Boundary Conditions - OpenFOAM-2.3.0

김병윤
넥스트폼 대표이사

Open Source CFD Consulting

NEXT$f$oam
153-790, 서울특별시 금천구 가산동 갑을그레이트밸리 A동 1106호

February 2014

차 례

1 Derived boundary conditions

1.1 activeBaffleVelocity

This velocity boundary condition simulates the opening of a baffle due to local flow conditions, by merging the behaviours of wall and cyclic conditions. The baffle joins two mesh regions, where the open fraction determines the interpolation weights applied to each cyclic- and neighbour-patch contribution.

We determine whether the baffle is opening or closing from the sign of the net force across the baffle, from which the baffle open fraction is updated using:

$$x = x_{old} + sign(F_{net})\frac{dt}{DT} \tag{1.1}$$

x : baffle open fraction [0-1]
x_{old} : baffle open fraction on previous evaluation
dt : simulation time step
DT : time taken to open the baffle
F_{net} : net force across the baffle

Property	Description	Required	Default value
p	pressure field name	no	p
cyclicPatch	cylclic patch name	yes	
orientation	1 or -1 used to switch flow direction	yes	
openFraction	current opatch open fraction [0-1]	yes	
openingTime	time taken to open the baffle	yes	
maxOpenFractionDelta	max open fraction change per timestep	yes	

Example

```
myPatch
{
    type            activeBaffleVelocity;
    p               p;
    cyclicPatch     cyclic1;
    orientation     1;
    openFraction    0.2;
    openingTime     5.0;
    maxOpenFractionDelta 0.1;
}
```

1.2 activePressureForceBaffleVelocity

This boundary condition is applied to the flow velocity, to simulate the opening of a baffle due to local flow conditions, by merging the behaviours of wall and cyclic conditions.

The baffle joins two mesh regions, where the open fraction determines the interpolation weights applied to each cyclic- and neighbour-patch contribution.

Once opened the baffle continues to open at a fixed rate using

$$x = x_{old} + \frac{dt}{DT} \tag{1.2}$$

x : baffle open fraction [0-1]
x_{old} : baffle open fraction on previous evaluation
dt : simulation time step
DT : time taken to open the baffle

Property	Description	Required	Default value
p	pressure field name	no	p
cyclicPatch	cylclic patch name	yes	
orientation	1 or -1 used to switch flow direction	yes	
openFraction	current opatch open fraction [0-1]	yes	
openingTime	time taken to open the baffle	yes	
maxOpenFractionDelta	max open fraction change per timestep	yes	
minThresholdValue	minimum open fraction for activation	yes	
forceBased	force (true) or pressure-based (false) activation	yes	

Example

```
myPatch
{
    type            activePressureForceBaffleVelocity;
    p               p;
    cyclicPatch     cyclic1;
    orientation     1;
    openFraction    0.2;
    openingTime     5.0;
    maxOpenFractionDelta 0.1;
    minThresholdValue 0.01;
    forceBased      false;
}
```

1.3 advective

This boundary condition provides an advective outflow condition, based on solving DDt(psi, U) = 0 at the boundary.

The standard (Euler, backward, CrankNicolson) time schemes are supported. Additionally an optional mechanism to relax the value at the boundary to a specified far-field value is provided which is switched on by specifying the relaxation length-scale ļInf and the far-field value ƒieldInf.

The flow/wave speed at the outlet is provided by the virtual function advectionSpeed() the default implementation of which requires the name of the flux field (ϕhi) and optionally the density (ρho) if the mass-flux rather than the volumetric-flux is given.

The flow/wave speed at the outlet can be changed by deriving a specialised BC from this class and over-riding advectionSpeed() e.g. in waveTransmissiveFvPatchField the advectionSpeed() calculates and returns the flow-speed plus the acoustic wave speed creating an acoustic wave transmissive boundary condition.

Property	Description	Required	Default value
phi	flux field name	no	phi
rho	density field name	no	rho
fieldInf	value of field beyond patch	no	
lInf	distance beyond patch for $fieldInf$	no	

Example

```
myPatch
{
    type            advective;
    phi             phi;
}
```

Note :
If $lInf$ is specified, $fieldInf$ will be required; rho is only required in the case of a mass-based flux.

1.4 codedFixedValue

Constructs on-the-fly a new boundary condition (derived from fixedValueFvPatchField) which is then used to evaluate.

Example

```
myPatch
{
    type            codedFixedValue;
    value           uniform 0;
    redirectType    rampedFixedValue;    // name of generated BC

    code
    #{
        operator==(min(10, 0.1*this->db().time().value()));
    #};

    //codeInclude
    //#{
    //    #include "fvCFD.H"
    //#};

    //codeOptions
    //#{
    //    -I$(LIB_SRC)/finiteVolume/lnInclude
    //#};
}
```

A special form is if the 'code' section is not supplied. In this case the code is read from a (runTimeModifiable!) dictionary system/codeDict which would have a corresponding entry:

Example

```
myPatch
{
    code
    #{
        operator==(min(10, 0.1*this->db().time().value()));
    #};
}
```

1.5 codedMixed

Constructs on-the-fly a new boundary condition (derived from mixedFvPatchField) which is then used to evaluate.

Example

```
myPatch
{
    type            codedMixed;

    refValue        uniform (0 0 0);
    refGradient     uniform (0 0 0);
    valueFraction   uniform 1;

    redirectType    rampedMixed;   // name of generated BC

    code
    #{
        this->refValue() =
            vector(1, 0, 0)
           *min(10, 0.1*this->db().time().value());
        this->refGrad() = vector::zero;
        this->valueFraction() = 1.0;
    #};

    //codeInclude
    //#{
    //    #include "fvCFD.H"
    //#};
    //codeOptions
    //#{
    //    -I$(LIB_SRC)/finiteVolume/lnInclude
    //#};
}
```

A special form is if the 'code' section is not supplied. In this case the code gets read from a (runTimeModifiable!) dictionary system/codeDict which would have a corresponding entry

Example

```
myPatch
{
    code
    #{
        this->refValue() = min(10, 0.1*this->db().time().value());
        this->refGrad() = vector::zero;
        this->valueFraction() = 1.0;
    #};
}
```

1.6 cylindricalInletVelocity

This boundary condition describes an inlet vector boundary condition in cylindrical co-ordinates given a central axis, central point, rpm, axial and radial velocity.

Property	Description	Required	Default value
axis	axis of rotation	yes	
centre	centre of rotation	yes	
axialVelocity	axial velocity profile [m/s]	yes	
radialVelocity	radial velocity profile [m/s]	yes	
rpm	rotational speed (revolutions per minute)	yes	

Example

```
myPatch
{
    type            cylindricalInletVelocity;
    axis            (0 0 1);
    centre          (0 0 0);
    axialVelocity   constant 30;
    radialVelocity  constant -10;
    rpm             constant 100;
}
```

Note :
The $axialVelocity$, $radialVelocity$ and rpm entries are DataEntry types, able to describe time varying functions. The example above gives the usage for supplying constant values.

1.7 cylindricalInletVelocity

This boundary condition describes an inlet vector boundary condition in cylindrical co-ordinates given a central axis, central point, rpm, axial and radial velocity.

Property	Description	Required	Default value
axis	axis of rotation	yes	
centre	centre of rotation	yes	
axialVelocity	axial velocity profile [m/s]	yes	
radialVelocity	radial velocity profile [m/s]	yes	
rpm	rotational speed (revolutions per minute)	yes	

Example

```
myPatch
{
    type            cylindricalInletVelocity;
    axis            (0 0 1);
    centre          (0 0 0);
    axialVelocity   constant 30;
    radialVelocity  constant -10;
    rpm             constant 100;
}
```

Note :
The $axialVelocity$, $radialVelocity$ and rpm entries are DataEntry types, able to describe time varying functions. The example above gives the usage for supplying constant values.

1.8 externalCoupledMixed

This boundary condition provides an interface to an external application. Values are transferred as plain text files, where OpenFOAM data is written as:

Patch: <patch name>
<magSf1> <value1> <surfaceNormalGradient1>
<magSf2> <value2> <surfaceNormalGradient2>
<magSf3> <value3> <surfaceNormalGradient3>
...
<magSfN> <valueN> <surfaceNormalGradientN>

and received as the constituent pieces of the 'mixed' condition, i.e.

Patch: <patch name>
<value1> <gradient1> <valueFracion1>
<value2> <gradient2> <valueFracion2>
<value3> <gradient3> <valueFracion3>
...
<valueN> <gradientN> <valueFracionN>

Data is sent/received as a single file for all patches from the directory

$FOAM_CASE/<commsDir>

At start-up, the boundary creates a lock file, i.e..

OpenFOAM.lock

... to signal the external source to wait. During the boundary condition update, boundary values are written to file, e.g.

<fileName>.out

The lock file is then removed, instructing the external source to take control of the program execution. When ready, the external program should create the return values, e.g. to file

<fileName>.in

... and then re-instate the lock file. The boundary condition will then read the return values, and pass program execution back to OpenFOAM.

Property	Description	Required	Default value
commsDir	communications directory	yes	
fileName	transfer file name	yes	
waitInterval	interval [s] between file checks	no	1
timeOut	time after which error invoked [s]	no	100*waitInterval
calcFrequency	calculation frequency	no	1
initByExternal	external app to initialises values	yes	
log	log program control	no	

Example

```
myPatch
{
    type            externalCoupled;
    commsDir        "$FOAM_CASE/comms";
    fileName        data;
    calcFrequency   1;
    initByExternal  yes;
}
```

1.9 fan

This boundary condition provides a jump condition, using the çyclic condition as a base.

The jump is specified as a ḊataEntry type, to enable the use of, e.g. contant, polynomial, table values.

Property	Description	Required	Default value
patchType	underlying patch type should be *cyclic*	yes	
jumpTable	jump data, e.g. *csvFile*	yes	

Example

```
myPatch
{
    type            fan;
    patchType       cyclic;
    jumpTable       csvFile;
    csvFileCoeffs
    {
        hasHeaderLine   1;
        refColumn       0;
        componentColumns 1(1);
        separator       ",";
        fileName        "$FOAM_CASE/constant/pressureVsU";
    }
    value           uniform 0;
}
```

The above example shows the use of a comma separated (CSV) file to specify the jump condition.

