

```

27 Description
28 PIMPLE control class to supply convergence information/checks for
29 the PIMPLE loop.
30 May also be used to fix PISO-based algorithms as PISO controls are a
31 sub-set of PIMPLE controls.
32
33
34
35
36 #ifndef pimpleControl_H
37 #define pimpleControl_H
38
39 #include "solutionControl.H"
40
41 // Declare that pimpleControl will be used
42 #define PIMPLE_CONTROL
43
44 // *****
45
46 namespace Foam
47 {
48
49 // *****
50
51 // *****
52
53 class pimpleControl
54 {
55 public:
56     pimpleControl() {}
57     // Private member functions
58     // Disallow default Mixture copy construct
59     pimpleControl(const pimpleControl&);
60     // Disallow default bitwise assignment
61     void operator=(const pimpleControl&);
62
63 protected:
64     // Protected data
65     // Solution controls
66     bool solveFlow;
67     // Flag to indicate whether to solve for the flow
68     label nCorrPIMPLE;
69     // Maximum number of PIMPLE correctors
70     label nCorrPISO;
71     // Maximum number of PISO correctors
72     label nCorrPISO;
73     // Current PISO corrector
74     label corrPISO;
75     // Flag to indicate whether to update density in SIMPLE
76     rather than PISO mode
77     bool SIMPLErho;
78     // Flag to indicate whether to only solve turbulence on final iter
79     bool turbOnFinalIterOnly;
80     // Converged flag
81     bool converged;
82
83 // Protected member functions
84 // Read controls from fvSolution dictionary
85 virtual void read();
86 // Return true if all convergence checks are satisfied
87 virtual bool criteriaSatisfied();
88
89
90
91
92
93
94
95
96
97
98
99
100
101

```

```

102 public:
103 // Static Data Members
104 // *****
105 // Run-time type information
106 TypeName("pimpleControl");
107
108 // Constructors
109 // *****
110 // Construct from mesh and the name of control sub-dictionary
111 pimpleControl(Foam::mesh& mesh, const word dictName "PIMPLE");
112
113 // Destructor
114 virtual ~pimpleControl();
115
116 // Member Functions
117 // *****
118 // Access
119 // *****
120 // Maximum number of PIMPLE correctors
121 inline label nCorrPIMPLE() const;
122 // Maximum number of PISO correctors
123 inline label nCorrPISO() const;
124 // Current PISO corrector index
125 inline label corrPISO() const;
126 // Flag to indicate whether to update density in SIMPLE
127 inline bool SIMPLErho() const;
128 // Solution control
129 virtual bool loop();
130 // Pressure corrector loop control
131 inline bool correct();
132 // Return true to store the initial residuals
133 inline bool storeInitialResiduals() const;
134 // Return true for first PIMPLE (outer) iteration
135 inline bool firstIter() const;
136 // Return true for final PIMPLE (outer) iteration
137 inline bool finalIter() const;
138 // Return true for final inner iteration
139 inline bool finalInnerIter() const;
140 // Return true to solve for flow
141 inline bool solveFlow() const;
142 // Return true to solve for turbulence
143 inline bool turbCorr() const;
144
145 // *****
146 // End namespace Foam
147
148 #endif

```

```

102 inline bool Foam::pimpleControl::correct()
103 {
104     corrPISO++;
105     if (debug)
106     {
107         Info<< algorithmName << " correct: corrPISO = " << corrPISO << endl;
108     }
109     if (corrPISO == nCorrPISO)
110     {
111         return true;
112     }
113     else
114     {
115         corrPISO = 1;
116         return false;
117     }
118 }

```

```

102 #include "pimpleControl.H"
103 #include "barotropicCompressibilityModel.H"
104 #include "basicThermo.H"
105 #include "chemistryModel.H"
106 #include "laminarTurbulenceModel.H"
107 #include "LESModel.H"
108 #include "turbulenceModel.H"
109 #include "transportModel.H"
110 #include "twoPhaseMixture.H"
111 #include "viscosityModel.H"
112
113 namespace Foam
114 {
115     namespace Incompressible
116     {
117         typedef IncompressibleTurbulenceModel<transportModel> turbulenceModel;
118         typedef laminarTurbulenceModel<laminarModel> laminarModel;
119         typedef LESModel<turbulenceModel> LESModel;
120         template<class BasicCompressibleTurbulenceModel>
121         autoPtr<BasicCompressibleTurbulenceModel> New
122         (
123             const volVectorField& U,
124             const surfaceScalarField& phi,
125             const typeName BasicCompressibleTurbulenceModel::transportModel
126             transport,
127             const word& propertiesName = turbulenceModel::propertiesName
128         )
129         {
130             return autoPtr<BasicCompressibleTurbulenceModel>::New
131             (
132                 transport,
133                 propertiesName
134             );
135         }
136     }
137 }

