on mixed-element grids with an incorrect weighting factor. Wait time tracking can be enabled as:

```
[backend]
collect-wait-times = true
```

with the resulting statistics being recorded in the `[backend-wait-times]` section of the `/stats` object which is included in all PyFR solution files. This can be extracted as:

```
h5dump -d /stats -b --output=stats.ini soln.pyfrs
```

Note that the number of graphs depends on the system, and not all graphs initiate MPI requests. The average amount of time each rank spends waiting for MPI requests per right hand side evaluation can be obtained by vertically summing all of the `-median` fields together.

There exists an inverse relationship between the amount of computational work a rank has to perform and the amount of time it spends waiting for MPI requests to complete. Hence, ranks which spend comparatively less time waiting than their peers are likely to be overloaded, whereas those which spend comparatively more time waiting are likely to be underloaded. This information can then be used to explicitly re-weight the partitions and/or the per-element weights.

4.5 Scaling

The general recommendation when running PyFR in parallel is to aim for a parallel efficiency of $\epsilon \simeq 0.8$ with the parallel efficiency being defined as:

$$\epsilon = \frac{1}{N} \frac{T_1}{T_N},$$

where N is the number of ranks, T_1 is the simulation time with one rank, and T_N is the simulation time with N ranks. This represents a reasonable trade-off between the overall time-to-solution and efficient resource utilisation.

4.6 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](#).

4.7 Plugins

A common source of performance issues is running plugins too frequently. Given the time steps taken by PyFR are typically much smaller than those associated with the underlying physics there is seldom any benefit to running integration and/or time average accumulation plugins more frequently than once every 50 steps. Further, when running with adaptive time stepping there is no need to run the NaN check plugin. For simulations with fixed time steps, it is not recommended to run the NaN check plugin more frequently than once every 10 steps.

4.8 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 optimised 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

Test cases are available in the *PyFR-Test-Cases* <<https://github.com/PyFR/PyFR-Test-Cases>> repository. 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. Navigate to the PyFR-Test-Cases/2d-euler-vortex directory:

```
cd PyFR-Test-Cases/2d-euler-vortex
```

2. Run pyfr to convert the Gmsh mesh file into a PyFR mesh file called 2d-euler-vortex.pyfrm:

```
pyfr import 2d-euler-vortex.msh 2d-euler-vortex.pyfrm
```

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

```
pyfr partition 2 2d-euler-vortex.pyfrm .
```

4. Run pyfr to solve the Euler equations on the mesh, generating a series of PyFR solution files called 2d-euler-vortex*.pyfrs:

```
mpiexec -n 2 pyfr run -b cuda -p 2d-euler-vortex.pyfrm 2d-euler-vortex.ini
```

5. Run pyfr on the solution file 2d-euler-vortex-100.0.pyfrs converting it into an unstructured VTK file called 2d-euler-vortex-100.0.vtu:

```
pyfr export 2d-euler-vortex.pyfrm 2d-euler-vortex-100.0.pyfrs 2d-euler-vortex-100.0.  
vtu
```

6. Visualise the unstructured VTK file in Paraview

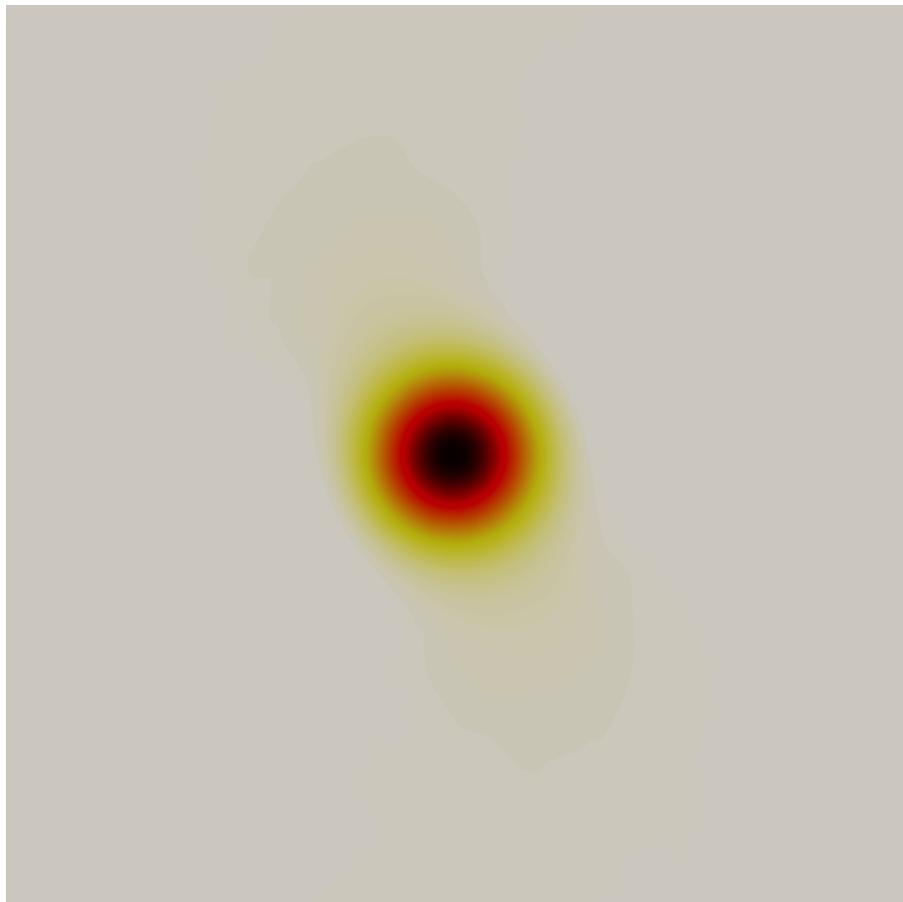


Fig. 1: Colour map of density distribution at 100 time units.

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. Navigate to the PyFR-Test-Cases/2d-couette-flow directory:

```
cd PyFR-Test-Cases/2d-couette-flow
```

2. Run pyfr to convert the Gmsh mesh file into a PyFR mesh file called 2d-couette-flow.pyfrm:

```
pyfr import 2d-couette-flow.msh 2d-couette-flow.pyfrm
```

3. Run pyfr to solve the Navier-Stokes equations on the mesh, generating a series of PyFR solution files called 2d-couette-flow-*.pyfrs:

```
pyfr run -b cuda -p 2d-couette-flow.pyfrm 2d-couette-flow.ini
```

4. Run pyfr on the solution file 2d-couette-flow-040.pyfrs converting it into an unstructured VTK file called 2d-couette-flow-040.vtu:

```
pyfr export 2d-couette-flow.pyfrm 2d-couette-flow-040.pyfrs 2d-couette-flow-040.vtu
```

5. Visualise the unstructured VTK file in Paraview

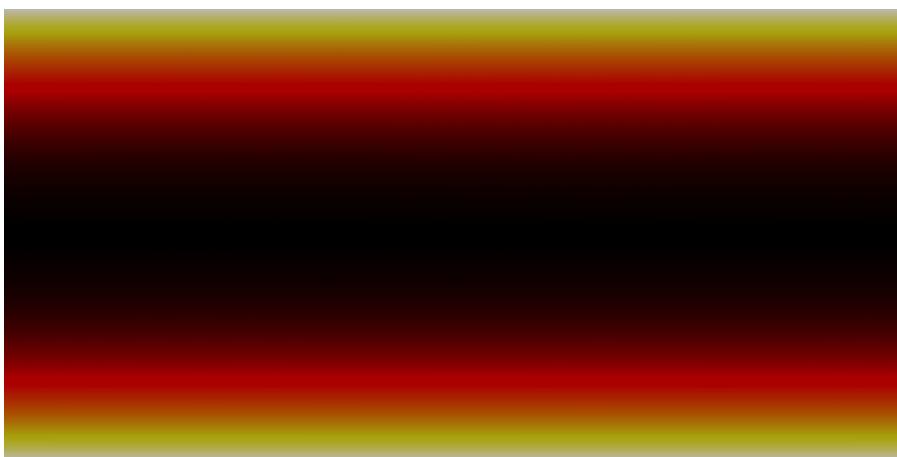


Fig. 2: Colour map of steady-state density distribution.

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. Navigate to the PyFR-Test-Cases/2d-inc-cylinder directory:

```
cd PyFR-Test-Cases/2d-inc-cylinder
```

2. Run pyfr to convert the Gmsh mesh file into a PyFR mesh file called `2d-inc-cylinder.pyfrm`:

```
pyfr import 2d-inc-cylinder.msh 2d-inc-cylinder.pyfrm
```

3. Run pyfr to solve the incompressible Navier-Stokes equations on the mesh, generating a series of PyFR solution files called `2d-inc-cylinder-* .pyfrs`:

```
pyfr run -b cuda -p 2d-inc-cylinder.pyfrm 2d-inc-cylinder.ini
```

4. Run pyfr on the solution file `2d-inc-cylinder-75.00.pyfrs` converting it into an unstructured VTK file called `2d-inc-cylinder-75.00.vtu`:

```
pyfr export 2d-inc-cylinder.pyfrm 2d-inc-cylinder-75.00.pyfrs 2d-inc-cylinder-75.00.  
↪vtu
```

5. Visualise the unstructured VTK file in Paraview

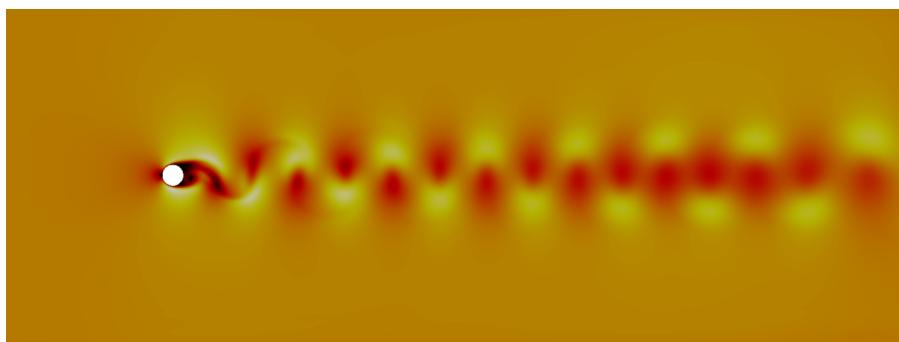


Fig. 3: Colour map of velocity magnitude distribution at 75 time units.