Note :
The underlying *patchType* should be set to *cyclic*

1.10 fanPressure

This boundary condition can be applied to assign either a pressure inlet or outlet total pressure condition for a fan.

Property	Description	Required	Default value
fileName	fan curve file name	yes	
outOfBounds	out of bounds handling	yes	
direction	direction of flow through fan [in/out]	yes	
p0	environmental total pressure	yes	

Example

```
inlet
{
    type            fanPressure;
    fileName        "fanCurve";
    outOfBounds     clamp;
    direction       in;
    p0              uniform 0;
    value           uniform 0;
}

outlet
{
    type            fanPressure;
    fileName        "fanCurve";
    outOfBounds     clamp;
    direction       out;
    p0              uniform 0;
    value           uniform 0;
}
```

Note :
If reverse flow is possible or expected use the pressureInletOutletVelocity condition instead.

1.11 fixedFluxPressure

This boundary condition adjusts the pressure gradient such that the flux on the boundary is that specified by the velocity boundary condition.

The predicted flux to be compensated by the pressure gradient is evaluated as $(\phi - \phi_{H/A})$, both of which are looked-up from the database, as is the pressure diffusivity used to calculate the gradient using:

$$\nabla(p) = \frac{\phi_{H/A} - \phi}{|Sf|D_p} \tag{1.3}$$

ϕ : flux
D_p : pressure diffusivity
Sf : patch face areas [m2]

Property	Description	Required	Default value
phiHbyA	name of predicted flux field	no	phiHbyA
phi	name of flux field	no	phi
rho	name of density field	no	rho
Dp	name of pressure diffusivity field	no	Dp

Example

```
myPatch
{
    type            fixedFluxPressure;
    phiHbyA         phiHbyA;
    phi             phi;
    rho             rho;
    Dp              Dp;
}
```

1.12 fixedInternalValue

This boundary condition provides a mechanism to set boundary (cell) values directly into a matrix, i.e. to set a constraint condition. Default behaviour is to act as a zero gradient condition.

Example

```
myPatch
{
    type            fixedInternalValue;
    value           uniform 0;              // place holder
}
```

Note :
This is used as a base for conditions such as the turbulence *epsilon* wall function, which applies a near-wall constraint for high Reynolds number flows.

1.13 fixedJump

This boundary condition provides a jump condition, using the *cyclic* condition as a base. The jump is specified as a fixed value field, applied as an offset to the 'owner' patch.

Property	Description	Required	Default value
patchType	underlying patch type should be *cyclic*	yes	
jump	current jump value	yes	

Example

```
myPatch
{
    type            fixedJump;
    patchType       cyclic;
    jump            uniform 10;
}
```

The above example shows the use of a fixed jump of '10'.

Note :
The underlying *patchType* should be set to *cyclic*

1.14 fixedJumpAMI

This boundary condition provides a jump condition, across non-conformal cyclic path-pairs, employing an arbitraryMeshInterface (AMI).
The jump is specified as a fixed value field, applied as an offset to the 'owner' patch.

Property	Description	Required	Default value
patchType	underlying patch type should be *cyclic*	yes	
jump	current jump value	yes	

Example

```
myPatch
{
    type            fixedJumpAMI;
    patchType       cyclic;
    jump            uniform 10;
}
```

The above example shows the use of a fixed jump of '10'.

Note :
The underlying *patchType* should be set to *cyclicAMI*

1.15 fixedMean

This boundary condition extrapolates field to the patch using the near-cell values and adjusts the distribution to match the specified mean value.

Property	Description	Required	Default value
meanValue	mean value	yes	

Example

```
myPatch
{
    type            fixedMean;
    meanValue       1.0;
}
```

1.16 fixedNormalSlip

This boundary condition sets the patch-normal component to a fixed value.

Property	Description	Required	Default value
fixedValue	fixed value	yes	

Example

```
myPatch
{
    type            fixedNormalSlip;
    fixedValue      uniform 0;      // example entry for a scalar field
}
```

1.17 fixedPressureCompressibleDensity

This boundary condition calculates a (liquid) compressible density as a function of pressure and fluid properties:

$$\rho = \rho_{l,sat} + \psi_l * (p - p_{sat}) \tag{1.4}$$

ρ : density [kg/m3]
$\rho_{l,sat}$: saturation liquid density [kg/m3]
ψ_l : liquid compressibility
p : pressure [Pa]
p_{sat} : saturation pressure [Pa]

The variables $\rho_{l,sat}$, p_{sat} and ψ_l are retrieved from the *thermodynamicProperties* dictionary.

Property	Description	Required	Default value
p	pressure field name	no	p

Example

```
myPatch
{
    type        fixedPressureCompressibleDensity;
    p           p;
    value       uniform 1;
}
```

1.18 flowRateInletVelocity

This boundary condition provides a velocity boundary condition, derived from the flux (volumetric or mass-based), whose direction is assumed to be normal to the patch.

For a mass-based flux:
- the flow rate should be provided in kg/s
- if *rhoName* is "none" the flow rate is in m3/s
- otherwise *rhoName* should correspond to the name of the density field
- if the density field cannot be found in the database, the user must specify the inlet density using the *rhoInlet* entry

For a volumetric-based flux:
- the flow rate is in m3/s

Property	Description	Required	Default value
massFlowRate	mass flow rate [kg/s]	no	
volumetricFlowRate	volumetric flow rate [m3/s]	no	
rhoInlet	inlet density	no	

Example for a volumetric flow rate

```
myPatch
{
    type        flowRateInletVelocity;
    volumetricFlowRate  0.2;
    value       uniform (0 0 0); // placeholder
}
```

Example for a mass flow rate

```
myPatch
{
    type                flowRateInletVelocity;
    massFlowRate        0.2;
    rho                 rho;
    rhoInlet            1.0;
}
```

The *flowRate* entry is a *DataEntry* type, meaning that it can be specified as constant, a polynomial fuction of time, and ...

Note :
- *rhoInlet* is required for the case of a mass flow rate, where the density field is not available at start-up
- the value is positive into the domain (as an inlet)
- may not work correctly for transonic inlets
- strange behaviour with potentialFoam since the U equation is not solved

1.19 fluxCorrectedVelocity

This boundary condition provides a velocity outlet boundary condition for patches where the pressure is specified. The outflow velocity is obtained by "zeroGradient" and then corrected from the flux:

$$U_p = U_c - n(n \cdot U_c) + \frac{n\phi_p}{|S_f|} \tag{1.5}$$

U_p : velocity at the patch [m/s]
U_c : velocity in cells adjacent to the patch [m/s]
n : patch normal vectors ϕ_p : flux at the patch [m3/s or kg/s]
S_f : patch face area vectors [m2]

Property	Description	Required	Default value
phi	name of flux field	no	phi
rho	name of density field	no	rho

Example

```
myPatch
{
    type            fluxCorrectedVelocity;
    phi             phi;
    rho             rho;
}
```

Note :
If reverse flow is possible or expected use the pressureInletOutletVelocity condition instead.

1.20 freestream

This boundary condition provides a free-stream condition. It is a 'mixed' condition derived from the *inletOutlet* condition, whereby the mode of operation switches between fixed (free stream) value and zero gradient based on the sign of the flux.

Property	Description	Required	Default value
freestreamValue	freestream velocity	yes	
phi	flux field name	no	phi

Example

```
myPatch
{
    type            freestream;
    phi             phi;
}
```

1.21 freestreamPressure

This boundary condition provides a free-stream condition for pressure. It is a zero-gradient condition that constrains the flux across the patch based on the free-stream velocity.

Example

```
myPatch
{
    type            freestreamPressure;
}
```

Note :
This condition is designed to operate with a freestream velocity condition

1.22 inletOutlet

This boundary condition provides a generic outflow condition, with specified inflow for the case of return flow.

Property	Description	Required	Default value
phi	flux field name	no	phi
inletValue	inlet value for reverse flow	yes	

Example

```
myPatch
{
    type            inletOutlet;
    phi             phi;
    inletValue      uniform 0;
    value           uniform 0;
}
```

The mode of operation is determined by the sign of the flux across the patch faces.

Note :
Sign conventions:
- positive flux (out of domain): apply zero-gradient condition
- negative flux (into of domain): apply the user-specified fixed value

1.23 inletOutletTotalTemperature

This boundary condition provides an outflow condition for total temperature for use with supersonic cases, where a user-specified value is applied in the case of reverse flow.

Property	Description	Required	Default value
U	velocity field name	no	U
phi	flux field name	no	phi
psi	compressibility field name	no	psi
gamma	heat capacity ration (Cp/Cv)	yes	
inletValue	reverse flow (inlet) value	yes	
T0	static temperature [K]	yes	

Example

```
myPatch
{
    type            inletOutletTotalTemperature;
    U               U;
    phi             phi;
    psi             psi;
    gamma           gamma;
    inletValue      uniform 0;
    T0              uniform 0;
    value           uniform 0;
}
```

1.24 interstitialInletVelocity

Inlet velocity in which the actual interstitial velocity is calculated by dividing the specified inletVelocity field with the local phase-fraction.

Example

```
inlet
{
    type              interstitialInletVelocity;
    inletVelocity     uniform (0 0.2 0; // Non-interstitial inlet velocity
    alpha             alpha.particles;  // Name of the phase-fraction field
    value             uniform (0 0 0);
}
```

1.25 mappedField

This boundary condition provides a self-contained version of the *mapped* condition. It does not use information on the patch; instead it holds
thr data locally.

Property	Description	Required	Default value
fieldName	name of field to be mapped	no	this field name
setAverage	flag to activate setting of average value	yes	
average	average value to apply if $setAverage$ = yes	yes	

Example

```
myPatch
{
    type            mappedField;
    fieldName       T;              // optional field name
    setAverage      no;             // apply an average value
    average         0;              // average to apply if setAverage
    value           uniform 0;      // place holder
}
```

Note :
Since this condition can be applied on a per-field and per-patch basis, it is possible to duplicate the mapping information. If possible, employ the *mapped* condition in preference to avoid this situation, and only employ this condition if it is not possible to change the underlying geometric (poly) patch type to *mapped*.

