NIRVANA v3.8

astrophysical gas dynamics code

Udo Ziegler

DOCU
- user's guide

April 15, 2014

Leibniz-Institut für Astrophysik Potsdam

Contents

1 Basic information

ON THIS DOCUMENT

This document is intended to give a basic introduction on the usage of the NIRVANA v3.8 software which consists of the astrophysical gas dynamics simulation code NIRVANA and the data converter tool CAIVS. This document does not claim completeness nor to cover all aspects and details of the software. Also, this document must not be regarded as a numerics, physics or developers tutorial. Efficient use of NIRVANA definitely requires gaining experience with the code!

CODE DEVELOPMENT

RELEASE: Udo Ziegler
NIRVANA VERSION: 3.8
CONTACT: Udo Ziegler
Leibniz-Institut für Astrophysik Potsdam
14482 Potsdam/Germany
e-mail: uziegler@aip.de
internet: nirvana-code.aip.de

CONTRIBUTIONS: Oliver Gressel, oliver.gressel@nordita.org
(HLLD, cubic interpolation)

TERMS OF USE

1. **DISTRIBUTION**. The NIRVANA code is distributed by Udo Ziegler only for non-commercial, scientific purposes.
2. **MODIFICATIONS**. Please keep the copyright notice in the code modules intact.
3. **CO-AUTHORSHIP**. Udo Ziegler requests to be a co-author in a user's first publication using the NIRVANA code.
4. **ACKNOWLEDGEMENTS**. Please acknowledge the use of the NIRVANA code in publications by adding a statement like "This work used the NIRVANA code developed by Udo Ziegler at the Leibniz-Institut für Astrophysik Potsdam".
5. **DISCLAIMER**. The NIRVANA code comes without any warranty. The distributor does not accept responsibility for consequences of its usage.

NIRVANA-CODE-RELATED PAPERS

- U. Ziegler
  The NIRVANA code: Parallel computational MHD with adaptive mesh refinement
  Comp. Phys. Commun. **179** (2008) 227.
- U. Ziegler
  A semi-discrete central-upwind scheme for magnetohydrodynamics on orthogonal-curvilinear grids
  J. Comput. Phys. **230** (2011) 1035.
- U. Ziegler
  Block-structured adaptive mesh refinement on curvilinear-orthogonal grids
  SIAM J. Sci. Comp. **34** (2012) C102.

ABBREVIATIONS

HD:	hydrodynamics	MHD:	magnetohydrodynamics
EOS:	equation of state	AMR:	adaptive mesh refinement
IC:	initial condition	BC:	boundary condition
MPI:	message passing interface	#:	wildcard for an integer
ODE:	ordinary differential equation	RK:	Runge-Kutta
CT:	constrained transport	GS-RB:	Gauss-Seidel red-black
FV:	Finite-Volume	SOR-RB:	successive overrelaxation red-black
2D/3D:	two/three space dimensions	AU:	advanced users (needs experience/care in using)
ApJ:	Astrophysical Journal	JCP:	Journal of Computational Physics
EPS:	tiny number (1.e-99)	PDE:	partial differential equation

SOFTWARE DIRECTORY TREE

After unpacking NIRVANA3.8.tar.gz

```
gunzip NIRVANA3.8.tar.gz | tar xvf NIRVANA3.8.tar
```

the following directory tree exists:

/NIRVANA3.8:
- /bin: run scripts
- /doc: software documentation
- /caivs: CAIVS source files
  - /bin: makefiles, CAIVS executable and object files (after compiling)
  - /idl: IDL procedures
- /nirvana: NIRVANA code source files
  - /bin: makefiles, NIRVANA executable and object files (after compiling)
  - /project: NIRVANA code problem collection
    - /MHD/problem#: mhd problems
    - /VISC/problem#: viscosity problems
    - /COND/problem#: thermal conduction problems
    - /DIFF/problem#: magnetic diffusion problems
    - /APDIFF/problem#: ambipolar diffusion problems
    - /HEATLOSS/problem#: heatloss term problems
    - /GRAVITY/problem#: selfgravity problems

CODE COMPILING

In order to compile the simulation code NIRVANA change to directory /nirvana/bin and type