**CHAPTER
SIX**

INDICES AND TABLES

- genindex
- modindex
- search

INDEX

Symbols

_a_1am (pyfr.integrators.dual.phys.steppers.SDIRK43Stepper) `addv()` (pyfr.integrators.dual.pseudo.pseudosteppers.DualDenseRKPseudoController attribute), 42
_accept_step() (pyfr.integrators.dual.phys.controllers.DualNoneController) `addv()` (pyfr.integrators.dual.pseudo.pseudosteppers.DualEulerPseudoStepper method), 46
_accept_step() (pyfr.integrators.std.controllers.StdNoneController) `addv()` (pyfr.integrators.dual.pseudo.pseudosteppers.DualRK34PseudoStepper method), 47
_accept_step() (pyfr.integrators.std.controllers.StdPICController) `addv()` (pyfr.integrators.dual.pseudo.pseudosteppers.DualRK45PseudoStepper method), 48
_add() (pyfr.integrators.dual.pseudo.pseudocontrollers.DualNonePseudoController) `addv()` (pyfr.integrators.dual.pseudo.pseudosteppers.DualRK4PseudoStepper method), 44
_add() (pyfr.integrators.dual.pseudo.pseudocontrollers.DualHIPSseudoController) `addv()` (pyfr.integrators.dual.pseudo.pseudosteppers.DualTVDRK3PseudoStepper method), 45
_add() (pyfr.integrators.dual.pseudo.pseudosteppers.DualDenseRKPseudoStepper) `addv()` (pyfr.integrators.std.controllers.StdNoneController method), 31
_add() (pyfr.integrators.dual.pseudo.pseudosteppers.DualEulerPseudoStepper) `addv()` (pyfr.integrators.std.controllers.StdPICController method), 32
_add() (pyfr.integrators.dual.pseudo.pseudosteppers.DualRK34PseudoStepper) `addv()` (pyfr.integrators.std.steppers.StdEulerStepper method), 36
_add() (pyfr.integrators.dual.pseudo.pseudosteppers.DualRK45PseudoStepper) `addv()` (pyfr.integrators.std.steppers.StdRK34Stepper method), 38
_add() (pyfr.integrators.dual.pseudo.pseudosteppers.DualRK45PseudoStepper) `addv()` (pyfr.integrators.std.steppers.StdRK45Stepper method), 39
_add() (pyfr.integrators.dual.pseudo.pseudosteppers.DualTVDRK3PseudoStepper) `addv()` (pyfr.integrators.std.steppers.StdRK4Stepper method), 37
_add() (pyfr.integrators.std.controllers.StdNoneController) `addv()` (pyfr.integrators.std.steppers.StdTVDRK3Stepper method), 40
_add() (pyfr.integrators.std.controllers.StdPICController) `a1()` (pyfr.integrators.dual.phys.steppers.SDIRK33Stepper attribute), 41
_add() (pyfr.integrators.std.steppers.StdEulerStepper) `at()` (pyfr.integrators.dual.phys.steppers.SDIRK33Stepper attribute), 41
_add() (pyfr.integrators.std.steppers.StdRK34Stepper) `b_r1am()` (pyfr.integrators.dual.phys.steppers.SDIRK43Stepper attribute), 42
_add() (pyfr.integrators.std.steppers.StdRK45Stepper) `build_arglst()` (pyfr.backends.cuda.provider.CUDAPointwiseKernelProvider method), 67
_add() (pyfr.integrators.std.steppers.StdRK4Stepper) `build_arglst()` (pyfr.backends.hip.provider.HIPPointwiseKernelProvider method), 68
_add() (pyfr.integrators.std.steppers.StdTVDRK3Stepper) `build_arglst()` (pyfr.backends.opencl.provider.OpenCLPointwiseKernelProvider method), 68
_addv() (pyfr.integrators.dual.pseudo.pseudocontrollers.DualNonePseudoController) `build_arglst()` (pyfr.backends.openmp.provider.OpenMPPointwiseKernelProvider method), 68
_addv() (pyfr.integrators.dual.pseudo.pseudocontrollers.DualHIPSseudoController) `build_function()` (pyfr.backends.openmp.provider.OpenMPPointwiseKernelProvider method), 68

```
        method), 68
_build_kernel() (pyfr.backends.cuda.provider.CUDAPointwiseKernelProvider._build_kernel() (pyfr.solvers.euler.inters.EulerIntInters
        method), 67
_build_kernel() (pyfr.backends.hip.provider.HIPPointwiseKernelProvider._build_kernel() (pyfr.solvers.euler.inters.EulerMPIInters
        method), 68
_build_kernel() (pyfr.backends.opencl.provider.OpenCLPointwiseKernelProvider._build_kernel() (pyfr.solvers.navstokes.inters.NavierStokesIntInters
        method), 68
_build_kernel() (pyfr.backends.openmp.provider.OpenMPPointwiseKernelProvider._build_kernel() (pyfr.solvers.navstokes.inters.NavierStokesMPIInters
        method), 68
_build_library() (pyfr.backends.openmp.provider.OpenMPPointwiseKernelProvider._build_library() (pyfr.backends.cuda.generator.CUDAKernelGenerator
        method), 68
_build_program() (pyfr.backends.opencl.provider.OpenCLPointwiseKernelProvider._build_program() (pyfr.backends.hip.generator.HIPKernelGenerator
        method), 68
_check_abort() (pyfr.integrators.dual.phys.controllers.DualNoneController._check_abort() (pyfr.backends.opencl.generator.OpenCLKernelGenerator
        method), 33
_check_abort() (pyfr.integrators.dual.phys.steppers.DualBackwardEulerStepper._check_abort() (pyfr.backends.openmp.generator.OpenMPKernelGenerator
        method), 41
_check_abort() (pyfr.integrators.dual.phys.steppers.SDIRK33Stepper._check_abort() (pyfr.backends.cuda.generator.CUDAKernelGenerator
        method), 41
_check_abort() (pyfr.integrators.dual.phys.steppers.SDIRK43Stepper._check_abort() (pyfr.backends.cuda.generator.CUDAKernelGenerator
        method), 42
_deref_arg_array_1d() (pyfr.backends.cuda.generator.CUDAKernelGenerator
        method), 70
_check_abort() (pyfr.integrators.std.controllers.StdNoneController._check_abort() (pyfr.backends.hip.generator.HIPKernelGenerator
        method), 31
_check_abort() (pyfr.integrators.std.controllers.StdPIController._check_abort() (pyfr.backends.opencl.generator.OpenCLKernelGenerator
        method), 32
_check_abort() (pyfr.integrators.std.steppers.StdEulerStepper._check_abort() (pyfr.backends.opencl.generator.OpenCLKernelGenerator
        method), 36
_deref_arg_array_1d() (pyfr.backends.opencl.generator.OpenCLKernelGenerator
        method), 71
_check_abort() (pyfr.integrators.std.steppers.StdRK34Stepper._check_abort() (pyfr.backends.openmp.generator.OpenMPKernelGenerator
        method), 38
_deref_arg_array_1d() (pyfr.backends.openmp.generator.OpenMPKernelGenerator
        method), 71
_check_abort() (pyfr.integrators.std.steppers.StdRK45Stepper._check_abort() (pyfr.backends.cuda.generator.CUDAKernelGenerator
        method), 39
_deref_arg_array_2d() (pyfr.backends.cuda.generator.CUDAKernelGenerator
        method), 70
_check_abort() (pyfr.integrators.std.steppers.StdRK4Stepper._check_abort() (pyfr.backends.hip.generator.HIPKernelGenerator
        method), 37
_deref_arg_array_2d() (pyfr.backends.hip.generator.HIPKernelGenerator
        method), 70
_check_abort() (pyfr.integrators.std.steppers.StdTVDRK3Stepper._check_abort() (pyfr.backends.hip.generator.HIPKernelGenerator
        method), 40
_deref_arg_array_2d() (pyfr.backends.hip.generator.HIPKernelGenerator
        method), 70
_compute_grads_graph() _deref_arg_array_2d() (pyfr.backends.opencl.generator.OpenCLKernelGenerator
        method), 50
(method), 71
_compute_grads_graph() _deref_arg_array_2d() (pyfr.backends.openmp.generator.OpenMPKernelGenerator
        method), 51
(method), 71
_compute_grads_graph() _deref_arg_view() (pyfr.backends.cuda.generator.CUDAKernelGenerator
        method), 51
(method), 70
_compute_grads_graph() _deref_arg_view() (pyfr.backends.hip.generator.HIPKernelGenerator
        method), 70
(method), 71
_compute_grads_graph() _deref_arg_view() (pyfr.backends.opencl.generator.OpenCLKernelGenerator
        method), 52
(method), 71
_const_mat() (pyfr.solvers.aceuler.inters.AC EulerIntInters._deref_arg_view() (pyfr.backends.openmp.generator.OpenMPKernelGenerator
        method), 60
(method), 71
_const_mat() (pyfr.solvers.aceuler.inters.AC EulerMPIInters._deref_arg_view() (pyfr.integrators.std.controllers.StdPIController
        method), 60
(method), 32
_const_mat() (pyfr.solvers.acnavstokes.inters.AC NavierStokesIntInters._finalize_step() (pyfr.integrators.dual.phys.steppers.DualBackwardEuler
        method), 60
(method), 41
_const_mat() (pyfr.solvers.acnavstokes.inters.AC NavierStokesMPIInters._finalize_step() (pyfr.integrators.dual.phys.steppers.SDIRK33Stepper
        method), 60
(method), 41
```

```

        method), 41
_finalize_step() (pyfr.integrators.dual.phys.steppers.SDIRK43Stepper
    method), 42
_get_axnpby_kerns() (pyfr.integrators.std.controllers.StdNoneController
method), 31
_gen_kernels() (pyfr.solvers.aceuler.system.ACEulerSystem
    method), 50
_get_axnpby_kerns() (pyfr.solvers.acnavstokes.system.ACNavierStokesSystem
    method), 51
_get_kernels() (pyfr.solvers.euler.system.EulerSystem
    method), 52
_get_axnpby_kerns() (pyfr.integrators.std.steppers.StdEulerStepper
method), 36
_gen_kernels() (pyfr.solvers.navstokes.system.NavierStokesSystem
    method), 52
_get_axnpby_kerns() (pyfr.integrators.std.steppers.StdRK34Stepper
method), 38
_gen_mpireqs() (pyfr.solvers.aceuler.system.ACEulerSystem
    method), 50
_get_axnpby_kerns() (pyfr.integrators.std.steppers.StdRK45Stepper
method), 39
_gen_mpireqs() (pyfr.solvers.acnavstokes.system.ACNavierStokesSystem
    method), 51
_get_axnpby_kerns() (pyfr.integrators.std.steppers.StdRK4Stepper
method), 37
_gen_mpireqs() (pyfr.solvers.euler.system.EulerSystem
    method), 52
_get_axnpby_kerns() (pyfr.integrators.std.steppers.StdTVDRK3Stepper
method), 40
_gen_perm() (pyfr.solvers.aceuler.inters.AC EulerIntInters
    method), 60
_get_gndofs() (pyfr.integrators.dual.pseudo.pseudocontrollers.DualNone
method), 34
_gen_perm() (pyfr.solvers.acnavstokes.inters.ACNavierStokesIntInters
    method), 60
_get_gndofs() (pyfr.integrators.dual.pseudo.pseudocontrollers.DualPIPs
method), 35
_gen_perm() (pyfr.solvers.euler.inters.EulerIntInters
    method), 61
_get_gndofs() (pyfr.integrators.dual.pseudo.pseudosteppers.DualDenseR
method), 44
_gen_perm() (pyfr.solvers.navstokes.inters.NavierStokesIntInters
    method), 62
_get_gndofs() (pyfr.integrators.dual.pseudo.pseudosteppers.DualEulerP
method), 46
_get_arg_cls() (pyfr.backends.openmp.provider.OpenMPPointwiseKernelProvider
    method), 68
_get_gndofs() (pyfr.integrators.dual.pseudo.pseudosteppers.DualRK34P
method), 47
_get_axnpby_kerns() (pyfr.integrators.dual.pseudo.pseudocontrollers.Du
    method), 34
_get_gndofs() (pyfr.integrators.dual.pseudo.pseudocontrollers.DualPIPSe
method), 48
_get_axnpby_kerns() (pyfr.integrators.dual.pseudo.pseudocontrollers.Du
    method), 35
_get_gndofs() (pyfr.integrators.dual.pseudo.pseudosteppers.DualRK4Pse
method), 45
_get_axnpby_kerns() (pyfr.integrators.dual.pseudo.pseudosteppers.DualR
    method), 44
_get_gndofs() (pyfr.integrators.dual.pseudo.pseudosteppers.DualRK45Pse
method), 31
_get_axnpby_kerns() (pyfr.integrators.dual.pseudo.pseudosteppers.DualR
    method), 46
_get_gndofs() (pyfr.integrators.dual.pseudo.pseudosteppers.DualTVDRK
method), 45
_get_axnpby_kerns() (pyfr.integrators.dual.pseudo.pseudosteppers.DualR
    method), 47
_get_gndofs() (pyfr.integrators.dual.pseudo.pseudosteppers.DualRK34Ste
method), 38
_get_axnpby_kerns() (pyfr.integrators.dual.pseudo.pseudosteppers.DualR
    method), 48
_get_gndofs() (pyfr.integrators.dual.pseudo.pseudosteppers.DualRK45Ste
method), 39
_get_axnpby_kerns() (pyfr.integrators.dual.pseudo.pseudosteppers.DualR
    method), 45
_get_gndofs() (pyfr.integrators.dual.pseudo.pseudosteppers.DualRK4Ste
method), 37
_get_axnpby_kerns() (pyfr.integrators.dual.pseudo.pseudosteppers.DualR
    method), 45
_get_gndofs() (pyfr.integrators.dual.pseudo.pseudosteppers.DualTVDRK3S
method), 40
_get_axnpby_kerns() (pyfr.integrators.dual.pseudo.pseudosteppers.DualTV
    method), 45
_get_kernels() (pyfr.solvers.aceuler.system.ACEulerSystem
method), 51
_get_axnpby_kerns() (pyfr.solvers.acnavstokes.system.ACNavierStokesSystem
method), 51