1.26 mappedFixedInternalValue

This boundary condition maps the boundary and internal values of a neighbour patch field to the boundary and internal values of *this.

Property	Description	Required	Default value
fieldName	name of field to be mapped	no	this field name
setAverage	flag to activate setting of average value	yes	
average	average value to apply if $setAverage$ = yes	yes	

Example

```
myPatch
{
    type            mappedFixedInternalValue;
    fieldName       T;
    setAverage      no;
    average         0;
    value           uniform 0;
}
```

Note :
This boundary condition can only be applied to patches that are of the *mappedPolyPatch* type.

1.27 mappedFixedPushedInternalValue

This boundary condition maps the boundary values of a neighbour patch field to the boundary and internal cell values of *this.

Property	Description	Required	Default value
fieldName	name of field to be mapped	no	this field name
setAverage	flag to activate setting of average value	yes	
average	average value to apply if $setAverage$ = yes	yes	

Example

```
myPatch
{
    type            mappedFixedPushedInternalValue;
    fieldName       T;
    setAverage      no;
    average         0;
    value           uniform 0;
}
```

Note :
This boundary condition can only be applied to patches that are of the *mappedPolyPatch* type.

1.28 mappedFixedValue

This boundary condition maps the value at a set of cells or patch faces back to *this.

The sample mode is set by the underlying mapping engine, provided by the mappedPatchBase class.

Property	Description	Required	Default value
fieldName	name of field to be mapped	no	this field name
setAverage	flag to activate setting of average value	yes	
average	average value to apply if *setAverage* = yes	yes	
interpolationScheme	type of interpolation scheme	no	

Example

```
myPatch
{
    type            mapped;
    fieldName       T;
    setAverage      no;
    average         0;
    interpolationScheme cell;
    value           uniform 0;
}
```

When employing the *nearestCell* sample mode, the user must also specify the interpolation scheme using the *interpolationScheme* entry.

In case of interpolation (where scheme != cell) the limitation is that there is only one value per cell. For example, if you have a cell with two boundary faces and both faces sample into the cell, both faces will get the same value.

Note :
It is not possible to sample internal faces since volume fields are not defined on faces.

1.29 mappedFlowRate

Describes a volumetric/mass flow normal vector boundary condition by its magnitude as an integral over its area.

The inlet mass flux is taken from the neighbour region.

The basis of the patch (volumetric or mass) is determined by the dimensions of the flux, phi. The current density is used to correct the velocity when applying the mass basis.

Property	Description	Required	Default value
phi	flux field name	no	phi
rho	density field name	no	rho
neigPhi	name of flux field on neighbour mesh	yes	

Example

```
myPatch
{
    type            mappedFlowRate;
    phi             phi;
    rho             rho;
    neigPhi         phi;
    value           uniform (0 0 0); // placeholder
}
```

1.30 mappedVelocityFluxFixedValue

This boundary condition maps the velocity and flux from a neighbour patch to this patch

Property	Description	Required	Default value
phi	flux field name	no	phi

Example

```
myPatch
{
    type            mappedVelocityFlux;
    phi             phi;
    value           uniform 0;  // place holder
}
```

The underlying sample mode should be set to *nearestPatchFace* or *nearestFace*

Note :
This boundary condition can only be applied to patches that are of the *mappedPolyPatch* type.

1.31 movingWallVelocity

This boundary condition provides a velocity condition for cases with moving walls. In addition, it should also be applied to 'moving' walls for moving reference frame (MRF) calculations.

Property	Description	Required	Default value
U	velociy field name	no	U

Example

```
myPatch
{
    type            movingWallVelocity;
    U               U;
    value           uniform 0; // initial value
}
```

1.32 oscillatingFixedValue

This boundary condition provides an oscillating condition in terms of amplitude and frequency.

$$x_p = (1 + asin(\pi ft))x_{ref} + x_o \tag{1.6}$$

x_p : patch values
x_{ref} : patch reference values
x_o : patch offset values
a : amplitude
f : frequency [1/s]
t : time [s]

Property	Description	Required	Default value
refValue	reference value	yes	
offset	offset value	no	0.0
amplitude	oscillation amplitude	yes	
frequency	oscillation frequency	yes	

Example

```
myPatch
{
    type            oscillatingFixedValue;
    refValue        uniform 5.0;
    offset          0.0;
    amplitude       constant 0.5;
    frequency       constant 10;
}
```

Note :
The amplitude and frequency entries are DataEntry types, able to describe time varying functions. The example above gives the usage for supplying constant values.

1.33 outletInlet

This boundary condition provides a generic inflow condition, with specified outflow for the case of return flow.

Property	Description	Required	Default value
phi	flux field name	no	phi
inletValue	inlet value	yes	

Example

```
myPatch
{
    type            outletInlet;
    phi             phi;        // name of flux field (default = phi)
    outletValue     uniform 0;  // reverse flow (inlet) value
    value           uniform 0;  // initial value
}
```

The mode of operation is determined by the sign of the flux across the patch faces.

Note :
Sign conventions:
- positive flux (out of domain): apply the user-specified fixed value
- negative flux (into of domain): apply zero-gradient condition

1.34 outletMappedUniformInlet

This boundary conditon averages the field over the "outlet" patch specified by name "outletPatchName" and applies this as the uniform value of the field over this patch.

Property	Description	Required	Default value
outletPatchName	name of outlet patch	yes	
phi	flux field name	no	phi

Example

```
myPatch
{
    type            outletMappedUniformInlet;
    outletPatchName aPatch;
    phi             phi;
    value           uniform 0;
}
```

1.35 outletPhaseMeanVelocity

This boundary condition adjusts the velocity for the given phase to achieve the specified mean thus causing the phase-fraction to adjust according to the mass flow rate.

Typical usage is as the outlet condition for a towing-tank ship simulation to maintain the outlet water level at the level as the inlet.

Property	Description	Required	Default value
Umean	mean velocity normal to the boundary [m/s]	yes	
alpha	phase-fraction field	yes	

Example

```
myPatch
{
    type            outletPhaseMeanVelocity;
    Umean           1.2;
    alpha           alpha.water;
    value           uniform (1.2 0 0);
}
```

1.36 partialSlip

This boundary condition provides a partial slip condition. The amount of slip is controlled by a user-supplied field.

Property	Description	Required	Default value
valueFraction	fraction od value used for boundary [0-1]	yes	

Example

```
myPatch
{
    type            partialSlip;
    valueFraction   uniform 0.1;
    value           uniform 0;
}
```

1.37 phaseHydrostaticPressure

This boundary condition provides a phase-based hydrostatic pressure condition, calculated as:

$$p_{hyd} = p_{ref} + \rho g(x - x_{ref}) \tag{1.7}$$

p_{hyd} : hyrostatic pressure [Pa]
p_{ref} : reference pressure [Pa]
x_{ref} : reference point in Cartesian co-ordinates
ρ : density (assumed uniform)
g : acceleration due to gravity [m/s2]

The values are assigned according to the phase-fraction field:
- 1: apply p_{hyd}
- 0: apply a zero-gradient condition

Property	Description	Required	Default value
phaseName	phase field name	no	alpha
rho	density field name	no	rho
pRefValue	reference pressure [Pa]	yes	
pRefPoint	reference pressure location	yes	

Example

```
myPatch
{
    type            phaseHydrostaticPressure;
    phaseName       alpha1;
    rho             rho;
    pRefValue       1e5;
    pRefPoint       (0 0 0);
    value           uniform 0; // optional initial value
}
```

1.38 pressureDirectedInletOutletVelocity

This velocity inlet/outlet boundary condition is applied to pressure boundaries where the pressure is specified. A zero-gradient condtion is applied for outflow (as defined by the flux); for inflow, the velocity is obtained from the flux with the specified inlet direction.

Property	Description	Required	Default value
phi	flux field name	no	phi
rho	density field name	no	rho
inletDirection	inlet direction per patch face	yes	

Example

```
myPatch
{
    type            pressureDirectedInletOutletVelocity;
    phi             phi;
    rho             rho;
    inletDirection  uniform (1 0 0);
    value           uniform 0;
}
```

Note :
Sign conventions:
- positive flux (out of domain): apply zero-gradient condition
- negative flux (into of domain): derive from the flux with specified direction

1.39 pressureDirectedInletVelocity

This velocity inlet boundary condition is applied to patches where the pressure is specified. The inflow velocity is obtained from the flux with the specified inlet direction" direction.

Property	Description	Required	Default value
phi	flux field name	no	phi
rho	density field name	no	rho
inletDirection	inlet direction per patch face	yes	

Example

```
myPatch
{
    type            pressureDirectedInletVelocity;
    phi             phi;
    rho             rho;
    inletDirection  uniform (1 0 0);
    value           uniform 0;
}
```

Note :
If reverse flow is possible or expected use the pressureDirectedInletOutletVelocityFvPatchVectorField condition instead.

1.40 pressureInletOutletParSlipVelocity

This velocity inlet/outlet boundary condition for pressure boundary where the pressure is specified. A zero-gradient is applied for outflow (as defined by the flux); for inflow, the velocity is obtained from the flux with the specified inlet direction.

A slip condition is applied tangential to the patch.

Property	Description	Required	Default value
phi	flux field name	no	phi
rho	density field name	no	rho
U	velocity field name	no	U

Example

```
myPatch
{
    type            pressureInletOutletParSlipVelocity;
    value           uniform 0;
}
```

Note :
Sign conventions:
- positive flux (out of domain): apply zero-gradient condition
- negative flux (into of domain): derive from the flux with specified direction

1.41 pressureInletOutletVelocity

This velocity inlet/outlet boundary condition is applied to pressure boundaries where the pressure is specified. A zero-gradient condition is applied for outflow (as defined by the flux); for inflow, the velocity is obtained from the patch-face normal component of the internal-cell value.

The tangential patch velocity can be optionally specified.