./makenirvana project_name

where project_name is the name of your project for which you should create a subfolder in /nirvana/project, i.e. /nirvana/project/project_name. This is the location you must put your customized parameter file **nirvana.in** and, eventually, your adapted/modified code modules. If no subfolder with name project_name exists, or project_name is simply skipped from the command line, compilation relates to original code in /nirvana. Note that makenirvana does not check for the existence of project_name!. Compiling a predefined test problem from the NIRVANA problem collection occurs in the same way. If, for example, project_name=MHD/problem2 the mhd problem defined in /nirvana/project/MHD/problem2 (Orszag-Tang vortex problem) is compiled. makenirvana runs the makefile Makefile_NIRVANA which must eventually be modified for your computer environment prior to compilation. makenirvana generates an executable named *NIRVANA3.8* located in folder /nirvana/bin.

MPI compilation requires a macro MPI to be posted to the code. This macro is specified via the option -D MPI in Makefile_NIRVANA. Corresponding lines in Makefile_NIRVANA must be enabled to allow for MPI compilations.

In order to compile the data converter tool CAIVS change to directory /caivs/bin and type

./makecaivs

It runs the makefile Makefile_CAIVS and generates an executable named *CAIVS* located in folder /caivs/bin. If SILO functionality is required the SILO library (https://wci.llnl.gov/codes/silo) must be installed on your system and corresponding lines in Makefile_CAIVS should be enabled/modified.

CODE RUNNING

There is a very simple script named *runnirvana* (*runcaivs*) located in /bin in order to run *NIRVANA* (*CAIVS*) on a single processor. *runnirvana* accepts project_name as command-line argument provoking *NIRVANA* to look for the parameter file **nirvana.in** in folder nirvana/project/project_name. E.g., to run test problem 2 in nirvana/project/MHD/problem2 type *./runnirvana* MHD/problem2. If project_name does not exist, the default parameter file located in /nirvana is used. The scripts must be modified according to your needs i.e. the aliases NIRVANA_HOME – home path of NIRVANA – and NIRVANA_RUN – I/O path for NIRVANA runs – have to be adapted.

There is no general procedure or script to run MPI jobs. Generating such scripts is up to the user completely. As an example, the script *run.leibniz* has been used by myself to do parallel simulations on the AIP compute server. By default, in a MPI simulation NIRVANA assumes I/O operations associated with any process rank# to take place on a storage device in a folder /baseDirectory/run# (#=1...number_of_MPI_threads), where baseDirectory has to be posted to *NIRVANA* via command line parameter (see script *run.leibniz* as an illustration where /baseDirectory = /work/ziegler). The user is therefore responsible for generating directories /baseDirectory/run# related to the MPI process ranks prior to job execution.

CHANGES TO PREVIOUS VERSION 3.7

- the implicit Euler/Pegasus solver for the heatloss term has been replaced by a more accurate exponential Rosenbrock method.
- the Coriolis force term is now treated within Strang splitting.
- miscellaneous changes (as usual).

KNOWN & UNSOLVED PROBLEMS

- Under certain physical conditions the HLLD_CT scheme is prone to small-scale numerical oscillations in 3D simulations, in particular, in conjuction with AMR.
- High Mach number/low plasma-β flows are prone to zero/negative pressure appearance, in particular with AMR. The dual energy formalism diminishes the problem but may not fully avoid it.
- In MPI simulations with very low load per MPI thread, mesh repartitioning can fail in case strong localized mesh refinement requires a complete data shift.
- CAIVS: Partially overlapping mesh blocks of different resolutions may produce little artifacts in 3D AMR visualizations.

2 NIRVANA simulation code

2.1 NIRVANA in brief

NIRVANA version 3.8 is a C code which numerically integrates the time-dependent equations of a multi-physics system consisting of non-relativistic compressible MHD, various dissipation processes (viscosity, magnetic diffusion, thermal conduction, ambipolar diffusion), self-gravity and a heatloss reaction term. The code works in 2D/3D Cartesian/cylindrical/spherical geometry and makes use of state-of-the-art numerical methods. NIRVANA allows for AMR to handle multi-scale problems and is MPI parallelized. The available physics functionality in combination with geometry, AMR and MPI is summarized in the following table:

Code functionality

	CARTESIAN				CYLINDRICAL/SPHERICAL			
	unigrid		AMR		unigrid		AMR**	
	serial	MPI	serial	MPI	serial	MPI*	serial	MPI
ideal MHD	+	+	+	+	+	+	+	+
viscosity	+	+	+	+	+	+	+	+
magnetic diffusion	+	+	+	+	+	+	+	+
thermal conduction	+	+	+	+	+	+	+	+
ambipolar diffusion	+	+	+	+	•	•	•	•
selfgravity	+	+	+	+	−	−	−	−
heatloss	+	+	+	+	+	+	+	+
tracer	+	+	+	+	+	+	+	+

∗ domain decomposition in $\phi(\theta)$-direction for 'F'-type BC with geometric axis unsupported.
∗∗ AMR for 'F'-type BC unsupported.
+ available
− not available
• implemented but not validated

2.1.1 Equations

NIRVANA numerically integrates (in full scale) the following set of PDEs:

$$
\begin{aligned}
\partial_t \varrho + \nabla\cdot(\varrho\mathbf{v}) &= 0\,, \\
\partial_t e + \nabla\cdot\left[(e+p_{\rm tot})\,\mathbf{v} - \frac{1}{\mu}(\mathbf{v}\cdot\mathbf{B})\mathbf{B}\right] &= \nabla\cdot\left[\mathbf{v}\tau + \frac{\eta}{\mu}\mathbf{B}\times(\nabla\times\mathbf{B}) - \frac{1}{\mu}\mathbf{B}\times\mathbf{E}_{\rm AD} - \mathbf{F}_{\rm C}\right], \\
&\quad -\varrho\nabla\Phi\cdot\mathbf{v} + \varrho\mathbf{f}_e\cdot\mathbf{v} + \mathbf{f}_{cc}\cdot\mathbf{v} + \mathcal{L}(T,\varrho) \\
\partial_t(\varrho\mathbf{v}) + \nabla\cdot\left[\varrho\mathbf{v}\mathbf{v} + p_{\rm tot}I - \frac{1}{\mu}\mathbf{B}\mathbf{B}\right] &= -\varrho\nabla\Phi + \nabla\cdot\tau + \varrho\mathbf{f}_e + \mathbf{f}_{cc}\,, \\
\partial_t\mathbf{B} - \nabla\times(\mathbf{v}\times\mathbf{B} - \eta\nabla\times\mathbf{B} + \mathbf{E}_{\rm AD}) &= 0\,, \\
\nabla^2\Phi &= 4\pi G\varrho \\
\partial_t C_\alpha + \mathbf{v}\cdot\nabla C_\alpha &= 0
\end{aligned}
$$

with

EOS: $p = p(\varrho, e)$

divergence constraint: $\nabla\cdot\mathbf{B} = 0$

total pressure: $p_{\mathrm{tot}} = p + \frac{1}{2\mu}|\mathbf{B}|^2$

stress tensor: $\tau = \nu\left(\nabla\mathbf{v} + (\nabla\mathbf{v})^\top - \frac{2}{3}(\nabla\cdot\mathbf{v})I\right)$; $tr(\tau) = 0$ (traceless); $\tau = \tau^\top$ (symmetric)

Coriolis- and centrifugal force (rotating frame): $\mathbf{f}_{cc} = -2\varrho\boldsymbol{\Omega}_0\times\mathbf{v} - \varrho\boldsymbol{\Omega}_0\times(\boldsymbol{\Omega}_0\times\mathbf{x})$

basic variables:
ϱ: mass density [$\mathrm{kg\,m^{-3}}$]
$\mathbf{v}$: fluid velocity [$\mathrm{m\,s^{-1}}$]
e: total (without gravity) energy density [$\mathrm{J\,m^{-3}}$]
$\mathbf{B}$: magnetic field [T]
Φ: gravitational potential [$\mathrm{m^2\,s^{-2}}$]
C_α: set of tracer variables [1]