```

```
_get_kernels() (pyfr.solvers.euler.system.EulerSystem  
    method), 52  
_get_kernels() (pyfr.solvers.navstokes.system.NavierStokes  
    method), 52  
_get_perm_for_view()  
    (pyfr.solvers.aceuler.inters.AC EulerIntInters  
    method), 60  
_get_perm_for_view()  
    (pyfr.solvers.aceuler.inters.AC EulerMPIInters  
    method), 60  
_get_perm_for_view()  
    (pyfr.solvers.acnavstokes.inters.AC NavierStokesInt  
    method), 61  
_get_perm_for_view()  
    (pyfr.solvers.acnavstokes.inters.AC NavierStokesMPI  
    method), 61  
_get_perm_for_view()  
    (pyfr.solvers.euler.inters.EulerIntInters  
    method), 61  
_get_perm_for_view()  
    (pyfr.solvers.euler.inters.EulerMPIInters  
    method), 62  
_get_perm_for_view()  
    (pyfr.solvers.navstokes.inters.NavierStokesIntInters  
    method), 62  
_get_perm_for_view()  
    (pyfr.solvers.navstokes.inters.NavierStokesMPIInters  
    method), 63  
_get_plugins() (pyfr.integrators.dual.phys.controllers.DualNoneController  
    method), 33  
_get_plugins() (pyfr.integrators.dual.phys.steppers.DualBackwardEulerStepper  
    method), 41  
_get_plugins() (pyfr.integrators.dual.phys.steppers.SDIRK23Stepper  
    method), 41  
_get_plugins() (pyfr.integrators.dual.phys.steppers.SDIRK43Stepper  
    method), 42  
_get_plugins() (pyfr.integrators.std.controllers.StdNoneController  
    method), 31  
_get_plugins() (pyfr.integrators.std.controllers.StdPIController  
    method), 32  
_get_plugins() (pyfr.integrators.std.controllers.StdEulerStepper  
    method), 36  
_get_plugins() (pyfr.integrators.std.steppers.StdRK34Stepper  
    method), 38  
_get_plugins() (pyfr.integrators.std.steppers.StdRK45Stepper  
    method), 39  
_get_plugins() (pyfr.integrators.std.steppers.StdRK4Stepper  
    method), 37  
_get_plugins() (pyfr.integrators.std.steppers.StdTVDRK3Stepper  
    method), 40  
_get_reduction_kerns()  
    (pyfr.integrators.dual.pseudo.pseudocontrollers.DualNoneP  
    method), 34  
_get_reduction_kerns()  
    (pyfr.integrators.dual.pseudo.pseudocontrollers.DualNoneP  
    method), 35  
_get_reduction_kerns()  
    (pyfr.integrators.dual.pseudo.pseudosteppers.DualDenseRKPseu  
    method), 44  
_get_reduction_kerns()  
    (pyfr.integrators.dual.pseudo.pseudosteppers.DualEulerPseudoSt  
    method), 46  
_get_reduction_kerns()  
    (pyfr.integrators.dual.pseudo.pseudosteppers.DualRK34PseudoSt  
    method), 47  
_get_reduction_kerns()  
    (pyfr.integrators.dual.pseudo.pseudosteppers.DualRK45PseudoSt  
    method), 48  
_get_reduction_kerns()  
    (pyfr.integrators.dual.pseudo.pseudosteppers.DualRK4PseudoSt  
    method), 45  
_get_reduction_kerns()  
    (pyfr.integrators.dual.pseudo.pseudosteppers.DualTVDRK3Pseu  
    method), 45  
_get_reduction_kerns()  
    (pyfr.integrators.std.controllers.StdNoneController  
    method), 31  
_get_reduction_kerns()  
    (pyfr.integrators.std.controllers.StdPIController  
    method), 32  
_get_reduction_kerns()  
    (pyfr.integrators.std.steppers.StdEulerStepper  
    method), 38  
_get_reduction_kerns()  
    (pyfr.integrators.std.steppers.StdRK34Stepper  
    method), 37  
_get_reduction_kerns()  
    (pyfr.integrators.std.steppers.StdRK4Stepper  
    method), 37  
_get_reduction_kerns()  
    (pyfr.integrators.std.steppers.StdTVDRK3Stepper  
    method), 40  
_get_rkvdh2_kerns()  
    (pyfr.integrators.std.steppers.StdRK34Stepper  
    method), 38  
_get_rkvdh2_kerns()  
    (pyfr.integrators.std.steppers.StdRK45Stepper  
    method), 39  
_get_rkvdh2_pseudo_kerns()  
    (pyfr.integrators.std.steppers.StdRK45Stepper  
    method), 40  
_get_rkvdh2_pseudo_kerns()  
    (pyfr.integrators.dual.pseudo.pseudosteppers.DualRK34PseudoSt  
    method), 47  
_get_rkvdh2_pseudo_kerns()  
    (pyfr.integrators.dual.pseudo.pseudosteppers.DualRK45PseudoSt  
    method), 48  
_instantiate_kernel()
```

```

(pyfr.backends.cuda.provider.CUDAPointwiseKernelProvider.method), 64
    method), 67
_instantiate_kernel() _malloc_impl() (pyfr.backends.opencl.base.OpenCLBackend
    (pyfr.backends.hip.provider.HIPPointwiseKernelProvider.method), 65
method), 68
_instantiate_kernel() _malloc_impl() (pyfr.backends.openmp.base.OpenMPBackend
    (pyfr.backends.opencl.provider.OpenCLPointwiseKernelProvider.method), 54
method), 66
_instantiate_kernel() _mesh_regions (pyfr.solvers.aceuler.elements.ACEulerElements
    (pyfr.backends.opencl.provider.OpenCLPointwiseKernelProvider.property), 54
method), 55
_instantiate_kernel() _mesh_regions (pyfr.solvers.acnavstokes.elements.ACNavierStokesElements
    (pyfr.backends.opencl.provider.OpenCLPointwiseKernelProvider.property), 55
method), 56
_kdeps() (pyfr.solvers.aceuler.system.ACEulerSystem _mesh_regions (pyfr.solvers.navstokes.elements.NavierStokesElements
    method), 50 property), 57
_kdeps() (pyfr.solvers.acnavstokes.system.ACNavierStokesSystem _nonce_seq (pyfr.solvers.aceuler.system.ACEulerSystem
    method), 51 attribute), 50
_kdeps() (pyfr.solvers.euler.system.EulerSystem _nonce_seq (pyfr.solvers.acnavstokes.system.ACNavierStokesSystem
    method), 52 attribute), 51
_kdeps() (pyfr.solvers.navstokes.system.NavierStokesSystem _nonce_seq (pyfr.solvers.euler.system.EulerSystem at-
    method), 52 tribute), 52
_load_bc_inters() (pyfr.solvers.aceuler.system.ACEulerSystem _nonce_seq (pyfr.solvers.navstokes.system.NavierStokesSystem
    method), 50 attribute), 53
_load_bc_inters() (pyfr.solvers.acnavstokes.system.ACNavierStokesSystem _ploc_in_src_exprs (pyfr.solvers.aceuler.elements.ACEulerElements
    method), 51 property), 54
_load_bc_inters() (pyfr.solvers.euler.system.EulerSystem _ploc_in_src_exprs (pyfr.solvers.acnavstokes.elements.ACNavierStokesElements
    method), 52 property), 55
_load_bc_inters() (pyfr.solvers.navstokes.system.NavierStokesSystem _ploc_in_src_exprs (pyfr.solvers.euler.elements.EulerElements
    method), 52 property), 56
_load_eles() (pyfr.solvers.aceuler.system.ACEulerSystem _ploc_in_src_exprs (pyfr.solvers.navstokes.elements.NavierStokesElements
    method), 50 property), 58
_load_eles() (pyfr.solvers.acnavstokes.system.ACNavierStokesSystem _pnorm_fpts (pyfr.solvers.aceuler.elements.ACEulerElements
    method), 51 property), 54
_load_eles() (pyfr.solvers.euler.system.EulerSystem _pnorm_fpts (pyfr.solvers.acnavstokes.elements.ACNavierStokesElements
    method), 52 property), 55
_load_eles() (pyfr.solvers.navstokes.system.NavierStokesSystem _pnorm_fpts (pyfr.solvers.euler.elements.EulerElements
    method), 53 property), 56
_load_int_inters() (pyfr.solvers.aceuler.system.ACEulerSystem _pnorm_fpts (pyfr.solvers.navstokes.elements.NavierStokesElements
    method), 50 property), 58
_load_int_inters() (pyfr.solvers.acnavstokes.system.ACNavierStokesSystem _prepare_kernels() (pyfr.solvers.aceuler.system.ACEulerSystem
    method), 51 method), 50
_load_int_inters() (pyfr.solvers.euler.system.EulerSystem _prepare_kernels() (pyfr.solvers.acnavstokes.system.ACNavierStokesSystem
    method), 52 method), 51
_load_int_inters() (pyfr.solvers.navstokes.system.NavierStokesSystem _pseudo_kernels() (pyfr.solvers.euler.system.EulerSystem
    method), 53 method), 52
_load_mpi_inters() (pyfr.solvers.aceuler.system.ACEulerSystem _pseudo_kernels() (pyfr.solvers.navstokes.system.NavierStokesSystem
    method), 50 method), 53
_load_mpi_inters() (pyfr.solvers.acnavstokes.system.ACNavierStokesSystem _pseudo_stepper_regidx
    method), 51 (pyfr.integrators.dual.pseudo.pseudocontrollers.DualNonePseudo
    property), 34
_load_mpi_inters() (pyfr.solvers.euler.system.EulerSystem _pseudo_stepper_regidx
    method), 52 (pyfr.integrators.dual.pseudo.pseudocontrollers.DualPIPseudoCo
    property), 35
_malloc_impl() (pyfr.backends.cuda.base.CUDABackend_pseudo_stepper_regidx
    method), 64 (pyfr.integrators.dual.pseudo.pseudosteppers.DualDenseRKPsue
    property), 44
_malloc_impl() (pyfr.backends.hip.base.HIPBackend
    method), 65