Property	Description	Required	Default value
phi	flux field name	no	phi
tangentialVelocity	tangential velocity field	no	

Example

```
myPatch
{
    type            pressureInletOutletVelocity;
    phi             phi;
    tangentialVelocity uniform (0 0 0);
    value           uniform 0;
}
```

Note :
Sign conventions:
- positive flux (out of domain): apply zero-gradient condition
- negative flux (into of domain): derive from the flux in the patch-normal direction

1.42 pressureInletUniformVelocity

This velocity inlet boundary condition is applied to patches where the pressure is specified. The uniform inflow velocity is obtained by averaging the flux over the patch, and then applying it in the direction normal to the patch faces.

Example

```
myPatch
{
    type            pressureInletUniformVelocity;
    value           uniform 0;
}
```

1.43 pressureInletVelocity

This velocity inlet boundary condition is applied to patches where the pressure is specified. The inflow velocity is obtained from the flux with a direction normal to the patch faces.

Example

```
myPatch
{
    type            pressureInletVelocity;
    phi             phi;
    rho             rho;
    value           uniform 0;
}
```

Note:
If reverse flow is possible or expected use the pressureInletOutletVelocityFvPatchVectorField condition instead.

1.44 pressureNormalInletOutletVelocity

This velocity inlet/outlet boundary condition is applied to patches where the pressure is specified. A zero-gradient condition is applied for outflow (as defined by the flux); for inflow, the velocity is obtained from the flux with a direction normal to the patch faces.

Property	Description	Required	Default value
phi	flux field name	no	phi
rho	density field name	no	rho

Example

```
myPatch
{
    type            pressureNormalInletOutletVelocity;
    phi             phi;
    rho             rho;
    value           uniform 0;
}
```

Note :
Sign conventions:
- positive flux (out of domain): apply zero-gradient condition
- negative flux (into of domain): derive from the flux and patch-normal direction

1.45 rotatingPressureInletOutletVelocity

This velocity inlet/outlet boundary condition is applied to patches in a rotating frame where the pressure is specified. A zero-gradient is applied for outflow (as defined by the flux); for inflow, the velocity is obtained from the flux with a direction normal to the patch faces.

Property	Description	Required	Default value
phi	flux field name	no	phi
tangentialVelocity	tangential velocity field	no	
omega	angular velocty of the frame [rad/s]	yes	

Example

```
myPatch
{
    type            rotatingPressureInletOutletVelocity;
    phi             phi;
    tangentialVelocity uniform (0 0 0);
    omega           100;
}
```

The *omega* entry is a DataEntry type, able to describe time varying functions.

Note :
Sign conventions:
- positive flux (out of domain): apply zero-gradient condition
- negative flux (into of domain): derive from the flux and patch-normal direction

1.46 rotatingTotalPressure

This boundary condition provides a total pressure condition for patches in a rotating frame.

Property	Description	Required	Default value
U	velocity field name	no	U
phi	flux field name	no	phi
rho	density field name	no	none
psi	compressibility field name	no	none
gamma	ratio of specific heats (Cp/Cv)	yes	
p0	static pressure reference	yes	
omega	angular velocty of the frame [rad/s]	yes	

Example

```
myPatch
{
    type            rotatingTotalPressure;
    U               U;
    phi             phi;
    rho             rho;
    psi             psi;
    gamma           1.4;
    p0              uniform 1e5;
    omega           100;
}
```

The *omega* entry is a DataEntry type, able to describe time varying functions.

1.47 rotatingWallVelocity

This boundary condition provides a rotational velocity condition.

Property	Description	Required	Default value
origin	origin of rotation in Cartesian co-ordinates	yes	
axis	axis of rotation	yes	
omega	angular velocty of the frame [rad/s]	yes	

Example

```
myPatch
{
    type            rotatingWallVelocity;
    origin          (0 0 0);
    axis            (0 0 1);
    omega           100;
}
```

The *omega* entry is a DataEntry type, able to describe time varying functions.

1.48 slip

This boundary condition provides a slip constraint.

Example

```
myPatch
{
    type            slip;
}
```

1.49 supersonicFreestream

This boundary condition provides a supersonic free-stream condition.

- supersonic outflow is vented according to ???
- supersonic inflow is assumed to occur according to the Prandtl-Meyer expansion process.
- subsonic outflow is applied via a zero-gradient condition from inside the domain.

Property	Description	Required	Default value
TName	Temperature field name	no	T
pName	Pressure field name	no	p
psiName	Compressibility field name	no	thermo:psi
UInf	free-stream velocity	yes	
pInf	free-stream pressure	yes	
TInf	free-stream temperature	yes	
gamma	heat capacity ratio (cp/Cv)	yes	

Example

```
myPatch
{
    type            supersonicFreestream;
    UInf            500;
    pInf            1e4;
    TInf            265;
    gamma           1.4;
}
```

Note:
This boundary condition is ill-posed if the free-stream flow is normal to the boundary.

1.50 surfaceNormalFixedValue

This boundary condition provides a surface-normal vector boundary condition by its magnitude.

Property	Description	Required	Default value
refValue	reference value	yes	

Example

```
myPatch
{
    type            surfaceNormalFixedValue;
    refValue        -10;           // 10 INTO the domain
}
```

Note:
Sign conventions:
- the value is positive for outward-pointing vectors

1.51 swirlFlowRateInletVelocity

This boundary condition provides a volumetric- OR mass-flow normal vector boundary condition by its magnitude as an integral over its area with a swirl component determined by the angular speed, given in revolutions per minute (RPM)

The basis of the patch (volumetric or mass) is determined by the dimensions of the flux, phi. The current density is used to correct the velocity when applying the mass basis.

Property	Description	Required	Default value
phi	flux field name	no	phi
rho	density field name	no	rho
flowRate	flow rate profile	yes	
rpm	rotational speed profile	yes	

Example

```
myPatch
{
    type            swirlFlowRateInletVelocity;
    flowRate        constant 0.2;
    rpm             constant 100;
}
```

Note:
- the *flowRate* and *rpm* entries are DataEntry types, able to describe time varying functions. The example above gives the usage for supplying constant values.
- the value is positive into the domain

1.52 syringePressure

This boundary condition provides a pressure condition, obtained from a zero-D model of the cylinder of a syringe.

The syringe cylinder is defined by its initial volume, piston area and velocity profile specified by regions of constant acceleration, speed and deceleration. The gas in the cylinder is described by its initial pressure and compressibility which is assumed constant, i.e. isothermal expansion/-compression.

Property	Description	Required	Default value
Ap	syringe piston area [m2]	yes	
Sp	syringe piston speed [m/s]	yes	
VsI	initial syringe volume [m3]	yes	
tas	start of piston acceleration [s]	yes	
tae	end of piston acceleration [s]	yes	
tds	start of piston deceleration [s]	yes	
tde	end of piston deceleration [s]	yes	
psI	initial syringe pressure [Pa]	yes	
psi	gas compressibility [m2/s2]	yes	
ams	added (or removed) gas mass [kg]	yes	

Example

```
myPatch
{
    type            syringePressure;
    Ap              1.388e-6;
    Sp              0.01;
    VsI             1.388e-8;
    tas             0.001;
    tae             0.002;
    tds             0.005;
    tde             0.006;
    psI             1e5;
    psi             1e-5;
    ams             0;
    value           uniform 0;
}
```

1.53 timeVaryingMappedFixedValue

This boundary conditions interpolates the values from a set of supplied points in space and time. Supplied data should be specified in constant/boundaryData/< *patchname* > where:
- points : pointField with locations
- ddd : supplied values at time ddd
The points should be more or less on a plane since they get triangulated in 2-D.

At startup, this condition generates the triangulation and performs a linear interpolation (triangle it is in and weights to the 3 vertices) for every face centre.

Values are interpolated linearly between times.

Property	Description	Required	Default value
setAverage	flag to activate setting of average value	yes	
perturb	perturb points for regular geometries	no	1e-5
fieldTableName	alternative field name to sample	no	this field name

Example

```
myPatch
{
    type            timeVaryingMappedFixedValue;
    setAverage      false;
    //perturb       0.0;
    //fieldTableName samples;
}
```

Note:
Switch on debug flag to have it dump the triangulation (in transformed space) and transform face centres.

1.54 totalPressure

This boundary condition provides a total pressure condition. Four variants are possible:

1. incompressible subsonic:

$$p_T = p + 0.5|U|^2 \tag{1.8}$$

p_T : incompressible total pressure [m2/s2]
p : incompressible reference pressure [m2/s2]
U : velocity

2. compressible subsonic:

$$p_T = p + 0.5\rho|U|^2 \tag{1.9}$$

p_T : total pressure [Pa]
p : reference pressure [Pa]
ρ : density [kg/m3]
U : velocity

3. compressible transonic ($\gamma <= 1$):

$$p = \frac{p_T}{1 + 0.5\psi|U|^2} \tag{1.10}$$

$$-> p_T = p + 0.5\rho U^2 \tag{1.11}$$

p_T : total pressure [Pa]
p : reference pressure [Pa]
ψ : 1/RT [s2/m2]

4. compressible supersonic ($\gamma > 1$):

$$p = \frac{p_T}{(1 + 0.5\psi G|U|^2)^{\frac{1}{G}}} \tag{1.12}$$

$$-> p_T = p\left(1 + \frac{\gamma - 1}{2}M^2\right)^{\frac{\gamma}{\gamma-1}} \tag{1.13}$$

γ : ratio of specific heats (Cp/Cv)
p_T : total pressure [Pa]

p : reference pressure [Pa]
ψ : 1/RT [s2/m2]
G : coefficient given by $\frac{\gamma-1}{\gamma}$

The modes of operation are set via the combination of *phi*, *rho*, and *psi* entries:

Mode	**phi**	**rho**	**psi**
incompressible subsonic	phi	none	none
compressible subsonic	phi	rho	none
compressible transonic	phi	none	psi
compressible supersonic	phi	none	psi

Property	**Description**	**Required**	**Default value**
U	velocity field name	no	U
phi	flux field name	no	phi
rho	density field name	no	none
psi	compressibility field name	no	none
gamma	ratio of specific heats (Cp/Cv)	yes	
p0	static pressure reference	yes	

Example

```
myPatch
{
    type            totalPressure;
    U               U;
    phi             phi;
    rho             none;
    psi             none;
    gamma           1.4;
    p0              uniform 1e5;
}
```

Note:
The default boundary behaviour is for subsonic, incompressible flow.