```

```

102 #include "pimpleControl.H"
103 #include "barotropicCompressibilityModel.H"
104 #include "basicThermo.H"
105 #include "chemistryModel.H"
106 #include "laminarTurbulenceModel.H"
107 #include "LESModel.H"
108 #include "turbulenceModel.H"
109 #include "transportModel.H"
110 #include "twoPhaseMixture.H"
111 #include "viscosityModel.H"
112
113 namespace Foam
114 {
115     namespace Incompressible
116     {
117         typedef IncompressibleTurbulenceModel<transportModel> turbulenceModel;
118         typedef laminarTurbulenceModel<laminarModel> laminarModel;
119         typedef LESModel<turbulenceModel> LESModel;
120         template<class BasicCompressibleTurbulenceModel>
121         autoPtr<BasicCompressibleTurbulenceModel> New
122         (
123             const volVectorField& U,
124             const surfaceScalarField& phi,
125             const typeName BasicCompressibleTurbulenceModel::transportModel
126             transport,
127             const word& propertiesName = turbulenceModel::propertiesName
128         )
129         {
130             return autoPtr<BasicCompressibleTurbulenceModel>::New
131             (
132                 transport,
133                 propertiesName
134             );
135         }
136     }
137 }

```

```

102 #include "pimpleControl.H"
103 #include "barotropicCompressibilityModel.H"
104 #include "basicThermo.H"
105 #include "chemistryModel.H"
106 #include "laminarTurbulenceModel.H"
107 #include "LESModel.H"
108 #include "turbulenceModel.H"
109 #include "transportModel.H"
110 #include "twoPhaseMixture.H"
111 #include "viscosityModel.H"
112
113 namespace Foam
114 {
115     namespace Incompressible
116     {
117         typedef IncompressibleTurbulenceModel<transportModel> turbulenceModel;
118         typedef laminarTurbulenceModel<laminarModel> laminarModel;
119         typedef LESModel<turbulenceModel> LESModel;
120         template<class BasicCompressibleTurbulenceModel>
121         autoPtr<BasicCompressibleTurbulenceModel> New
122         (
123             const volVectorField& U,
124             const surfaceScalarField& phi,
125             const typeName BasicCompressibleTurbulenceModel::transportModel
126             transport,
127             const word& propertiesName = turbulenceModel::propertiesName
128         )
129         {
130             return autoPtr<BasicCompressibleTurbulenceModel>::New
131             (
132                 transport,
133                 propertiesName
134             );
135         }
136     }
137 }

```

```

102 #include "pimpleControl.H"
103 #include "barotropicCompressibilityModel.H"
104 #include "basicThermo.H"
105 #include "chemistryModel.H"
106 #include "laminarTurbulenceModel.H"
107 #include "LESModel.H"
108 #include "turbulenceModel.H"
109 #include "transportModel.H"
110 #include "twoPhaseMixture.H"
111 #include "viscosityModel.H"
112
113 namespace Foam
114 {
115     namespace Incompressible
116     {
117         typedef IncompressibleTurbulenceModel<transportModel> turbulenceModel;
118         typedef laminarTurbulenceModel<laminarModel> laminarModel;
119         typedef LESModel<turbulenceModel> LESModel;
120         template<class BasicCompressibleTurbulenceModel>
121         autoPtr<BasicCompressibleTurbulenceModel> New
122         (
123             const volVectorField& U,
124             const surfaceScalarField& phi,
125             const typeName BasicCompressibleTurbulenceModel::transportModel
126             transport,
127             const word& propertiesName = turbulenceModel::propertiesName
128         )
129         {
130             return autoPtr<BasicCompressibleTurbulenceModel>::New
131             (
132                 transport,
133                 propertiesName
134             );
135         }
136     }
137 }

```