```

_pseudo_stepper_regridx
 (pyfr.integrators.dual.pseudo.pseudosteppers.DualEulerIntInters.
 method), 53
 (pyfr.integrators.dual.pseudo.pseudosteppers.DualRK34PseudoStep.
 property), 46
_pseudo_stepper_regridx
 (pyfr.integrators.dual.pseudo.pseudosteppers.DualRK34PseudoStep.
 method), 47
_pseudo_stepper_regridx
 (pyfr.integrators.dual.pseudo.pseudosteppers.DualRK45PseudoStep.
 property), 48
_pseudo_stepper_regridx
 (pyfr.integrators.dual.pseudo.pseudosteppers.DualRK45PseudoStep.
 method), 49
_reject_step() (pyfr.integrators.std.controllers.StdNoneController.
 method), 31
_reject_step() (pyfr.integrators.std.controllers.StdPICController.
 method), 32
_render_args() (pyfr.backends.openmp.generator.OpenMPKernelGenerator.
 method), 71
_render_body() (pyfr.backends.cuda.generator.CUDAKernelGenerator.
 method), 70
_render_body() (pyfr.backends.hip.generator.HIPKernelGenerator.
 method), 70
_render_body() (pyfr.backends.opencl.generator.OpenCLKernelGenerator.
 method), 71
_render_body() (pyfr.backends.openmp.generator.OpenMPKernelGenerator.
 method), 71
_render_kernel() (pyfr.backends.cuda.provider.CUDAPointwiseKernelProvider.
 method), 67
_render_kernel() (pyfr.backends.hip.provider.HIPPointwiseKernelProvider.
 method), 68
_render_kernel() (pyfr.backends.opencl.provider.OpenCLPointwiseKernelProvider.
 method), 68
_render_kernel() (pyfr.backends.openmp.provider.OpenMPPointwiseKernelProvider.
 method), 68
_render_spec() (pyfr.backends.cuda.generator.CUDAKernelGenerator.
 method), 70
_render_spec() (pyfr.backends.hip.generator.HIPKernelGenerator.
 method), 70
_resid() (pyfr.integrators.dual.pseudo.pseudocontrollers.DualNonePseudoController.
 method), 34
_resid() (pyfr.integrators.dual.pseudo.pseudocontrollers.DualPICPseudoController.
 method), 35
_rhs_graphs() (pyfr.solvers.aceuler.system.ACEulerSystem._scratch_bufs
 (pyfr.solvers.acnavstokes.elements.ACNavierStokesElements.
 method), 50
 (property), 55
_rhs_graphs() (pyfr.solvers.acnavstokes.system.ACNavierStokesSystem._scratch_bufs
 (pyfr.solvers.euler.elements.EulerElements.
 method), 51
 (property), 56
_rhs_graphs() (pyfr.solvers.euler.system.EulerSystem._scratch_bufs
 (pyfr.solvers.navstokes.elements.NavierStokesElements.
 method), 52
 (property), 58
_rhs_graphs() (pyfr.solvers.navstokes.system.NavierStokesSystem._set_external
 (pyfr.solvers.aceuler.inters.ACEulerIntInters.
 method), 53

```

        method), 60
_set_external() (pyfr.solvers.aceuler.inters.AC EulerMPIInters _src_expressions (pyfr.solvers.aceuler.elements.AC EulerElements
        method), 60
property), 54
_set_external() (pyfr.solvers.acnavstokes.inters.AC NavierStokesMPIInters (pyfr.solvers.acnavstokes.elements.AC NavierStokesElements
        method), 61
property), 55
_set_external() (pyfr.solvers.acnavstokes.inters.AC NavierStokesMPIInters (pyfr.solvers.euler.elements.EulerElements
        method), 61
property), 56
_set_external() (pyfr.solvers.euler.inters.EulerIntInters _src_expressions (pyfr.solvers.navstokes.elements.NavierStokesElements
        method), 61
property), 58
_set_external() (pyfr.solvers.euler.inters.EulerMPIInters _srcd_face_fpts (pyfr.solvers.aceuler.elements.AC EulerElements
        method), 62
property), 54
_set_external() (pyfr.solvers.navstokes.inters.NavierStokesMPIInterface_fpts (pyfr.solvers.acnavstokes.elements.AC NavierStokesElements
        method), 62
property), 55
_set_external() (pyfr.solvers.navstokes.inters.NavierStokesMPIInterface_fpts (pyfr.solvers.euler.elements.EulerElements
        method), 63
property), 56
_slice_mat() (pyfr.solvers.aceuler.elements.AC EulerElements _std_face_fpts (pyfr.solvers.navstokes.elements.NavierStokesElements
        method), 54
property), 58
_slice_mat() (pyfr.solvers.acnavstokes.elements.AC NavierStokesElements _stage_RegIdx (pyfr.integrators.dual.pseudo.pseudocontrollers.DualNone
        method), 55
property), 34
_slice_mat() (pyfr.solvers.euler.elements.EulerElements _stage_RegIdx (pyfr.integrators.dual.pseudo.pseudocontrollers.DualPIPs
        method), 56
property), 35
_slice_mat() (pyfr.solvers.navstokes.elements.NavierStokesElements _stage_RegIdx (pyfr.integrators.dual.pseudo.pseudosteppers.DualDenseR
        method), 58
property), 44
_smats_djacs_mpts (pyfr.solvers.aceuler.elements.AC EulerElements _stage_RegIdx (pyfr.integrators.dual.pseudo.pseudosteppers.DualEulerPs
        method), 54
property), 46
_smats_djacs_mpts (pyfr.solvers.acnavstokes.elements.AC NavierStokesElements _stage_RegIdx (pyfr.integrators.dual.pseudo.pseudosteppers.DualRK34P
        method), 55
property), 47
_smats_djacs_mpts (pyfr.solvers.euler.elements.EulerElements _stage_RegIdx (pyfr.integrators.dual.pseudo.pseudosteppers.DualRK45P
        method), 56
property), 48
_smats_djacs_mpts (pyfr.solvers.navstokes.elements.NavierStokesElements _stage_RegIdx (pyfr.integrators.dual.pseudo.pseudosteppers.DualRK4Ps
        method), 58
property), 45
_soln_in_src_expressions (pyfr.solvers.aceuler.elements.AC EulerElements _stage_RegIdx (pyfr.integrators.dual.pseudo.pseudosteppers.DualTVDRK
        method), 54
property), 46
_soln_in_src_expressions (pyfr.solvers.acnavstokes.elements.AC NavierStokesElements _stage_RegIdx (pyfr.integrators.std.steppers.StdEulerStepper
        method), 55
property), 36
_soln_in_src_expressions (pyfr.solvers.euler.elements.EulerElements _stepper_nfevals (pyfr.integrators.std.steppers.StdRK34Stepper
        method), 56
property), 38
_soln_in_src_expressions (pyfr.solvers.navstokes.elements.NavierStokesElements _stepper_nfevals (pyfr.integrators.std.steppers.StdRK45Stepper
        method), 58
property), 39
_source_RegIdx (pyfr.integrators.dual.pseudo.pseudocontrollers._stepper_nfevals (pyfr.integrators.std.steppers.StdRK4Stepper
        method), 34
property), 37
_source_RegIdx (pyfr.integrators.dual.pseudo.pseudocontrollers._stepper_nfevals (pyfr.integrators.std.steppers.StdTVDRK3Stepper
        method), 35
property), 40
_source_RegIdx (pyfr.integrators.dual.pseudo.pseudosteppers._stepper_nfevals (pyfr.integrators.dual.pseudo.pseudocontrollers.DualNone
        method), 44
property), 34
_source_RegIdx (pyfr.integrators.dual.pseudo.pseudosteppers._stepper_nfevals (pyfr.integrators.dual.pseudo.pseudocontrollers.DualNone
        method), 46
property), 35
_source_RegIdx (pyfr.integrators.dual.pseudo.pseudosteppers._stepper_nfevals (pyfr.integrators.dual.pseudo.pseudosteppers.DualDenseR
        method), 47
property), 44
_source_RegIdx (pyfr.integrators.dual.pseudo.pseudosteppers._stepper_nfevals (pyfr.integrators.dual.pseudo.pseudosteppers.DualEulerPs
        method), 48
property), 46
_source_RegIdx (pyfr.integrators.dual.pseudo.pseudosteppers._stepper_nfevals (pyfr.integrators.dual.pseudo.pseudosteppers.DualRK34P
        method), 45
property), 47
_source_RegIdx (pyfr.integrators.dual.pseudo.pseudosteppers._stepper_nfevals (pyfr.integrators.dual.pseudo.pseudosteppers.DualRK45P
        method), 45
property), 47
_source_RegIdx (pyfr.integrators.dual.pseudo.pseudosteppers._stepper_nfevals (pyfr.integrators.dual.pseudo.pseudosteppers.DualRK4Ps
        method), 45
property), 47

```

property), 48
_stepper_regidx(pyfr.integrators.dual.pseudo.pseudostepinfo) (pyfr.integrators.dual.pseudo.pseudostepinfo method), 62
_stepper_regidx(pyfr.integrators.dual.pseudo.pseudostepinfo) (pyfr.integrators.dual.pseudo.pseudostepinfo method), 62
_stepper_regidx(pyfr.integrators.dual.pseudo.pseudostepinfo) (pyfr.integrators.dual.pseudo.pseudostepinfo method), 63
_update_pseudostepinfo() (pyfr.integrators.dual.pseudo.pseudocontrollers.DualNonePseudoController method), 34
_update_pseudostepinfo() (pyfr.integrators.dual.pseudo.pseudocontrollers.DualNonePseudoController method), 60
_update_pseudostepinfo() (pyfr.integrators.dual.pseudo.pseudocontrollers.DualNonePseudoController method), 61
_vect_view() (pyfr.solvers.aceuler.inters.ACEulerIntInters_xchg_view()) (pyfr.solvers.acnavstokes.inters.ACNavierStokesIntInters method), 60
_vect_view() (pyfr.solvers.aceuler.inters.ACEulerMPIInters_xchg_view()) (pyfr.solvers.euler.inters.EulerIntInters method), 62
_vect_view() (pyfr.solvers.acnavstokes.inters.ACNavierStokesIntInters_xchg_view()) (pyfr.solvers.euler.inters.EulerMPIInters method), 61
_vect_view() (pyfr.solvers.acnavstokes.inters.ACNavierStokesIntInters_xchg_view()) (pyfr.solvers.navstokes.inters.NavierStokesIntInters method), 62
_vect_view() (pyfr.solvers.euler.inters.EulerIntInters_xchg_view()) (pyfr.solvers.navstokes.inters.NavierStokesMPIInters method), 63
_vect_view() (pyfr.solvers.euler.inters.EulerMPIInters_xchg_view()) (pyfr.solvers.euler.inters.EulerMPIInters method), 62
_vect_view() (pyfr.solvers.navstokes.inters.NavierStokesIntInters_xchg_view()) (pyfr.integrators.dual.phys.steppers.DualBackwardEulerStepper attribute), 41
_vect_view() (pyfr.solvers.navstokes.inters.NavierStokesMPIInters_xchg_view()) (pyfr.integrators.dual.phys.steppers.SDIRK33Stepper attribute), 41
_vect_xchg_view() (pyfr.solvers.aceuler.inters.ACEulerIntInters_xchg_view()) (pyfr.integrators.dual.phys.steppers.SDIRK43Stepper attribute), 42
_vect_xchg_view() (pyfr.solvers.aceuler.inters.ACEulerMPIInters_xchg_view()) (pyfr.integrators.dual.pseudo.pseudostepers.DualRK34PseudoStepper attribute), 47
_vect_xchg_view() (pyfr.solvers.acnavstokes.inters.ACNavierStokesIntInters_xchg_view()) (pyfr.integrators.dual.pseudo.pseudostepers.DualRK45PseudoStepper attribute), 48
_vect_xchg_view() (pyfr.solvers.acnavstokes.inters.ACNavierStokesMPIInters_xchg_view()) (pyfr.integrators.std.steppers.StdRK34Stepper attribute), 38
_vect_xchg_view() (pyfr.solvers.euler.inters.EulerIntInters_xchg_view()) (pyfr.integrators.std.steppers.StdRK45Stepper attribute), 39
_vect_xchg_view() (pyfr.solvers.euler.inters.EulerMPIInters_xchg_view()) (ACEulerElements class in pyfr.solvers.aceuler.elements), 54
_vect_xchg_view() (pyfr.solvers.navstokes.inters.NavierStokesIntInters_xchg_view()) (ACEulerIntInters class in pyfr.solvers.aceuler.inters), 60
_vect_xchg_view() (pyfr.solvers.navstokes.inters.NavierStokesMPIInters_xchg_view()) (ACEulerMPIInters class in pyfr.solvers.aceuler.inters), 60
_view() (pyfr.solvers.aceuler.inters.ACEulerIntInters_xview()) (ACEulerSystem class in pyfr.solvers.aceuler.system), 50
_view() (pyfr.solvers.aceuler.inters.ACEulerMPIInters_xview()) (ACNavierStokesElements class in pyfr.solvers.acnavstokes.elements), 55
_view() (pyfr.solvers.acnavstokes.inters.ACNavierStokesIntInters_xview()) (ACNavierStokesIntInters class in pyfr.solvers.acnavstokes.inters), 60
_view() (pyfr.solvers.acnavstokes.inters.ACNavierStokesMPIInters_xview()) (ACNavierStokesMPIInters class in pyfr.solvers.acnavstokes.inters), 61
_view() (pyfr.solvers.euler.inters.EulerIntInters_xview()) (ACNavierStokesSystem class in pyfr.solvers.acnavstokes.system), 51
_view() (pyfr.solvers.euler.inters.EulerMPIInters_xview()) (advance_to() (pyfr.integrators.dual.phys.controllers.DualNoneController method), 33