1.55 totalTemperature

This boundary condition provides a total temperature condition.

Property	Description	Required	Default value
U	Velocity field name	no	U
phi	Flux field name	no	phi
psi	Compressibility field name	no	thermo:psi
gamma	ratio of specific heats (Cp/Cv)	yes	
T0	reference temperature	yes	

Example

```
myPatch
{
    type            totalTemperature;
    T0              uniform 300;
}
```

1.56 translatingWallVelocity

This boundary condition provides a velocity condition for translational motion on walls.

Property	Description	Required	Default value
U	translational velocity	yes	

Example

```
myPatch
{
    type            translatingWallVelocity;
    U               (100 0 0);
}
```

1.57 turbulentInlet

This boundary condition generates a fluctuating inlet condition by adding a random component to a reference (mean) field.

$$x_p = (1 - \alpha)x_p^{n-1} + \alpha(x_{ref} + sC_{RMS}x_{ref}) \tag{1.14}$$

x_p : patch values
x_{ref} : reference patch values
n : time level
α : fraction of new random component added to previous time value
C_{RMS} : RMS coefficient
s : fluctuation scale

Property	Description	Required	Default value
fluctuationScale	RMS fluctuation scale (fraction of mean)	yes	
referenceField	reference (mean) field	yes	
alpha	fraction of new random component added to previous	no	0.1

Example

```
myPatch
{
    type            turbulentInlet;
    fluctuationScale 0.1;
    referenceField  uniform 10;
    alpha           0.1;
}
```

1.58 turbulentIntensityKineticEnergyInlet

This boundary condition provides a turbulent kinetic energy condition, based on user-supplied turbulence intensity, defined as a fraction of the mean velocity:

$$k_p = 1.5I|U|^2 \tag{1.15}$$

k_p : kinetic energy at the patch
I : turbulence intensity
U : velocity field

In the event of reverse flow, a zero-gradient condition is applied.

Property	Description	Required	Default value
intensity	fraction of mean field [0-1]	yes	
U	velocity field name	no	U
phi	flux field name	no	phi

Example

```
myPatch
{
    type        turbulentIntensityKineticEnergyInlet;
    intensity   0.05;           // 5% turbulence
    value       uniform 1;      // placeholder
}
```

1.59 uniformDensityHydrostaticPressure

This boundary condition provides a hydrostatic pressure condition, calculated as:

$$p_{hyd} = p_{ref} + \rho g(x - x_{ref}) \tag{1.16}$$

p_{hyd} : hyrostatic pressure [Pa]
p_{ref} : reference pressure [Pa]
x_{ref} : reference point in Cartesian co-ordinates
ρ : density (assumed uniform)
g : acceleration due to gravity [m/s2]

Property	Description	Required	Default value
rho	uniform density [kg/m3]	yes	
pRefValue	reference pressure [Pa]	yes	
pRefPoint	reference pressure location	yes	

Example

```
myPatch
{
    type            uniformDensityHydrostaticPressure;
    rho             rho;
    pRefValue       1e5;
    pRefPoint       (0 0 0);
    value           uniform 0; // optional initial value
}
```

1.60 uniformFixedGradient

This boundary condition provides a uniform fixed gradient condition.

Property	Description	Required	Default value
uniformGradient	uniform gradient	yes	

Example

```
myPatch
{
    type            uniformFixedGradient;
    uniformGradient constant 0.2;
}
```

Note:
The uniformGradient entry is a DataEntry type, able to describe time varying functions. The example above gives the usage for supplying a constant value.

1.61 uniformFixedValue

This boundary condition provides a uniform fixed value condition.

Property	Description	Required	Default value
uniformValue	uniform value	yes	

Example

```
myPatch
{
    type            uniformFixedValue;
    uniformValue    constant 0.2;
}
```

Note:
The uniformValue entry is a DataEntry type, able to describe time varying functions. The example above gives the usage for supplying a constant value.

1.62 uniformJump

This boundary condition provides a jump condition, using the çyclic condition as a base. The jump is specified as a time-varying uniform value across the patch.

Property	Description	Required	Default value
patchType	underlying patch type should be *cyclic*	yes	
jumpTable	jump value	yes	

Example

```
myPatch
{
    type            uniformJump;
    patchType       cyclic;
    jumpTable       constant 10;
}
```

The above example shows the use of a fixed jump of '10'.

Note:
The uniformValue entry is a DataEntry type, able to describe time varying functions. The example above gives the usage for supplying a constant value.

1.63 uniformJumpAMI

This boundary condition provides a jump condition, using the çyclicAMI condition as a base. The jump is specified as a time-varying uniform value across the patch.

Property	Description	Required	Default value
patchType	underlying patch type should be *cyclicAMI*	yes	
jumpTable	jump value	yes	

Example

```
myPatch
{
    type            uniformJumpAMI;
    patchType       cyclicAMI;
    jumpTable       constant 10;
}
```

The above example shows the use of a fixed jump of '10'.

Note:
The uniformValue entry is a DataEntry type, able to describe time varying functions. The example above gives the usage for supplying a constant value.
The underlying *patchType* should be set to *cyclic*.

1.64 uniformTotalPressure

This boundary condition provides a time-varying form of the uniform total pressure boundary condition.

Property	Description	Required	Default value
U	velocity field name	no	U
phi	flux field name	no	phi
rho	density field name	no	none
psi	compressibility field name	no	none
gamma	ratio of specific heats (Cp/Cv)	yes	
pressure	total pressure as a function of time	yes	

Example

```
myPatch
{
    type            uniformTotalPressure;
    U               U;
    phi             phi;
    rho             rho;
    psi             psi;
    gamma           1.4;
    pressure        uniform 1e5;
}
```

The *pressure* entry is specified as a DataEntry type, able to describe time varying functions.

Note:
The default boundary behaviour is for subsonic, incompressible flow.

1.65 variableHeightFlowRate

This boundary condition provides a phase fraction condition based on the local flow conditions, whereby the values are constrained to lay between user-specified upper and lower bounds. The behaviour is described by:

if alpha > upperBound:
- apply a fixed value condition, with a uniform level of the upper bound

if lower bound <= alpha <= upper bound:
- apply a zero-gradient condition

if alpha < lowerBound:
- apply a fixed value condition, with a uniform level of the lower bound

Property	Description	Required	Default value
phi	flux field name	no	phi
lowerBound	lower bound for clipping	yes	
upperBound	upper bound for clipping	yes	

Example

```
myPatch
{
    type            variableHeightFlowRate;
    lowerBound      0.0;
    upperBound      0.9;
    value           uniform 0;
}
```

1.66 variableHeightFlowRateInletVelocity

This boundary condition provides a velocity boundary condition for multphase flow based on a user-specified volumetric flow rate.

The flow rate is made proportional to the phase fraction alpha at each face of the patch and alpha is ensured to be bound between 0 and 1.

Property	Description	Required	Default value
flowRate	volumetric flow rate [m3/s]	yes	

Example

```
myPatch
{
    type            variableHeightFlowRateInletVelocity;
    flowRate        0.2;
    value           uniform (0 0 0); // placeholder
}
```

Note:
- the value is positive into the domain
- may not work correctly for transonic inlets
- strange behaviour with potentialFoam since the momentum equation is not solved

1.67 waveSurfacePressure

This is a pressure boundary condition, whose value is calculated as the hydrostatic pressure based on a given displacement:

$$p = -\rho * g * \zeta \tag{1.17}$$

ρ : density [kg/m3]
g : acceleration due to gravity [m/s2]
ζ : wave amplitude [m]

The wave amplitude is updated as part of the calculation, derived from the local volumetric flux.

Property	Description	Required	Default value
phi	flux field name	no	phi
rho	density field name	no	rho
zeta	wave amplitude field name	no	zeta

Example

```
myPatch
{
    type            waveSurfacePressure;
    phi             phi;
    rho             rho;
    zeta            zeta;
    value           uniform 0;  // place holder
}
```

The density field is only required if the flux is mass-based as opposed to volumetric-based.

1.68 waveTransmissive

This boundary condition provides a wave transmissive outflow condition, based onsolving DDt(psi, U) = 0 at the boundary.

$$x_p = \frac{\phi_p}{|Sf|} + \sqrt{\frac{\gamma}{\psi_p}} \tag{1.18}$$

x_p : patch values
ϕ_p : patch face flux
ψ_p : patch compressibility
Sf : patch face area vector
γ : ratio of specific heats

Property	Description	Required	Default value
phi	flux field name	no	phi
rho	density field name	no	rho
psi	compressibility field name	no	psi
gamma	ratio of specific heats (Cp/Cv)	yes	

Example

```
myPatch
{
    type            waveTransmissive;
    phi             phi;
    psi             psi;
    gamma           1.4;
}
```

2 Turbulence and thermal boundary conditions

2.1 externalCoupledTemperatureMixed

This boundary condition provides a temperatue interface to an external application. Values are transferred as plain text files, where OpenFOAM data is written as:

Patch: <patch name>
<magSf1> <value1> <qDot1> <htc1>
<magSf2> <value2> <qDot2> <htc2>
<magSf3> <value3> <qDot3> <htc2>
...
<magSfN> <valueN> <qDotN> <htcN>

and received as the constituent pieces of the 'mixed' condition, i.e.

Patch: <patch name>
<value1> <gradient1> <valueFracion1>
<value2> <gradient2> <valueFracion2>
<value3> <gradient3> <valueFracion3>
...
<valueN> <gradientN> <valueFracionN>

Data is sent/received as a single file for all patches from the directory

$FOAM_CASE/<commsDir>

At start-up, the boundary creates a lock file, i.e..

OpenFOAM.lock

... to signal the external source to wait. During the boundary condition update, boundary values are written to file, e.g.

<fileName>.out

The lock file is then removed, instructing the external source to take control of the program execution. When ready, the external program should create the return values, e.g. to file

<fileName>.in

... and then re-instate the lock file. The boundary condition will then read the return values, and pass program execution back to OpenFOAM.