A

B

- advance_to() (*pyfr.integrators.dual.phys.steppers.DualBackwardEulerStepper method*), 41
- advance_to() (*pyfr.integrators.dual.phys.steppers.SDIRK33Stepper method*), 42
 - b (*pyfr.integrators.dual.phys.steppers.SDIRK43Stepper attribute*), 42
 - b (*pyfr.integrators.dual.pseudo.pseudosteppers.DualRK34PseudoStepper attribute*), 47
- advance_to() (*pyfr.integrators.dual.phys.steppers.SDIRK43Stepper method*), 42
 - b (*pyfr.integrators.dual.pseudo.pseudosteppers.DualRK45PseudoStepper attribute*), 48
- advance_to() (*pyfr.integrators.std.controllers.StdNoneController method*), 31
 - b (*pyfr.integrators.std.steppers.StdRK34Stepper attribute*), 38
- advance_to() (*pyfr.integrators.std.controllers.StdPIController method*), 32
 - b (*pyfr.integrators.std.steppers.StdRK45Stepper attribute*), 39
- advance_to() (*pyfr.integrators.std.steppers.StdEulerStepper method*), 36
 - BASE_MPI_TAG (*pyfr.solvers.aceuler.inters.ACEulerMPIInters attribute*), 60
- advance_to() (*pyfr.integrators.std.steppers.StdRK34Stepper method*), 38
 - BASE_MPI_TAG (*pyfr.solvers.acnavstokes.inters.ACNavierStokesMPIInters attribute*), 61
- advance_to() (*pyfr.integrators.std.steppers.StdRK45Stepper method*), 39
 - BASE_MPI_TAG (*pyfr.solvers.euler.inters.EulerMPIInters attribute*), 62
- advance_to() (*pyfr.integrators.std.steppers.StdRK4Stepper method*), 37
 - BASE_MPI_TAG (*pyfr.solvers.navstokes.inters.NavierStokesMPIInters attribute*), 63
- advance_to() (*pyfr.integrators.std.steppers.StdTVDRK3Stepper method*), 40
 - bbcinterscls (*pyfr.solvers.aceuler.system.ACEulerSystem attribute*), 50
- alias() (*pyfr.backends.cuda.base.CUDABackend method*), 64
- alias() (*pyfr.backends.hip.base.HIPBackend method*), 64
- alias() (*pyfr.backends.opencl.base.OpenCLBackend method*), 65
- alias() (*pyfr.backends.openmp.base.OpenMPBackend method*), 66
- argspec() (*pyfr.backends.cuda.generator.CUDAKernelGenerator method*), 70
- argspec() (*pyfr.backends.hip.generator.HIPKernelGenerator method*), 70
 - bhat (*pyfr.integrators.dual.pseudo.pseudosteppers.DualRK34PseudoStepper attribute*), 47
 - bhat (*pyfr.integrators.dual.pseudo.pseudosteppers.DualRK45PseudoStepper attribute*), 48
 - bhat (*pyfr.integrators.std.steppers.StdRK34Stepper attribute*), 38
 - bhat (*pyfr.integrators.std.steppers.StdRK45Stepper attribute*), 39
- argspec() (*pyfr.backends.opencl.generator.OpenCLKernelGenerator method*), 71
- argspec() (*pyfr.backends.openmp.generator.OpenMPKernelGenerator method*), 71
 - block1d (*pyfr.backends.hip.generator.HIPKernelGenerator attribute*), 70
- aux_nregs (*pyfr.integrators.dual.pseudo.pseudocontrollers.DualNonePseudoController attribute*), 34
- aux_nregs (*pyfr.integrators.dual.pseudo.pseudocontrollers.DualPIPseudoController attribute*), 35
- aux_nregs (*pyfr.integrators.dual.pseudo.pseudosteppers.DualDenseRK34PseudoStepper attribute*), 44
- aux_nregs (*pyfr.integrators.dual.pseudo.pseudosteppers.DualEulerPseudoStepper attribute*), 47
- aux_nregs (*pyfr.integrators.dual.pseudo.pseudosteppers.DualRK34PseudoStepper attribute*), 47
- aux_nregs (*pyfr.integrators.dual.pseudo.pseudosteppers.DualRK45PseudoStepper attribute*), 48
- aux_nregs (*pyfr.integrators.dual.pseudo.pseudosteppers.DualRK4PseudoStepper attribute*), 45
- aux_nregs (*pyfr.integrators.dual.pseudo.pseudosteppers.DualTVDRK3PseudoStepper attribute*), 46
 - call_plugin_dt() (*pyfr.integrators.dual.phys.controllers.DualNoneController method*), 53
 - call_plugin_dt() (*pyfr.integrators.dual.phys.steppers.DualBackwardEulerStepper method*), 41

```
call_plugin_dt() (pyfr.integrators.dual.phys.steppers.SDIRK3Stepper.stats() (pyfr.integrators.dual.pseudo.pseudosteppers.DualRK4
    method), 42
call_plugin_dt() (pyfr.integrators.dual.phys.steppers.SDIRK4Stepper.stats() (pyfr.integrators.dual.pseudo.pseudosteppers.DualRK4
    method), 45
call_plugin_dt() (pyfr.integrators.std.controllers.StdNoneController.collect_stats() (pyfr.integrators.dual.pseudo.pseudosteppers.DualTVDF
    method), 31
call_plugin_dt() (pyfr.integrators.std.controllers.StdPICController.collect_stats() (pyfr.integrators.std.controllers.StdNoneController
    method), 31
call_plugin_dt() (pyfr.integrators.std.steppers.StdEulerStepper.collect_stats() (pyfr.integrators.std.controllers.StdPICController
    method), 32
call_plugin_dt() (pyfr.integrators.std.steppers.StdRK34Stepper.collect_stats() (pyfr.integrators.std.steppers.StdEulerStepper
    method), 36
call_plugin_dt() (pyfr.integrators.std.steppers.StdRK45Stepper.collect_stats() (pyfr.integrators.std.steppers.StdRK34Stepper
    method), 39
call_plugin_dt() (pyfr.integrators.std.steppers.StdRK45Stepper.collect_stats() (pyfr.integrators.std.steppers.StdRK45Stepper
    method), 37
call_plugin_dt() (pyfr.integrators.std.steppers.StdTVDRK3Stepper.collect_stats() (pyfr.integrators.std.steppers.StdRK45Stepper
    method), 40
cfgmeta (pyfr.integrators.dual.phys.controllers.DualNoneController.commit() (pyfr.integrators.std.steppers.StdTVDRK3Stepper
    property), 33
cfgmeta (pyfr.integrators.dual.phys.steppers.DualBackwardEulerStepper (pyfr.backends.cuda.base.CUDABackend
    property), 41
cfgmeta (pyfr.integrators.dual.phys.steppers.SDIRK33Stepper.commit() (pyfr.backends.hip.base.HIPBackend method),
    property), 42
cfgmeta (pyfr.integrators.dual.phys.steppers.SDIRK43Stepper.commit() (pyfr.backends.opencl.base.OpenCLBackend
    property), 42
cfgmeta (pyfr.integrators.std.controllers.StdNoneController.commit() (pyfr.backends.openmp.base.OpenMPBackend
    property), 31
cfgmeta (pyfr.integrators.std.controllers.StdPICController.compute_grads() (pyfr.solvers.aceuler.system.ACEulerSystem
    property), 32
cfgmeta (pyfr.integrators.std.steppers.StdEulerStepper.compute_grads() (pyfr.solvers.acnavstokes.system.ACNavierStokesSystem
    property), 36
cfgmeta (pyfr.integrators.std.steppers.StdRK34Stepper.compute_grads() (pyfr.solvers.euler.system.EulerSystem
    property), 38
cfgmeta (pyfr.integrators.std.steppers.StdRK45Stepper.compute_grads() (pyfr.solvers.navstokes.system.NavierStokesSystem
    property), 39
cfgmeta (pyfr.integrators.std.steppers.StdRK4Stepper.con_to_pri() (pyfr.solvers.aceuler.elements.ACEulerElements
    property), 37
cfgmeta (pyfr.integrators.std.steppers.StdTVDRK3Stepper.con_to_pri() (pyfr.solvers.acnavstokes.elements.ACNavierStokesElements
    property), 40
collect_stats() (pyfr.integrators.dual.phys.controllers.DualNoneController.con_to_pri() (pyfr.solvers.euler.elements.EulerElements
    method), 33
collect_stats() (pyfr.integrators.dual.phys.steppers.DualBackwardEulerStepper.con_to_pri() (pyfr.solvers.navstokes.elements.NavierStokesElements
    method), 41
collect_stats() (pyfr.integrators.dual.phys.steppers.SDIRK33Stepper.con_to_pri() (pyfr.backends.cuda.base.CUDABackend
    method), 42
collect_stats() (pyfr.integrators.dual.phys.steppers.SDIRK43Stepper.con_to_pri() (pyfr.backends.hip.base.HIPBackend
    method), 43
collect_stats() (pyfr.integrators.dual.pseudo.pseudosteppers.DualRK45Stepper.collect_stats() (pyfr.backends.opencl.base.OpenCLBackend
    method), 44
collect_stats() (pyfr.integrators.dual.pseudo.pseudosteppers.DualRK45Stepper.collect_stats() (pyfr.backends.openmp.base.OpenMPBackend
    method), 47
collect_stats() (pyfr.integrators.dual.pseudo.pseudosteppers.DualRK45Stepper.collect_stats() (pyfr.integrators.dual.phys.controllers.DualNoneController
    method), 48
```

controller_name (pyfr.integrators.std.controllers.StdNoneController attribute), 32	DualCoeffs (pyfr.solvers.aceuler.elements.ACEulerElements attribute), 54
controller_name (pyfr.integrators.std.controllers.StdPICController attribute), 32	DualCoeffs (pyfr.solvers.acnavstokes.elements.ACNavierStokesElements attribute), 55
controller_needs_errrest (pyfr.integrators.std.controllers.StdNoneController property), 32	dualcoeffs (pyfr.solvers.euler.elements.EulerElements attribute), 57
controller_needs_errrest (pyfr.integrators.std.controllers.StdPICController property), 32	dualcoeffs (pyfr.solvers.navstokes.elements.NavierStokesElements attribute), 58
controller_needs_errrest (pyfr.integrators.std.steppers.StdEulerStepper property), 36	DualDenseRKPseudoStepper (class in pyfr.integrators.dual.pseudo.pseudosteppers), 44
controller_needs_errrest (pyfr.integrators.std.steppers.StdRK34Stepper property), 38	DualEulerPseudoStepper (class in pyfr.integrators.dual.pseudo.pseudosteppers), 46
controller_needs_errrest (pyfr.integrators.std.steppers.StdRK45Stepper property), 39	DualNoneController (class in pyfr.integrators.dual.phys.controllers), 33
controller_needs_errrest (pyfr.integrators.std.steppers.StdRK4Stepper property), 37	DualNonePseudoController (class in pyfr.integrators.dual.pseudo.pseudocontrollers), 34
controller_needs_errrest (pyfr.integrators.std.steppers.StdTVDRK3Stepper property), 40	DualPIPseudoController (class in pyfr.integrators.dual.pseudo.pseudocontrollers), 35
convvarmap (pyfr.solvers.aceuler.elements.ACEulerElements attribute), 54	DualRK45PseudoStepper (class in pyfr.integrators.dual.pseudo.pseudosteppers), 47
convvarmap (pyfr.solvers.acnavstokes.elements.ACNavierStokesElements attribute), 55	DualRK4PseudoStepper (class in pyfr.integrators.dual.pseudo.pseudosteppers), 44
convmon() (pyfr.integrators.dual.pseudo.pseudocontrollers.DualNonePseudoController method), 34	DualTVDRK3PseudoStepper (class in pyfr.integrators.dual.pseudo.pseudosteppers), 48
convmon() (pyfr.integrators.dual.pseudo.pseudocontrollers.DualPIPseudoController method), 35	DualPIPseudoController
CUDABackend (class in pyfr.backends.cuda.base), 64	ele_scal_upts() (pyfr.solvers.aceuler.system.ACEulerSystem method), 50
CUDAKernelGenerator (class in pyfr.backends.cuda.generator), 70	ele_scal_upts() (pyfr.solvers.acnavstokes.system.ACNavierStokesSystem method), 51
CUDAPointwiseKernelProvider (class in pyfr.backends.cuda.provider), 67	ele_scal_upts() (pyfr.solvers.euler.system.EulerSystem method), 52
curved_smat_at() (pyfr.solvers.aceuler.elements.ACEulerElements method), 54	elementscls (pyfr.solvers.navstokes.system.NavierStokesSystem method), 53
curved_smat_at() (pyfr.solvers.acnavstokes.elements.ACNavierStokesElements method), 55	elementscls (pyfr.solvers.aceuler.system.ACEulerSystem attribute), 50
curved_smat_at() (pyfr.solvers.euler.elements.EulerElements method), 57	elementscls (pyfr.solvers.acnavstokes.system.ACNavierStokesSystem attribute), 51
curved_smat_at() (pyfr.solvers.navstokes.elements.NavierStokesElements method), 58	elementscls (pyfr.solvers.euler.system.EulerSystem attribute), 52
D	elementscls (pyfr.solvers.navstokes.system.NavierStokesSystem attribute), 53
DualBackwardEulerStepper (class in pyfr.integrators.dual.phys.steppers), 40	EulerElements (class in pyfr.solvers.euler.elements), 56
	EulerIntInters (class in pyfr.solvers.euler.inters), 61

EulerMPIInters (*class in pyfr.solvers.euler.inters*), 62
 EulerSystem (*class in pyfr.solvers.euler.system*), 51

F

filt() (*pyfr.solvers.aceuler.system.ACEulerSystem method*), 50
 filt() (*pyfr.solvers.acnavstokes.system.ACNavierStokesSystem method*), 51
 filt() (*pyfr.solvers.euler.system.EulerSystem method*), 52
 filt() (*pyfr.solvers.navstokes.system.NavierStokesSystem method*), 53

formulation (*pyfr.integrators.dual.phys.controllers.DualNoneController attribute*), 33
 formulation (*pyfr.integrators.dual.phys.steppers.DualBackwardEulerStepper attribute*), 41
 formulation (*pyfr.integrators.dual.phys.steppers.SDIRK33Stepper attribute*), 42
 formulation (*pyfr.integrators.dual.phys.steppers.SDIRK43Stepper attribute*), 43

G

getCartvisc_fpts_for_inter() (*pyfr.solvers.acnavstokes.elements.ACNavierStokesElements method*), 55
 get_artvisc_fpts_for_inter() (*pyfr.solvers.navstokes.elements.NavierStokesElements method*), 58
 get_ploc_for_inter() (*pyfr.solvers.aceuler.elements.ACEulerElements method*), 55
 get_ploc_for_inter() (*pyfr.solvers.dual.pseudo.pseudocontrollers.DualNoneController attribute*), 34
 get_ploc_for_inter() (*pyfr.solvers.dual.pseudo.pseudocontrollers.DualPIMETHODController attribute*), 35
 get_ploc_for_inter() (*pyfr.solvers.dual.pseudo.pseudosteppers.DualRK34Stepper attribute*), 44
 get_ploc_for_inter() (*pyfr.solvers.dual.pseudo.pseudosteppers.DualRK45Stepper attribute*), 47
 get_ploc_for_inter() (*pyfr.solvers.dual.pseudo.pseudosteppers.DualRK34Stepper attribute*), 48
 get_ploc_for_inter() (*pyfr.solvers.dual.pseudo.pseudosteppers.DualRK45Stepper attribute*), 49
 get_ploc_for_inter() (*pyfr.solvers.dual.pseudo.pseudosteppers.DualRK45Stepper attribute*), 45
 get_ploc_for_inter() (*pyfr.solvers.dual.pseudo.pseudosteppers.DualTVD34Stepper attribute*), 46
 get_plugin_data_prefix() (*pyfr.integrators.std.controllers.StdNoneController attribute*), 32
 get_plugin_data_prefix() (*pyfr.integrators.std.controllers.StdPIController attribute*), 32
 get_plugin_data_prefix() (*pyfr.integrators.std.steppers.StdEulerStepper attribute*), 36
 get_plugin_data_prefix() (*pyfr.integrators.std.steppers.StdRK34Stepper attribute*), 38
 get_plugin_data_prefix() (*pyfr.integrators.std.steppers.StdRK45Stepper attribute*), 39
 get_plugin_data_prefix() (*pyfr.integrators.std.steppers.StdRK4Stepper attribute*), 37
 get_plugin_data_prefix() (*pyfr.integrators.std.steppers.StdTVDRK3Stepper attribute*), 40
 formulations (*pyfr.solvers.aceuler.elements.ACEulerElements attribute*), 54
 formulations (*pyfr.solvers.acnavstokes.elements.ACNavierStokesElements attribute*), 55

fsal (*pyfr.integrators.dual.phys.steppers.DualBackwardEulerStepper attribute*), 41
 fsal (*pyfr.integrators.dual.phys.steppers.SDIRK33Stepper attribute*), 42
 fsal (*pyfr.integrators.dual.phys.steppers.SDIRK43Stepper attribute*), 43

get_plugin_data_prefix()
 (pyfr.integrators.std.steppers.StdRK45Stepper
 static method), 39

get_plugin_data_prefix()
 (pyfr.integrators.std.steppers.StdRK4Stepper
 static method), 37

get_plugin_data_prefix()
 (pyfr.integrators.std.steppers.StdTVDRK3Stepper
 static method), 40

get_pnorms() (pyfr.solvers.aceuler.elements.ACEulerElements
 method), 54

get_pnorms() (pyfr.solvers.acnavstokes.elements.ACNavierStokesElements
 method), 55

get_pnorms() (pyfr.solvers.euler.elements.EulerElements
 method), 57

get_pnorms() (pyfr.solvers.navstokes.elements.NavierStokesElements
 method), 58

get_pnorms_for_inter()
 (pyfr.solvers.aceuler.elements.ACEulerElements
 method), 54

get_pnorms_for_inter()
 (pyfr.solvers.acnavstokes.elements.ACNavierStokesElements
 method), 55

get_pnorms_for_inter()
 (pyfr.solvers.euler.elements.EulerElements
 method), 57

get_pnorms_for_inter()
 (pyfr.solvers.navstokes.elements.NavierStokesElements
 method), 58

get_scal_fpts_for_inter()
 (pyfr.solvers.aceuler.elements.ACEulerElements
 method), 54

get_scal_fpts_for_inter()
 (pyfr.solvers.acnavstokes.elements.ACNavierStokesElements
 method), 55

get_scal_fpts_for_inter()
 (pyfr.solvers.euler.elements.EulerElements
 method), 57

get_scal_fpts_for_inter()
 (pyfr.solvers.navstokes.elements.NavierStokesElements
 method), 58

get_vect_fpts_for_inter()
 (pyfr.solvers.aceuler.elements.ACEulerElements
 method), 54

get_vect_fpts_for_inter()
 (pyfr.solvers.acnavstokes.elements.ACNavierStokesElements
 method), 56

get_vect_fpts_for_inter()
 (pyfr.solvers.euler.elements.EulerElements
 method), 57

get_vect_fpts_for_inter()
 (pyfr.solvers.navstokes.elements.NavierStokesElements
 method), 58

grad_con_to_pri() (pyfr.solvers.acnavstokes.elements.ACNavierStokesElements
 method), 56

grad_con_to_pri() (pyfr.solvers.navstokes.elements.NavierStokesElements
 method), 58

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.SDIRK33Stepper
 property), 42

grad_soln(pyfr.integrators.dual.phys.steppers.SDIRK43Stepper
 property), 43

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), 38

grad_soln(pyfr.integrators.std.steppers.StdRK45Stepper
 property), 39

grad_soln(pyfr.integrators.std.steppers.StdRK4Stepper
 property), 37

grad_soln(pyfr.integrators.std.steppers.StdTVDRK3Stepper
 property), 40

graph() (pyfr.backends.cuda.base.CUDABackend
 method), 64

graph() (pyfr.backends.hip.base.HIPBackend
 method), 65

graph() (pyfr.backends.opencl.base.OpenCLBackend
 method), 65

graph() (pyfr.backends.openmp.base.OpenMPBackend
 method), 66

H

HIPBackend (class in pyfr.backends.hip.base), 64

HIPKernelGenerator (class in pyfr.backends.hip.generator), 70

HIPPointwiseKernelProvider (class in pyfr.backends.hip.provider), 68

I

init_stage() (pyfr.integrators.dual.pseudo.pseudocontrollers.DualNonePseudoController
 method), 34

init_stage() (pyfr.integrators.dual.pseudo.pseudocontrollers.DualPICPseudoController
 method), 35

init_stage() (pyfr.integrators.dual.pseudo.pseudosteppers.DualDenseRK34PseudoStepper
 method), 44

init_stage() (pyfr.integrators.dual.pseudo.pseudosteppers.DualEulerPseudoStepper
 method), 47

init_stage() (pyfr.integrators.dual.pseudo.pseudosteppers.DualRK34PseudoStepper
 method), 48

init_stage() (pyfr.integrators.dual.pseudo.pseudosteppers.DualRK45PseudoStepper
 method), 49

`init_stage()` (`pyfr.integrators.dual.pseudo.pseudostepper`)
 `DualTVDRK4PseudoStepper.openmp.base.OpenMPBackend`
 `method`), 45
`init_stage()` (`pyfr.integrators.dual.pseudo.pseudosteppers.DualTVDRK3PseudoStepper`)
 `M`
`intinterscls` (`pyfr.solvers.aceuler.system.ACEulerSystem`)
 `malloc()` (`pyfr.backends.cuda.base.CUDABackend`
 `attribute`), 50
`intinterscls` (`pyfr.solvers.acnavstokes.system.ACNavierStokesSystem`)
 `malloc()` (`pyfr.backends.hip.base.HIPBackend` method),
 `attribute`), 51
`intinterscls` (`pyfr.solvers.euler.system.EulerSystem`)
 `attribute`), 52
`intinterscls` (`pyfr.solvers.navstokes.system.NavierStokesSystem`)
 `malloc()` (`pyfr.backends.openmp.base.OpenMPBackend`
 `attribute`), 53

K
`kernel()` (`pyfr.backends.cuda.base.CUDABackend`
 `method`), 64
`kernel()` (`pyfr.backends.hip.base.HIPBackend` method),
 `65`
`kernel()` (`pyfr.backends.opencl.base.OpenCLBackend`
 `method`), 65
`kernel()` (`pyfr.backends.openmp.base.OpenMPBackend`
 `method`), 66
`kernel_generator_cls`
 `(pyfr.backends.cuda.provider.CUDAPointwiseKernelProvider` method), 65
 `attribute`), 67
`kernel_generator_cls`
 `(pyfr.backends.hip.provider.HIPPointwiseKernelProvider` attribute), 68
`kernel_generator_cls`
 `(pyfr.backends.opencl.provider.OpenCLPointwiseKernelProvider` attribute), 68
`kernel_generator_cls`
 `(pyfr.backends.openmp.provider.OpenMPPointwiseKernelProvider` attribute), 68
`krunner` (`pyfr.backends.openmp.base.OpenMPBackend`
 `property`), 66

L

`ldim_size()` (`pyfr.backends.cuda.generator.CUDAKernelGenerator`)
 `name` (`pyfr.backends.cuda.base.CUDABackend` attribute),
 `method`), 70
`ldim_size()` (`pyfr.backends.hip.generator.HIPKernelGenerator`)
 `name` (`pyfr.backends.hip.base.HIPBackend` attribute), 65
 `method`), 70
`ldim_size()` (`pyfr.backends.opencl.generator.OpenCLKernelGenerator`)
 `attribute`), 65
 `method`), 71
`ldim_size()` (`pyfr.backends.openmp.generator.OpenMPKernelGenerator`)
 `attribute`), 66
 `method`), 71
`localerrest()` (`pyfr.integrators.dual.pseudo.pseudocontrollers.DualFIRuleController`
 `method`), 35
`lookup` (`pyfr.backends.cuda.base.CUDABackend` prop-
 `erty`), 64
`lookup` (`pyfr.backends.hip.base.HIPBackend` property),
 `65`
`lookup` (`pyfr.backends.opencl.base.OpenCLBackend`
 `property`), 65

N
`mpiinterscls` (`pyfr.solvers.aceuler.system.ACEulerSystem`)
 `name` (`pyfr.backends.openmp.base.OpenMPBackend` attribute),
 `method`), 50
`mpiinterscls` (`pyfr.solvers.acnavstokes.system.ACNavierStokesSystem`)
 `name` (`pyfr.backends.openmp.base.OpenMPBackend` attribute), 51
`mpiinterscls` (`pyfr.solvers.euler.system.EulerSystem` at-
 `tribute`), 52
`mpiinterscls` (`pyfr.solvers.navstokes.system.NavierStokesSystem`)
 `name` (`pyfr.backends.openmp.base.OpenMPBackend` attribute), 53

N
`NavierStokesElements` (class
 `in`
 `pyfr.solvers.navstokes.elements`), 57

O

- NavierStokesIntInters (class in `pyfr.solvers.navstokes.inters`), 62
- NavierStokesMPIIInters (class in `pyfr.solvers.navstokes.inters`), 63
- NavierStokesSystem (class in `pyfr.solvers.navstokes.system`), 52
- needs_1dim() (`pyfr.backends.cuda.generator.CUDAKernelGenerator` method), 44
- needs_1dim() (`pyfr.backends.hip.generator.HIPKernelGenerator` method), 47
- needs_1dim() (`pyfr.backends.opencl.generator.OpenCLKernelGenerator` method), 48
- needs_1dim() (`pyfr.backends.openmp.generator.OpenMPKernelGenerator` method), 49
- nstages (`pyfr.integrators.dual.phys.steppers.DualBackwardEulerStepper` attribute), 41
- nstages (`pyfr.integrators.dual.phys.steppers.SDIRK33Stepper` attribute), 42
- nstages (`pyfr.integrators.dual.phys.steppers.SDIRK43Stepper` attribute), 43
- nsteps (`pyfr.integrators.dual.phys.controllers.DualNoneController` property), 33
- nsteps (`pyfr.integrators.dual.phys.steppers.DualBackwardEulerStepper` property), 41
- nsteps (`pyfr.integrators.dual.phys.steppers.SDIRK33Stepper` property), 42
- nsteps (`pyfr.integrators.dual.phys.steppers.SDIRK43Stepper` property), 43
- nsteps (`pyfr.integrators.std.controllers.StdNoneController` property), 32
- nsteps (`pyfr.integrators.std.controllers.StdPIController` property), 32
- nsteps (`pyfr.integrators.std.steppers.StdEulerStepper` property), 36
- nsteps (`pyfr.integrators.std.steppers.StdRK34Stepper` property), 38
- nsteps (`pyfr.integrators.std.steppers.StdRK45Stepper` property), 39
- nsteps (`pyfr.integrators.std.steppers.StdRK4Stepper` property), 37
- nsteps (`pyfr.integrators.std.steppers.StdTVDRK3Stepper` property), 40
- ntotiters (`pyfr.integrators.dual.pseudo.pseudosteppers.DualDenseRK3PseudoStepper` property), 44
- ntotiters (`pyfr.integrators.dual.pseudo.pseudosteppers.DualEulerPseudoStepper` property), 47
- ntotiters (`pyfr.integrators.dual.pseudo.pseudosteppers.DualRK34PseudoStepper` property), 48
- ntotiters (`pyfr.integrators.dual.pseudo.pseudosteppers.DualRK45PseudoStepper` property), 49
- ntotiters (`pyfr.integrators.dual.pseudo.pseudosteppers.DualRK4PseudoStepper` property), 45
- ntotiters (`pyfr.integrators.dual.pseudo.pseudosteppers.DualTVDRK3PseudoStepper` property), 46