Property	Description	Required	Default value
commsDir	communications directory	yes	
fileName	transfer file name	yes	
waitInterval	interval [s] between file chekcs	no	
timeOut	time after which error invoked [s]	no	100waitInterval
calcFrequency	calculation frequency	no	1
log	log program control	no	no

Example

```
myPatch
{
    type            externalCoupledTemperature;
    commsDir        "$FOAM_CASE/comms";
    fileName        data;
    calcFrequency   1;
}
```

2.2 externalWallHeatFluxTemperature

This boundary condition supplies a heat flux condition for temperature on an external wall. Optional thin thermal layer resistances can be specified through thicknessLayers and kappaLayers entries for the fixed heat transfer coefficient mode.

The condition can operate in two modes:
- fixed heat transfer coefficient: supply h and Ta
- fixed heat flux: supply q

where
h = heat transfer coefficient [W/m2/K]
Ta = ambient temperature [K]
q = heat flux [W/m2]

The thermal conductivity, κ, can either be retrieved from the mesh database using the *lookup* option, or from a *solidThermo* thermophysical package.

Property	Description	Required	Default value
kappa	thermal conductivity option	yes	
q	heat flux [W/m2]me	yes	
Ta	ambient temperature [K]	yes	
h	heat transfer coefficient [W/m/K]	yes	
thicknessLayers	list of thickness per layer [m]	no	
kappaLayers	list of thermal conductivities per layer [W/m/K]	no	
kappaName	name of thermal conductivity field	yes	

Example

```
myPatch
{
    type            externalWallHeatFluxTemperature;
    kappa           fluidThermo; // solidThermo, lookup,
        directionalSolidThermo
    q               uniform 1000;
    Ta              uniform 300.0;
    h               uniform 10.0;
    thicknessLayers (0.1 0.2 0.3 0.4); // thickness of solid walls
    kappaLayers     (1 2 3 4); // kappa for each solid walls
    value           uniform 300.0;
    kappaName       none;
```

```
    }
```

Note:

- Only supply h and Ta, or q in the dictionary (see above)
- kappa entries can be: fluidThermo, solidThermo or lookup

2.3 thermalBaffle1D

This BC solves a steady 1D thermal baffle. The solid properties are specify as dictionary. Optionaly radiative heat flux (Qr) can be incorporated into the balance. Some under-relaxation might be needed on Qr.
Baffle and solid properties need to be specified on the master side of the baffle.

Example

```
myPatch_master
{
    type   compressible::thermalBaffle1D<hConstSolidThermoPhysics>;
    samplePatch     myPatch_slave;

    thickness       uniform 0.005;  // thickness [m]
    Qs              uniform 100;    // heat flux [W/m2]
    Qr              none;
    relaxation      0;

    // Solid thermo
    specie
    {
        nMoles          1;
        molWeight       20;
    }
Specifies gradient and temperature such that the equations are the same
on both sides:
- refGradient = zero gradient
- refValue = neighbour value
- mixFraction = nbrKDelta / (nbrKDelta + myKDelta())

where KDelta is heat-transfer coefficient K * deltaCoeffs
    transport
    {
        kappa           1;
    }
    thermodynamics
    {
        Hf              0;
        Cp              10;
    }
    equationOfState
    {
        rho             10;
    }

    value               uniform 300;
}

myPatch_slave
```

```
{
    type   compressible::thermalBaffle1D<hConstSolidThermoPhysics>;
    samplePatch     myPatch_master_master;

    Qr              none;
    relaxation      0;
}
```

2.4 totalFlowRateAdvectiveDiffusive

This BC is used for species inlets. The diffusion and advection fluxes are considered to calculate the inlet value for the species The massFluxFraction sets the fraction of the flux of each particular species.

2.5 turbulentHeatFluxTemperature

Fixed heat boundary condition to specify temperature gradient. Input heat source either specified in terms of an absolute power [W], or as a flux [W/m2].

Example

```
myPatch
{
        type            compressible::turbulentHeatFluxTemperature;
        heatSource      flux;        // power [W]; flux [W/m2]
        q               uniform 10;  // heat power or flux
        kappa           fluidThermo; // calculate kappa=alphaEff*thermo.Cp
        Qr              none;        // name of the radiative flux
        value           uniform 300; // initial temperature value
}
```

2.6 turbulentTemperatureCoupledBaffleMixed

Mixed boundary condition for temperature, to be used for heat-transfer on back-to-back baffles.Optional thin thermal layer resistances can be specified through thicknessLayers and kappaLayers entries.

The thermal conductivity, κ, can either be retrieved from the mesh database using the *lookup* option, or from a *solidThermo* or $fluidThermo$ thermophysical package.

Specifies gradient and temperature such that the equations are the same on both sides:
- refGradient = zero gradient
- refValue = neighbour value
- mixFraction = nbrKDelta / (nbrKDelta + myKDelta())

where KDelta is heat-transfer coefficient K * deltaCoeffs

Property	Description	Required	Default value
kappa	thermal conductivity option	yes	
kappaName	name of thermal conductivity field	no	T
Tnbr	name of the field	no	
thicknessLayers	list of thicknesses per layer [m]	no	
kappaLayers	list of thermal conductivities per layer [W/m/K]	no	

Example

```
myPatch
{
    type            compressible::turbulentTemperatureCoupledBaffleMixed;
    Tnbr            T;
    kappa           lookup;
    KappaName       kappa;
    thicknessLayers (0.1 0.2 0.3 0.4);
    kappaLayers     (1 2 3 4)
    value           uniform 300;
}
```

Needs to be on underlying mapped(Wall)FvPatch.

Note: kappa : heat conduction at patch. Gets supplied how to lookup calculate kappa:
- 'lookup' : lookup volScalarField (or volSymmTensorField) with name
- 'fluidThermo' : use fluidThermo and compressible::RASmodel to calculate kappa
- 'solidThermo' : use solidThermo kappa()

- 'directionalSolidThermo' directionalKappa()

2.7 turbulentTemperatureRadCoupledMixed

Mixed boundary condition for temperature and radiation heat transfer to be used for in multiregion cases. Optional thin thermal layer resistances can be specified through thicknessLayers and kappaLayers entries.

The thermal conductivity, κ, can either be retrieved from the mesh database using the *lookup* option, or from a *solidThermo* or *fluidThermo* thermophysical package.

Property	Description	Required	Default value
kappa	thermal conductivity option	yes	
kappaName	name of thermal conductivity field	no	T
Tnbr	name of the field	no	
QrNbr	name of the radiative flux in the nbr region	no	none
Qr	name of the radiative flux in this region	no	none
thicknessLayers	list of thicknesses per layer [m]	no	
kappaLayers	list of thermal conductivities per layer [W/m/K]	no	

Example

```
myPatch
{
    type            compressible::turbulentTemperatureRadCoupledMixed;
    Tnbr            T;
    kappa           lookup;
    KappaName       kappa;
    QrNbr           Qr; // or none. Name of Qr field on neighbour region
    Qr              Qr; // or none. Name of Qr field on local region
    thicknessLayers (0.1 0.2 0.3 0.4);
    kappaLayers     (1 2 3 4)
    value           uniform 300;
}
```

Needs to be on underlying mapped(Wall)FvPatch.

Note: kappa : heat conduction at patch. Gets supplied how to lookup/calculate kappa:
- 'lookup' : lookup volScalarField (or volSymmTensorField) with name
- 'fluidThermo' : use fluidThermo and compressible::RASmodel to calculate K
- 'solidThermo' : use solidThermo kappa()
- 'directionalSolidThermo' directionalKappa()

2.8 wallHeatTransfer

This boundary condition provides an enthalpy condition for wall heat transfer

Property	Description	Required	Default value
Tinf	wall temperature	yes	
alphaWall	thermal diffusivity	yes	

Example

```
myPatch
{
    type            wallHeatTransfer;
    Tif             uniform 500; //ambient temperature[K]
    alphaWall       uniform 1; // thermal diffusivity [W/m2]
    value           uniform 300;
}
```

2.9 convectiveHeatTransfer

This boundary condition provides a convective heat transfer coefficient condition

if Re $>$ 500000

$$htc_p = \frac{0.664 Re^{0.5} Pr^{0.333} \kappa_p}{L} \tag{2.1}$$

else

$$htc_p = \frac{0.037 Re^{0.8} Pr^{0.333} \kappa_p}{L} \tag{2.2}$$

htc_p : patch convective heat transfer coefficient
Re : Reynolds number
Pr : Prandtl number
κ_p : thermal conductivity
L : length scale

Property	Description	Required	Default value
L	Length scale [m]	yes	

Example

```
myPatch
{
    type            convectiveHeatTransfer;
    L               0.1;
}
```

2.10 turbulentMixingLengthDissipationRateInlet

This boundary condition provides a turbulence dissipation, ϵ (epsilon) inlet condition based on a specified mixing length. The patch values are calculated using:

$$\epsilon_p = \frac{C_\mu^{0.75} k^{1.5}}{L} \tag{2.3}$$

ϵ_p : patch epsilon values
C_μ : Model coefficient, set to 0.09
k : turbulence kinetic energy
L : length scale

Property	Description	Required	Default value
mixingLength	Length scale [m]	yes	
phi	flux field name	no	phi
k	turbulence kinetic energy field name	no	k

Example

```
myPatch
{
    type            compressible::turbulentMixingLengthDissipationRateInlet;
    mixingLength    0.005;
    value           uniform 200;   // placeholder
}
```

Note:
In the event of reverse flow, a zero-gradient condition is applied

2.11 turbulentMixingLengthFrequencyInlet

This boundary condition provides a turbulence specific dissipation, ω (omega) inlet condition based on a specified mixing length. The patch values are calculated using:

$$\omega_p = \frac{k^{0.5}}{C_\mu^{0.25} L} \tag{2.4}$$

ω_p : patch omega values
C_μ : Model coefficient, set to 0.09
k : turbulence kinetic energy
L : length scale

Property	Description	Required	Default value
mixingLength	Length scale [m]	yes	
phi	flux field name	no	phi
k	turbulence kinetic energy field name	no	k

Example

```
myPatch
{
    type            compressible::turbulentMixingLengthFrequencyInlet;
    mixingLength    0.005;
    value           uniform 200;   // placeholder
}
```