P

- obtain_solution() (`pyfr.integrators.dual.pseudo.pseudocontrollers.DualEulerController` method), 34
- obtain_solution() (`pyfr.integrators.dual.pseudo.pseudocontrollers.DualRK34Controller` method), 35
- obtain_solution() (`pyfr.integrators.dual.pseudo.pseudosteppers.DualEulerStepper` method), 44
- obtain_solution() (`pyfr.integrators.dual.pseudo.pseudosteppers.DualRK34Stepper` method), 47
- obtain_solution() (`pyfr.integrators.dual.pseudo.pseudosteppers.DualRK45Stepper` method), 48
- obtain_solution() (`pyfr.integrators.dual.pseudo.pseudosteppers.DualTVDRK3Stepper` method), 49
- OpenCLBackend (class in `pyfr.backends.opencl.base`), 65
- OpenCLKernelGenerator (class in `pyfr.backends.opencl.generator`), 70
- OpenCLPointwiseKernelProvider (class in `pyfr.backends.opencl.provider`), 68
- OpenMPBackend (class in `pyfr.backends.openmp.base`), 66
- OpenMPKernelGenerator (class in `pyfr.backends.openmp.generator`), 71
- OpenMPPointwiseKernelProvider (class in `pyfr.backends.openmp.provider`), 68
- opmat() (`pyfr.solvers.aceuler.elements.ACEulerElements` method), 54
- opmat() (`pyfr.solvers.acnavstokes.elements.ACNavierStokesElements` method), 56
- opmat() (`pyfr.solvers.euler.elements.EulerElements` method), 57
- opmat() (`pyfr.solvers.navstokes.elements.NavierStokesElements` method), 58
- ordered_meta_kernel() (`pyfr.backends.cuda.base.CUDABackend` method), 64
- ordered_meta_kernel() (`pyfr.backends.hip.base.HIPBackend` method), 65
- ordered_meta_kernel() (`pyfr.backends.opencl.base.OpenCLBackend` method), 65
- ordered_meta_kernel() (`pyfr.backends.openmp.base.OpenMPBackend` method), 66

```
ploc_at() (pyfr.solvers.euler.elements.EulerElements pseudo_advance() (pyfr.integrators.dual.pseudo.pseudocontrollers.DualPseudoController
           method), 57                                         method), 35
ploc_at() (pyfr.solvers.navstokes.elements.NavierStokesElements pseudo_controller_name
           method), 58                                         (pyfr.integrators.dual.pseudo.pseudocontrollers.DualNonePseudoController
                                                       attribute), 35
ploc_at_np() (pyfr.solvers.aceuler.elements.ACEulerElements pseudo_controller_name
              method), 54                                         attribute), 35
ploc_at_np() (pyfr.solvers.acnavstokes.elements.ACNavierStokesElements pseudo_controller_name
              method), 56                                         (pyfr.integrators.dual.pseudo.pseudocontrollers.DualPIPpseudoController
                                                       attribute), 35
ploc_at_np() (pyfr.solvers.euler.elements.EulerElements pseudo_controller_needs_lerrest
              method), 57                                         (pyfr.integrators.dual.pseudo.pseudocontrollers.DualNonePseudoController
                                                       attribute), 35
ploc_at_np() (pyfr.solvers.navstokes.elements.NavierStokesElements pseudo_controller_needs_lerrest
              method), 58                                         attribute), 35
plocfpts (pyfr.solvers.aceuler.elements.ACEulerElements pseudo_controller_needs_lerrest
           property), 54                                         (pyfr.integrators.dual.pseudo.pseudocontrollers.DualPIPpseudoController
                                                       attribute), 35
plocfpts (pyfr.solvers.acnavstokes.elements.ACNavierStokesElements pseudo_controller_needs_lerrest
           property), 56                                         (pyfr.integrators.dual.pseudo.pseudosteppers.DualEulerPseudoStepper
                                                       attribute), 47
plocfpts (pyfr.solvers.euler.elements.EulerElements pseudo_controller_needs_lerrest
           property), 57                                         attribute), 47
plocfpts (pyfr.solvers.navstokes.elements.NavierStokesElements pseudo_controller_needs_lerrest
           property), 58                                         (pyfr.integrators.dual.pseudo.pseudosteppers.DualRK34PseudoStepper
                                                       attribute), 48
prepare() (pyfr.solvers.aceuler.inters.ACEulerIntInters pseudo_stepper_has_lerrest
           method), 60                                         (pyfr.integrators.dual.pseudo.pseudosteppers.DualRK45PseudoStepper
                                                       attribute), 49
prepare() (pyfr.solvers.aceuler.inters.ACEulerMPIInters pseudo_stepper_has_lerrest
           method), 60                                         property), 49
prepare() (pyfr.solvers.acnavstokes.inters.ACNavierStokesIntInters pseudo_stepper_has_lerrest
           method), 61                                         (pyfr.integrators.dual.pseudo.pseudosteppers.DualRK4PseudoStepper
                                                       attribute), 45
prepare() (pyfr.solvers.acnavstokes.inters.ACNavierStokesIntInters pseudo_stepper_has_lerrest
           method), 61                                         (pyfr.integrators.dual.pseudo.pseudosteppers.DualTVDRK3PseudoStepper
                                                       attribute), 46
prepare() (pyfr.solvers.euler.inters.EulerIntInters pseudo_stepper_name
           method), 62                                         attribute), 46
prepare() (pyfr.solvers.euler.inters.EulerMPIInters pseudo_stepper_name
           method), 62                                         (pyfr.integrators.dual.pseudo.pseudosteppers.DualEulerPseudoStepper
                                                       attribute), 47
prepare() (pyfr.solvers.navstokes.inters.NavierStokesIntInters pseudo_stepper_name
           method), 62                                         (pyfr.integrators.dual.pseudo.pseudosteppers.DualRK34PseudoStepper
                                                       attribute), 48
prepare() (pyfr.solvers.navstokes.inters.NavierStokesMPIInters pseudo_stepper_name
           method), 63                                         attribute), 48
pri_to_con() (pyfr.solvers.aceuler.elements.ACEulerElements pseudo_stepper_name
              static method), 54                                         (pyfr.integrators.dual.pseudo.pseudosteppers.DualRK45PseudoStepper
                                                       attribute), 49
pri_to_con() (pyfr.solvers.acnavstokes.elements.ACNavierStokesElements pseudo_stepper_name
              static method), 56                                         (pyfr.integrators.dual.pseudo.pseudosteppers.DualRK4PseudoStepper
                                                       attribute), 45
pri_to_con() (pyfr.solvers.euler.elements.EulerElements pseudo_stepper_name
              static method), 57                                         attribute), 45
pri_to_con() (pyfr.solvers.navstokes.elements.NavierStokesElements pseudo_stepper_nfevals
              static method), 58                                         (pyfr.integrators.dual.pseudo.pseudosteppers.DualTVDRK3PseudoStepper
                                                       attribute), 46
privarmap (pyfr.solvers.aceuler.elements.ACEulerElements pseudo_stepper_nfevals
           attribute), 54                                         (pyfr.integrators.dual.pseudo.pseudosteppers.DualDenseRKPseudoStepper
                                                       attribute), 44
privarmap (pyfr.solvers.acnavstokes.elements.ACNavierStokesElements pseudo_stepper_nfevals
           attribute), 56                                         property), 44
privarmap (pyfr.solvers.euler.elements.EulerElements pseudo_stepper_nfevals
           attribute), 57                                         attribute), 47
privarmap (pyfr.solvers.navstokes.elements.NavierStokesElements pseudo_stepper_nfevals
           attribute), 58                                         (pyfr.integrators.dual.pseudo.pseudosteppers.DualRK34PseudoStepper
                                                       attribute), 48
pseudo_advance() (pyfr.integrators.dual.pseudo.pseudocontrollers.DualNonePseudoController
                  method), 34                                         pseudo_stepper_nfevals
```

(`pyfr.integrators.dual.pseudo.pseudosteppers.DualRK45PseudoStepper`
`property`), 49

`pseudo stepper_nfevals`
 (`pyfr.integrators.dual.pseudo.pseudosteppers.DualRK45PseudoStepper`
`property`), 45

`pseudo stepper_nfevals`
 (`pyfr.integrators.dual.pseudo.pseudosteppers.DualTVDRK3PseudoStepper`
`property`), 46

`pseudo stepper_nregs`
 (`pyfr.integrators.dual.pseudo.pseudosteppers.DualEulerPseudoStepper`
`attribute`), 47

`pseudo stepper_nregs`
 (`pyfr.integrators.dual.pseudo.pseudosteppers.DualRK34PseudoStepper`
`property`), 48

`pseudo stepper_nregs`
 (`pyfr.integrators.dual.pseudo.pseudosteppers.DualRK45PseudoStepper`
`property`), 49

`pseudo stepper_nregs`
 (`pyfr.integrators.dual.pseudo.pseudosteppers.DualTVDRK3PseudoStepper`
`attribute`), 45

`pseudo stepper_nregs`
 (`pyfr.integrators.dual.pseudo.pseudosteppers.DualEulerPseudoStepper`
`attribute`), 46

`pseudo stepper_order`
 (`pyfr.integrators.dual.pseudo.pseudosteppers.DualEulerPseudoStepper`
`attribute`), 47

`pseudo stepper_order`
 (`pyfr.integrators.dual.pseudo.pseudosteppers.DualRK34PseudoStepper`
`attribute`), 48

`pseudo stepper_order`
 (`pyfr.integrators.dual.pseudo.pseudosteppers.DualRK45PseudoStepper`
`attribute`), 49

`pseudo stepper_order`
 (`pyfr.integrators.dual.pseudo.pseudosteppers.DualRK4PseudoStepper`
`attribute`), 45

`pseudo stepper_order`
 (`pyfr.integrators.dual.pseudo.pseudosteppers.DualTVDRK3PseudoStepper`
`attribute`), 46

`pseudostepinfo` (`pyfr.integrators.dual.phys.controllers.DualNoneController`
`property`), 33

`pseudostepinfo` (`pyfr.integrators.dual.phys.steppers.DualBackwardEulerStepper`
`property`), 41

`pseudostepinfo` (`pyfr.integrators.dual.phys.steppers.SDIRK33Stepper`
`property`), 42

`pseudostepinfo` (`pyfr.integrators.dual.phys.steppers.SDIRK43Stepper`
`property`), 43

Q

`qpts` (`pyfr.solvers.aceuler.elements.ACEulerElements`
`property`), 54

`qpts` (`pyfr.solvers.acnavstokes.elements.ACNavierStokesElements`
`property`), 56

`qpts` (`pyfr.solvers.euler.elements.EulerElements`
`property`), 57

R

`rcpdjac_at()` (`pyfr.solvers.aceuler.elements.ACEulerElements`
`method`), 55

`rcpdjac_at()` (`pyfr.solvers.acnavstokes.elements.ACNavierStokesElements`
`method`), 56

`rcpdjac_at()` (`pyfr.solvers.euler.elements.EulerElements`
`method`), 57

`rcpdjac_at()` (`pyfr.solvers.navstokes.elements.NavierStokesElements`
`method`), 58

`rcpdjac_at_np()` (`pyfr.solvers.acnavstokes.elements.ACNavierStokesElements`
`method`), 56

`rcpdjac_at_np()` (`pyfr.solvers.euler.elements.EulerElements`
`method`), 57

`rcpdjac_at_np()` (`pyfr.solvers.navstokes.elements.NavierStokesElements`
`method`), 58

`register()` (`pyfr.backends.cuda.provider.CUDAPointwiseKernelProvider`
`method`), 68

`register()` (`pyfr.backends.hip.provider.HIPPointwiseKernelProvider`
`method`), 68

`register()` (`pyfr.backends.openmp.provider.OpenMPPointwiseKernelProvider`
`method`), 68

`render()` (`pyfr.backends.cuda.generator.CUDAKernelGenerator`
`method`), 70

`render()` (`pyfr.backends.hip.generator.HIPKernelGenerator`
`method`), 70

`render()` (`pyfr.backends.opencl.generator.OpenCLKernelGenerator`
`method`), 71

`render()` (`pyfr.backends.openmp.generator.OpenMPKernelGenerator`
`method`), 71

`rhs()` (`pyfr.solvers.aceuler.system.ACEulerSystem`
`method`), 50

`rhs()` (`pyfr.solvers.acnavstokes.system.ACNavierStokesSystem`
`method`), 51

`rhs()` (`pyfr.solvers.euler.system.EulerSystem`
`method`), 52

`rhs()` (`pyfr.solvers.navstokes.system.NavierStokesSystem`
`method`), 53

`rhs_wait_times()` (`pyfr.solvers.aceuler.system.ACEulerSystem`
`method`), 51

`rhs_wait_times()` (`pyfr.solvers.acnavstokes.system.ACNavierStokesSystem`
`method`), 51