Note:
In the event of reverse flow, a zero-gradient condition is applied

2.12 atmBoundaryLayerInletEpsilon

This boundary condition specifies an inlet value for the turbulence dissipation, ϵ (*epsilon*), appropriate for atmospheric boundary layers (ABL), and designed to be used in conjunction with the *ABLInletVelocity* inlet velocity boundary condition.

$$\epsilon = \frac{(U^*)^3}{K(z - z_g + z_0)} \tag{2.5}$$

U^* : frictional velocity
K : Karman's constant
z : vertical co-ordinate [m]
z_0 : surface roughness length [m]
z_g : minimum vlaue in z direction [m]

and:

$$U^* = K\frac{U_{ref}}{ln\left(\frac{Z_{ref}+z_0}{z_0}\right)} \tag{2.6}$$

U_{ref} : reference velocity at Z_{ref} [m/s]
Z_{ref} : reference height [m]

Property	Description	Required	Default value
z	vertical co-ordinate [m]	yes	
kappa	Karman's constanat	no	0.41
Uref	reference velocity [m/s]	yes	
Href	reference height [m]	yes	
z0	surface roughness length [m]	yes	
zGround	minimum z co-ordinate [m]	yes	

Example

```
myPatch
{
    type            atmBoundaryLayerInletEpsilon;
    z               1.0;
    kappa           0.41;
    Uref            1.0;
    Href            0.0;
    z0              uniform 0.0;
```

```
        zGround         uniform 0.0;
    }
```

Reference:

D.M. Hargreaves and N.G. Wright, "On the use of the k-epsilon model in commercial CFD software to model the neutral atmospheric boundary layer", Journal of Wind Engineering and Industrial Aerodynamics 95(2007), pp 355-369.

2.13 atmBoundaryLayerInletVelocity

This boundary condition specifies a velocity inlet profile appropriate for atmospheric boundary layers (ABL). The profile is derived from the friction velocity, flow direction and the direction of the parabolic co-ordinate z.

$$U = \frac{U^*}{K} ln\left(\frac{z - z_g + z_0}{z_0}\right) \tag{2.7}$$

U^* : frictional velocity
K : Karman's constant
z : vertical co-ordinate [m]
z_0 : surface roughness length [m]
z_g : minimum vlaue in z direction [m]

and:

$$U^* = K \frac{U_{ref}}{ln\left(\frac{Z_{ref}+z_0}{z_0}\right)} \tag{2.8}$$

U_{ref} : reference velocity at Z_{ref} [m/s]
Z_{ref} : reference height [m]

Reference:
D.M. Hargreaves and N.G. Wright, "On the use of the k-epsilon model in commercial CFD software to model the neutral atmospheric boundary layer", Journal of Wind Engineering and Industrial Aerodynamics 95(2007), pp 355-369.

Property	Description	Required	Default value
n	flow direction	yes	
z	vertical co-ordinate [m]	yes	
kappa	Karman's constanat	no	0.41
Uref	reference velocity [m/s]	yes	
Href	reference height [m]	yes	
z0	surface roughness length [m]	yes	
zGround	minimum z co-ordinate [m]	yes	

Example

```
myPatch
```

```
{
    type            atmBoundaryLayerInletVelocity;
    n               (0 1 0);
    z               1.0;
    kappa           0.41;
    Uref            1.0;
    Href            0.0;
    z0              uniform 0.0;
    zGround         uniform 0.0;
}
```

Note:
D.M. Hargreaves and N.G. Wright recommend Gamma epsilon in the k-epsilon model should be changed from 1.3 to 1.11 for consistency. The roughness height (Er) is given by Er = 20 z0 following the same reference.

2.14 turbulentHeatFluxTemperature

Fixed heat boundary condition to specify temperature gradient. Input heat source either specified in terms of an absolute power [W], or as a flux [W/m2].

Property	Description	Required	Default value
heatSource	heat source type: $flux[W/m2]$ or $power[W]$	yes	
q	heat source value	yes	
alphaEff	turbulent thermal diffusivity field name	yes	

Example

```
myPatch
{
    type            turbulentHeatFluxTemperature;
    heatSource      flux;
    q               uniform 10;
    alphaEff        alphaEff;
    value           uniform 300; // place holder
}
```

Note :
- it is assumed that the units of α_{eff} are [kg/m/s]
- the specific heat capcaity is read from the transport dictionary entry $Cp0$

3 Wall Functions

3.1 compressible::alphatJayatillekeWallFunction

This boundary condition provides a thermal wall function for turbulent thermal diffusivity (usually α_t) based on the Jayatilleke model.

Property	Description	Required	Default value
Prt	turbulent Prandtl number	no	0.85
Cmu	model coefficient	no	0.09
kappa	Von Karman constant	no	0.41
E	model coefficient	no	9.8

Example

```
myPatch
{
    type            alphatJayatillekeWallFunction;
    Prt             0.85;
    kappa           0.41;
    E               9.8;
    value           uniform 0; // optional value entry
}
```

3.2 compressible::alphatWallFunction

This boundary condition provides a turbulent thermal diffusivity conditon
when using wall functions
- replicates OpenFOAM v1.5 (and earlier) behaviour

The turbulent thermal diffusivity calculated using:

$$\alpha_t = \frac{\mu_t}{Pr_t} \tag{3.1}$$

α_t : turblence thermal diffusivity
μ_t : turblence viscosity
Pr_t : turblent Prandtl number

Property	Description	Required	Default value
mut	turbulence viscosity field name	no	mut
Prt	turbulent Prandtl number	no	0.85

Example

```
myPatch
{
    type            alphatWallFunction;
    mut             mut;
    Prt             0.85;
    value           uniform 0; // optional value entry
}
```

3.3 compressible::epsilonLowReWallFunction

This boundary condition provides a turbulence dissipation wall function condition for low- and high-Reynolds number turbulent flow cases.

The condition can be applied to wall boundaries, whereby it inserts near wall epsilon values directly into the epsilon equation to act as a constraint.

The model operates in two modes, based on the computed laminar-to-turbulent switch-over y+ value derived from kappa and E.

Property	Description	Required	Default value
Cmu	model coefficient	no	0.09
kappa	Von Karman constant	no	0.41
E	model coefficient	no	9.8

Example

```
myPatch
{
    type            epsilonLowReWallFunction;
}
```

3.4 compressible::epsilonWallFunction

This boundary condition provides a turbulence dissipation wall function condition for high Reynolds number, turbulent flow cases.

The condition can be applied to wall boundaries, whereby it
- calculates ϵ and G
- inserts near wall epsilon values directly into the epsilon equation to act as a constraint

ϵ : turblence dissipation field
G : turblence generation field

Property	Description	Required	Default value
Cmu	model coefficient	no	0.09
kappa	Von Karman constant	no	0.41
E	model coefficient	no	9.8

Example

```
myPatch
{
    type            compressible::epsilonWallFunction;
}
```

3.5 fWallFunction

This boundary condition provides a turbulence damping function, f, wall function condition for low- and high Reynolds number, turbulent flow cases

The model operates in two modes, based on the computed laminar-to-turbulent switch-over y+ value derived from kappa and E.

Property	Description	Required	Default value
Cmu	model coefficient	no	0.09
kappa	Von Karman constant	no	0.41
E	model coefficient	no	9.8

Example

```
myPatch
{
    type            fWallFunction;
}
```

3.6 compressible::kLowReWallFunction

This boundary condition provides a turbulence kinetic energy wall function condition for low- and high-Reynolds number turbulent flow cases.

The model operates in two modes, based on the computed laminar-to-turbulent switch-over y+ value derived from kappa and E.

Property	Description	Required	Default value
Cmu	model coefficient	no	0.09
kappa	Von Karman constant	no	0.41
E	model coefficient	no	9.8
Ceps2	model coefficient	no	1.9

Example

```
myPatch
{         type            kLowReWallFunction;
}
```

3.7 compressible::kqRWallFunction

This boundary condition is applied to turbulence k, q, and R when using wall functions, and simply enforces a zero-gradient condition.

Example

```
myPatch
{
    type            compressible::kqRWallFunction;
}
```

3.8 compressible::mutkRoughWallFunction

This boundary condition provides a turbulent viscosity condition when using wall functions for rough walls, based on turbulence kinetic energy. The condition manipulates the E parameter to account for roughness effects.

Parameter ranges
- roughness height = sand-grain roughness (0 for smooth walls)
- roughness constant = 0.5-1.0

Property	Description	Required	Default value
Ks	sand-grain roughness height	yes	
Cs	roughness constant	yes	

Example

```
myPatch
{
    type            mutkRoughWallFunction;
    Ks              uniform 0;
    Cs              uniform 0.5;
}
```

3.9 compressible::mutkWallFunction

This boundary condition provides a turbulent viscosity condition when using wall functions, based on turbulence kinetic energy.
- replicates OpenFOAM v1.5 (and earlier) behaviour

Example

```
myPatch
{
    type            mutkWallFunction;
}
```

3.10 compressible::mutLowReWallFunction

This boundary condition provides a turbulent viscosity condition for use with low Reynolds number models. It sets *nut* to zero, and provides an access function to calculate y+.

Example

```
myPatch
{
    type            mutLowReWallFunction;
}
```

3.11 compressible::mutURoughWallFunction

This boundary condition provides a turbulent viscosity condition when using wall functions for rough walls, based on velocity.

Property	Description	Required	Default value
roughnessHeight	roughness height	yes	
roughnessConstant	roughness constanr	yes	
roughnessFactor	scaling factor	yes	

Example

```
myPatch
{
    type            mutURoughWallFunction;
    roughnessHeight 1e-5;
    roughnessConstant 0.5;
    roughnessFactor 1;
}
```

3.12 compressible::mutUSpaldingWallFunction

This boundary condition provides a turbulent viscosity condition when using wall functions for rough walls, based on velocity, using Spalding's law to give a continuous nut profile to the wall (y+ = 0)

$$y^+ = u^+ + \frac{1}{E}\left[exp(\kappa u^+) - 1 - \kappa u^+ - 0.5(\kappa u^+)^2 - \frac{1}{6}(\kappa u^+)^3\right] \tag{3.2}$$

y^+ : non-dimensional position
u^+ : non-dimensional velocity
κ : Von Karman constant

Example

```
myPatch
{
    type            mutUSpaldingWallFunction;
}
```

3.13 compressible::mutUWallFunction

This boundary condition provides a turbulent viscosity condition when using wall functions, based on velocity.

Example

```
myPatch
{
    type            mutUWallFunction;
}
```

3.14 compressible::mutWallFunction

This boundary condition provides a turbulent viscosity condition when using wall functions, based on turbulence kinetic energy.
- replicates OpenFOAM v1.5 (and earlier) behaviour

Example

```
myPatch
{
    type            mutWallFunction;
}
```

3.15 compressible::omegaWallFunction

This boundary condition provides a wall function constraint on turbulnce specific dissipation, omega. The values are computed using:

$$\omega = sqrt(\omega_{vis}^2 + \omega_{log}^2) \tag{3.3}$$

ω_{vis} : omega in viscous region
ω_{log} : omega in logarithmic region

Menter, F., Esch, T.
"Elements of Industrial Heat Transfer Prediction" 16th Brazilian Congress of Mechanical Engineering (COBEM), Nov. 2001

Property	Description	Required	Default value
Cmu	model coefficient	no	0.09
kappa	Von Karman constant	no	0.41
E	model coefficient	no	9.8
beta1	model coefficient	no	0.075

Example

```
myPatch
{
    type            compressible::omegaWallFunction;
}
```

3.16 compressible::v2WallFunction

This boundary condition provides a turbulence stress normal to streamlines wall function condition for low- and high-Reynolds number, turbulent flow cases.

The model operates in two modes, based on the computed laminar-to-turbulent switch-over y+ value derived from kappa and E.

Property	Description	Required	Default value
Cmu	model coefficient	no	0.09
kappa	Von Karman constant	no	0.41
E	model coefficient	no	9.8

Example

```
myPatch
{
    type            v2WallFunction;
}
```

3.17 incompressible::alphatJayatillekeWallFunction

This boundary condition provides a kinematic turbulent thermal conductivity for using wall functions, using the Jayatilleke 'P' function.

Property	Description	Required	Default value
Prt	turbulent Prandtl number	no	0.85
Cmu	model coefficient	no	0.09
kappa	Von Karman constant	no	0.41
E	model coefficient	no	9.8

Example

```
myPatch
{
    type            alphatJayatillekeWallFunction;
}
```

Note:
The units of kinematic turbulent thermal conductivity are [m2/s]

3.18 incompressible::epsilonLowReWallFunction

This boundary condition provides a turbulence dissipation wall function condition for low- and high-Reynolds number turbulent flow cases.

The condition can be applied to wall boundaries, whereby it inserts near wall epsilon values directly into the epsilon equation to act as a constraint.

The model operates in two modes, based on the computed laminar-to-turbulent switch-over y+ value derived from kappa and E.

Property	Description	Required	Default value
Cmu	model coefficient	no	0.09
kappa	Von Karman constant	no	0.41
E	model coefficient	no	9.8

Example

```
myPatch
{
    type            epsilonLowReWallFunction;
}
```

3.19 incompressible::epsilonWallFunction

This boundary condition provides a turbulence dissipation wall function condition for high Reynolds number, turbulent flow cases.

The condition can be applied to wall boundaries, whereby it
- calculates ϵ and G
- inserts near wall epsilon values directly into the epsilon equation to act as a constraint

ϵ : turblence dissipation field
G : turblence generation field

Property	Description	Required	Default value
Cmu	model coefficient	no	0.09
kappa	Von Karman constant	no	0.41
E	model coefficient	no	9.8

Example

```
myPatch
{
    type            epsilonWallFunction;
}
```

3.20 incompressible::kqRWallFunction

This boundary condition is applied to turbulence k, q, and R when using wall functions, and simply enforces a zero-gradient condition.

Example

```
myPatch
{
    type            kqRWallFunction;
}
```

3.21 incompressible::nutkAtmRoughWallFunction

This boundary condition provides a turbulent kinematic viscosity for atmospheric velocity profiles. It is desinged to be used in conjunction with the atmBoundaryLayerInletVelocity boundary condition. The values are calculated using:

$$U = fracU_f K ln(\frac{z + z_0}{z_0}) \tag{3.4}$$

U_f : frictional velocity
K : Von Karman's constant
z_0 : surface roughness length
z : vertical co-ordinate

Property	Description	Required	Default value
z0	surface roughness length	yes	

Example

```
myPatch
{
    type            nutkAtmRoughWallFunction;
    z0              uniform 0;
}
```

3.22 incompressible::nutkRoughWallFunction

This boundary condition provides a turbulent kinematic viscosity condition when using wall functions for rough walls, based on turbulence kinetic energy. The condition manipulates the E parameter to account for roughness effects.

Parameter ranges
- roughness height = sand-grain roughness (0 for smooth walls)
- roughness constant = 0.5-1.0

Property	Description	Required	Default value
Ks	sand-grain roughness height	yes	
Cs	roughness constant	yes	

Example

```
myPatch
{
    type            nutkRoughWallFunction;
    Ks              uniform 0;
    Cs              uniform 0.5;
}
```

3.23 incompressible::nutkWallFunction

This boundary condition provides a turbulent kinematic viscosity condition when using wall functions, based on turbulence kinetic energy.
- replicates OpenFOAM v1.5 (and earlier) behaviour

Example

```
myPatch
{
    type            nutkWallFunction;
}
```

3.24 incompressible::nutLowReWallFunction

This boundary condition provides a turbulent kinematic viscosity condition for use with low Reynolds number models. It sets *nut* to zero, and provides an access function to calculate y+.

Example

```
myPatch
{
    type            nutLowReWallFunction;
}
```

3.25 incompressible::nutURoughWallFunction

This boundary condition provides a turbulent kinematic viscosity condition when using wall functions for rough walls, based on velocity.

Property	Description	Required	Default value
roughnessHeight	roughness height	yes	
roughnessConstant	roughness constanr	yes	
roughnessFactor	scaling factor	yes	

Example

```
myPatch
{
    type            nutURoughWallFunction;
    roughnessHeight 1e-5;
    roughnessConstant 0.5;
    roughnessFactor 1;
}
```

3.26 incompressible::nutUSpaldingWallFunction

This boundary condition provides a turbulent kinematic viscosity condition when using wall functions for rough walls, based on velocity, using Spalding's law to give a continuous nut profile to the wall (y+ = 0)

$$y^+ = u^+ + \frac{1}{E}\left[exp(\kappa u^+) - 1 - \kappa u^+ - 0.5(\kappa u^+)^2 - \frac{1}{6}(\kappa u^+)^3\right] \tag{3.5}$$

y^+ : non-dimensional position
u^+ : non-dimensional velocity
κ : Von Karman constant

Example

```
myPatch
{
    type            nutUSpaldingWallFunction;
}
```

3.27 incompressible::nutUTabulatedWallFunction

This boundary condition provides a turbulent kinematic viscosity condition when using wall functions. As input, the user specifies a look-up table of U+ as a function of near-wall Reynolds number. The table should be located in the $FOAM_CASE/constant folder.

Property	Description	Required	Default value
uPlusTable	U+ as a function of Re table name	yes	

Example

```
myPatch
{
    type            nutTabulatedWallFunction;
    uPlusTable      myUPlusTable;
}
```

Note:
The tables are not registered since the same table object may be used for more than one patch.

3.28 incompressible::nutUWallFunction

This boundary condition provides a turbulent kinematic viscosity condition when using wall functions, based on velocity.

Example

```
myPatch
{
    type            nutUWallFunction;
}
```

3.29 incompressible::nutWallFunction

This boundary condition provides a turbulent kinematic viscosity condition when using wall functions, based on turbulence kinetic energy.
- replicates OpenFOAM v1.5 (and earlier) behaviour

Example

```
myPatch
{
    type            nutWallFunction;
}
```

4 Radiation boundary conditions

4.1 greyDiffusiveRadiationMixed

This boundary condition provides a grey-diffuse condition for radiation intensity, I, for use with the finite-volume discrete-ordinates model (fvDOM), in which the radiation temperature is retrieved from the temperature field boundary condition.

Property	Description	Required	Default value
T	temperature field name	no	T
emissivityMode	emissivity mode: solidThermo or lookup	yes	

Example

```
myPatch
{
    type            greyDiffusiveRadiation;
    T               T;
    emissivityMode  solidThermo;
    value           uniform 0;
}
```

4.2 greyDiffusiveViewFactor

This boundary condition provides a grey-diffuse condition for radiative heat flux, Qr, for use with the view factor model

Property	Description	Required	Default value
Qro	external radiative heat flux	yes	
emissivityMode	emissivity mode: solidThermo or lookup	yes	

Example

```
myPatch
{
    type            greyDiffusiveRadiationViewFactor;
    Qro             uniform 0;
    emissivityMode  solidThermo;
    value           uniform 0;
}
```

4.3 MarshakRadiation

A 'mixed' boundary condition that implements a Marshak condition for the incident radiation field (usually written as G)

The radiation temperature is retrieved from the mesh database, using a user specified temperature field name.

Property	Description	Required	Default value
T	temperature field name	no	T

Example

```
myPatch
{
    type            MarshakRadiation;
    T               T;
    value           uniform 0;
}
```

Note:
In the event of reverse flow, a zero-gradient condition is applied

4.4 MarshakRadiationFixedTemperature

A 'mixed' boundary condition that implements a Marshak condition for the incident radiation field (usually written as G)

The radiation temperature field across the patch is supplied by the user using the $Trad$ entry.

Property	Description	Required	Default value
T	temperature field name	no	T

Example

```
myPatch
{
    type            MarshakRadiationFixedTemperature;
    Trad            uniform 1000;       // radiation temperature field
    value           uniform 0;          // place holder
}
```

Note:
In the event of reverse flow, a zero-gradient condition is applied

4.5 wideBandDiffusiveRadiation

This boundary condition provides a wide-band, diffusive radiation condition, where the patch temperature is specified.

Property	Description	Required	Default value
T	temperature field name	no	T

Example

```
myPatch
{
    type            wideBandDiffusiveRadiation;
    value           uniform 0;
}
```