`rhs_wait_times()` (`pyfr.solvers.euler.system.EulerSystem`
`method`), 52

`rhs_wait_times()` (`pyfr.solvers.navstokes.system.NavierStokesSystem`
`method`), 53

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

```
run() (pyfr.integrators.dual.phys.steppers.DualBackwardEulerStepper.set_ics_from_cfg() (pyfr.solvers.euler.elements.EulerElements
      method), 41                                         method), 57
run() (pyfr.integrators.dual.phys.steppers.SDIRK33Stepper.set_ics_from_cfg() (pyfr.solvers.navstokes.elements.NavierStokesElements
      method), 42                                         method), 59
run() (pyfr.integrators.dual.phys.steppers.SDIRK43Stepper.set_ics_from_soln()
      method), 43                                         (pyfr.solvers.aceuler.elements.ACEulerElements
      method), 55                                         method), 55
run() (pyfr.integrators.std.controllers.StdNoneController
      method), 32                                         set_ics_from_soln()
run() (pyfr.integrators.std.controllers.StdPIController
      method), 33                                         (pyfr.solvers.acnavstokes.elements.ACNavierStokesElements
      method), 56                                         method), 56
run() (pyfr.integrators.std.steppers.StdEulerStepper
      method), 37                                         set_ics_from_soln()
run() (pyfr.integrators.std.steppers.StdRK34Stepper
      method), 38                                         (pyfr.solvers.euler.elements.EulerElements
      method), 57                                         method), 57
run() (pyfr.integrators.std.steppers.StdRK45Stepper
      method), 39                                         set_ics_from_soln()
run() (pyfr.integrators.std.steppers.StdRK4Stepper
      method), 37                                         (pyfr.solvers.navstokes.elements.NavierStokesElements
      method), 59                                         method), 59
run() (pyfr.integrators.std.steppers.StdTVDRK3Stepper
      method), 40                                         shockvar(pyfr.solvers.navstokes.elements.NavierStokesElements
      method), 59                                         attribute), 59
run_graph() (pyfr.backends.cuda.base.CUDABackend sliceat() (pyfr.solvers.aceuler.elements.ACEulerElements
      method), 64                                         method), 55
run_graph() (pyfr.backends.hip.base.HIPBackend sliceat() (pyfr.solvers.acnavstokes.elements.ACNavierStokesElements
      method), 65                                         method), 56
run_graph() (pyfr.backends.opencl.base.OpenCLBackend smat_at_np() (pyfr.solvers.aceuler.elements.ACEulerElements
      method), 66                                         method), 55
run_graph() (pyfr.backends.openmp.base.OpenMPBackend smat_at_np() (pyfr.solvers.acnavstokes.elements.ACNavierStokesElements
      method), 66                                         method), 56
run_kernels() (pyfr.backends.cuda.base.CUDABackend smat_at_np() (pyfr.solvers.euler.elements.EulerElements
      method), 64                                         method), 57
run_kernels() (pyfr.backends.hip.base.HIPBackend smat_at_np() (pyfr.solvers.euler.elements.EulerElements
      method), 65                                         method), 57
run_kernels() (pyfr.backends.opencl.base.OpenCLBackend smat_at_np() (pyfr.solvers.navstokes.elements.NavierStokesElements
      method), 66                                         method), 59
run_kernels() (pyfr.backends.openmp.base.OpenMPBackend soln(pyfr.integrators.dual.phys.controllers.DualNoneController
      method), 66                                         property), 33
                                                    soln(pyfr.integrators.dual.phys.steppers.DualBackwardEulerStepper
                                                    property), 41
S
SDIRK33Stepper          (class           in  soln(pyfr.integrators.dual.phys.steppers.SDIRK33Stepper
      pyfr.integrators.dual.phys.steppers), 41           property), 42
SDIRK43Stepper          (class           in  soln(pyfr.integrators.dual.phys.steppers.SDIRK43Stepper
      pyfr.integrators.dual.phys.steppers), 42           property), 43
set_backend() (pyfr.solvers.aceuler.elements.ACEulerElements soln(pyfr.integrators.std.controllers.StdNoneController
      method), 55                                         property), 32
set_backend() (pyfr.solvers.acnavstokes.elements.ACNavierStokesElements soln(pyfr.integrators.std.controllers.StdPIController
      method), 56                                         property), 33
set_backend() (pyfr.solvers.euler.elements.EulerElements soln(pyfr.integrators.std.steppers.StdEulerStepper prop-
      method), 57                                         erty), 37
set_backend() (pyfr.solvers.navstokes.elements.NavierStokesElements soln(pyfr.integrators.std.steppers.StdRK34Stepper prop-
      method), 59                                         erty), 38
set_ics_from_cfg() (pyfr.solvers.aceuler.elements.ACEulerElements soln(pyfr.integrators.std.steppers.StdRK45Stepper prop-
      method), 55                                         erty), 39
set_ics_from_cfg() (pyfr.solvers.acnavstokes.elements.ACNavierStokesElements soln(pyfr.integrators.std.steppers.StdRK4Stepper prop-
      method), 56                                         erty), 37
```

```

soln (pyfr.integrators.std.steppers.StdTVDRK3Stepper step() (pyfr.integrators.std.steppers.StdTVDRK3Stepper
    property), 40
stage_nregs (pyfr.integrators.dual.phys.steppers.DualBackwardEulerStepper_has_error (pyfr.integrators.std.steppers.StdEulerStepper
    property), 41
stage_nregs (pyfr.integrators.dual.phys.steppers.SDIRK33Stepper_has_error (pyfr.integrators.std.steppers.StdRK34Stepper
    property), 42
stage_nregs (pyfr.integrators.dual.phys.steppers.SDIRK43Stepper_has_error (pyfr.integrators.std.steppers.StdRK45Stepper
    property), 43
StdEulerStepper (class in pyfr.integrators.std.steppers), 36
StdNoneController (class in pyfr.integrators.std.controllers), 31
StdPIController (class in pyfr.integrators.std.controllers), 32
StdRK34Stepper (class in pyfr.integrators.std.steppers), 38
StdRK45Stepper (class in pyfr.integrators.std.steppers), 39
StdRK4Stepper (class in pyfr.integrators.std.steppers), 37
StdTVDRK3Stepper (class in pyfr.integrators.std.steppers), 40
step() (pyfr.integrators.dual.phys.controllers.DualNoneController stepper_name (pyfr.integrators.std.steppers.StdRK45Stepper
    method), 33
step() (pyfr.integrators.dual.phys.steppers.DualBackwardEulerStepper stepper_name (pyfr.integrators.std.steppers.StdRK4Stepper
    method), 41
step() (pyfr.integrators.dual.phys.steppers.SDIRK33Stepper stepper_name (pyfr.integrators.std.steppers.StdTVDRK3Stepper
    method), 42
step() (pyfr.integrators.dual.phys.steppers.SDIRK43Stepper stepper_nregs (pyfr.integrators.dual.phys.steppers.DualBackwardEulerStepper
    method), 43
step() (pyfr.integrators.dual.pseudo.pseudosteppers.DualDenseStepper stepper_nregs (pyfr.integrators.dual.phys.steppers.SDIRK33Stepper
    method), 44
step() (pyfr.integrators.dual.pseudo.pseudosteppers.DualEulerStepper stepper_nregs (pyfr.integrators.dual.phys.steppers.SDIRK43Stepper
    method), 47
step() (pyfr.integrators.dual.pseudo.pseudosteppers.DualRK34Stepper stepper_nregs (pyfr.integrators.std.steppers.StdEulerStepper
    method), 48
step() (pyfr.integrators.dual.pseudo.pseudosteppers.DualRK34Stepper stepper_nregs (pyfr.integrators.std.steppers.StdRK34Stepper
    method), 49
step() (pyfr.integrators.dual.pseudo.pseudosteppers.DualRK45Stepper stepper_nregs (pyfr.integrators.std.steppers.StdRK45Stepper
    method), 45
step() (pyfr.integrators.dual.pseudo.pseudosteppers.DualTVDRK3Stepper stepper_nregs (pyfr.integrators.std.steppers.StdRK4Stepper
    method), 46
step() (pyfr.integrators.std.controllers.StdNoneController stepper_nregs (pyfr.integrators.std.steppers.StdTVDRK3Stepper
    method), 32
step() (pyfr.integrators.std.controllers.StdPIController stepper_order (pyfr.integrators.std.steppers.StdEulerStepper
    method), 33
step() (pyfr.integrators.std.steppers.StdEulerStepper stepper_order (pyfr.integrators.std.steppers.StdRK34Stepper
    method), 37
step() (pyfr.integrators.std.steppers.StdRK34Stepper stepper_order (pyfr.integrators.std.steppers.StdRK45Stepper
    method), 38
step() (pyfr.integrators.std.steppers.StdRK45Stepper stepper_order (pyfr.integrators.std.steppers.StdRK4Stepper
    method), 39
step() (pyfr.integrators.std.steppers.StdRK4Stepper stepper_order (pyfr.integrators.std.steppers.StdTVDRK3Stepper
    method), 37

```

V

store_current_soln()
(*pyfr.integrators.dual.pseudo.pseudocontrollers.DualNonePseudoController*.view(*pyfr.backends.cuda.base.CUDABackend*.method), 35
store_current_soln()
(*pyfr.integrators.dual.pseudo.pseudocontrollers.DualPseudoController*.view(*pyfr.backends.hip.base.HIPBackend*.method), 64
store_current_soln()
(*pyfr.integrators.dual.pseudo.pseudocontrollers.DualPseudoController*.view(*pyfr.backends.opencl.base.OpenCLBackend*.method), 66
store_current_soln()
(*pyfr.integrators.dual.pseudo.pseudosteppers.DualDenseRK45PseudoStepper*.view(), 60
store_current_soln()
(*pyfr.integrators.dual.pseudo.pseudosteppers.DualEulerPseudoStepper*.view(), 55
store_current_soln()
(*pyfr.integrators.dual.pseudo.pseudosteppers.DualEulerPseudoStepper*.visvarmap(*pyfr.solvers.acnavstokes.elements.ACNavierStokesElements*.attribute), 56
store_current_soln()
(*pyfr.integrators.dual.pseudo.pseudosteppers.DualEulerPseudoStepper*.visvarmap(*pyfr.solvers.euler.elements.EulerElements*.attribute), 57
store_current_soln()
(*pyfr.integrators.dual.pseudo.pseudosteppers.DualRK34PseudoStepper*.visvarmap(*pyfr.solvers.navstokes.elements.NavierStokesElements*.attribute), 59
store_current_soln()
(*pyfr.integrators.dual.pseudo.pseudosteppers.DualRK45PseudoStepper*.visvarmap(*pyfr.solvers.navstokes.elements.NavierStokesElements*.attribute), 59

X

store_current_soln()
(*pyfr.integrators.dual.pseudo.pseudosteppers.DualRK45PseudoStepper*.xchg_matrix(*pyfr.backends.cuda.base.CUDABackend*.method), 64
store_current_soln()
(*pyfr.integrators.dual.pseudo.pseudosteppers.DualTVDRK3PseudoStepper*.xchg_matrix(*pyfr.backends.hip.base.HIPBackend*.method), 65
store_current_soln()
(*pyfr.integrators.dual.pseudo.pseudosteppers.DualTVDRK3PseudoStepper*.xchg_matrix(*pyfr.backends.opencl.base.OpenCLBackend*.method), 66
system(*pyfr.integrators.dual.phys.controllers.DualNoneController*.xchg_matrix(*pyfr.backends.openmp.base.OpenMPBackend*.method), 33
system(*pyfr.integrators.dual.phys.steppers.DualBackwardEulerStepper*.xchg_matrix_for_view(),
property), 41
system(*pyfr.integrators.dual.phys.steppers.SDIRK33Stepper*.xchg_matrix_for_view(),
property), 42
system(*pyfr.integrators.dual.phys.steppers.SDIRK43Stepper*.xchg_matrix_for_view(),
property), 43

U

unordered_meta_kernel()
(*pyfr.backends.cuda.base.CUDABackend*.method), 64
unordered_meta_kernel()
(*pyfr.backends.hip.base.HIPBackend*.method), 65
unordered_meta_kernel()
(*pyfr.backends.opencl.base.OpenCLBackend*.method), 66
unordered_meta_kernel()
(*pyfr.backends.openmp.base.OpenMPBackend*.method), 66
upts (*pyfr.solvers.aceler.elements.ACEulerElements*.property), 55
upts (*pyfr.solvers.acnavstokes.elements.ACNavierStokesElements*.property), 56
upts (*pyfr.solvers.euler.elements.EulerElements*.property), 57
upts (*pyfr.solvers.navstokes.elements.NavierStokesElements*.property), 59