other quantities:
T: temperature [K]
p: thermal pressure [Pa]
p_{tot}: total pressure [Pa]
ε: thermal energy density [$\mathrm{J\,m^{-3}}$]
u: specific thermal energy [$\mathrm{J\,kg^{-1}}$]
$\mathbf{F}_{\mathrm{C}}$: conductive heat flux [$\mathrm{J\,m^{-2}\,s^{-1}}$]
$\mathbf{E}_{\mathrm{AD}}$: ambipolar diffusion field [$\mathrm{T\,m\,s^{-1}}$]
$\mathbf{f}_e$: external specific force [$\mathrm{N\,kg^{-1}}$]
$\mathcal{L}$: heatloss term [$\mathrm{J\,m^{-3}\,s^{-1}}$]
$\mathbf{f}_{cc}$: non-inertial forces (rotating frame) [$\mathrm{N\,m^{-3}}$]
$\mathbf{x}$: Cartesian position vector [m]
γ: ratio of specific heats [1]
Γ: polytropic exponent [1]
K: polytropic constant [$\mathrm{Pa\,(kg\,m^{-3})^{-\Gamma}}$]
μ: magnetic permeability coefficient [$\mathrm{V\,s\,A^{-1}\,m^{-1}}$]
$\overline{\mu}$: mean molecular weight [1]
$\overline{Y}$: mean ionisation degree [1]
τ: stress tensor [$\mathrm{N\,m^{-2}}$]
ν: dynamic viscosity coefficient [$\mathrm{kg\,m^{-1}\,s^{-1}}$]
η: magnetic diffusion coefficient [$\mathrm{m^2\,s^{-1}}$]
$\kappa, \kappa_{\mathrm{Sp}}, \kappa_{||}, \kappa_\perp$: thermal conductivity coefficients [$\mathrm{J\,K^{-1}\,m^{-1}\,s^{-1}}$]
T_{iso}: temperature in isothermal EOS [K]
c_s: sound speed [$\mathrm{m\,s^{-1}}$]
$\boldsymbol{\Omega}_0$: rotating frame angular velocity [$\mathrm{s^{-1}}$]
$m_u = 1.66057\cdot 10^{-27}\mathrm{kg}$: atomic mass unit
$k = 1.3806\cdot 10^{-23}\mathrm{J\,K^{-1}}$: Boltzmann constant
$G = 6.673\cdot 10^{-11}\mathrm{m^3\,kg^{-1}\,s^{-2}}$: gravitational constant

Thermodynamic EOS. NIRVANA can handle different EOS:

- adiabatic EOS with index γ.
- polytropic EOS with polytropic constant K and polytropic exponent Γ.
- isothermal EOS with constant temperature T_{iso}.

- user-defined analytic/tabulated EOS.

The pressure and temperature as a function of density ϱ and thermal energy density ε (or specific thermal energy $u = \varepsilon/\varrho$) are given by

$$p = \begin{cases} (\gamma-1)\varepsilon & \text{adiabatic} \\ K\varrho^\Gamma & \text{polytropic} \\ \varrho k T_{iso}(1+\overline{Y})/(\overline{\mu} m_u) & \text{isothermal} \\ p(\varrho,u) & \text{user-defined (analytic/tabulated)} \end{cases}$$

$$T = \begin{cases} (\gamma-1)\frac{m_u}{k}\frac{\overline{\mu}}{1+\overline{Y}}\frac{\varepsilon}{\varrho} & \text{adiabatic} \\ K\varrho^{\Gamma-1}\frac{m_u}{k}\frac{\overline{\mu}}{1+\overline{Y}} & \text{polytropic} \\ T_{iso} & \text{isothermal} \\ T(\varrho,u) & \text{user-defined (analytic/tabulated)} \end{cases}$$

where ε is obtained from the total energy density e according to $\varepsilon = e - \varrho\mathbf{v}^2/2 - \mathbf{B}^2/2\mu$. In the cases where p and T are a function of ϱ only, the energy equation is redundant and therefore skipped from the PDEs.

Heat conduction. NIRVANA can handle the classical form of heat conduction as well as allows for saturation effects:

- classical form of heat conduction with the anisotropic (magnetic-field-dependent) heat flux given by

$$\mathbf{F}_\mathrm{C} = -\kappa_{||}(\nabla T\cdot\hat{\mathbf{B}})\hat{\mathbf{B}} - \kappa_\perp\left(\nabla T - (\nabla T\cdot\hat{\mathbf{B}})\hat{\mathbf{B}}\right)$$

  where $\hat{\mathbf{B}} = \mathbf{B}/|\mathbf{B}|$ is the unit vector in the direction of magnetic field. The parallel and perpendicular conductivity coefficients, $\kappa_{||}$ and $\kappa_\perp$, are assumed either constant values, user-defined functions or are given by Spitzer theory (book: L. Spitzer; Physics of fully ionized gases; 1962). In Spitzer theory – valid for highly ionized gases where electrons are the main contributor to heat conduction –

$$\begin{aligned} \kappa_{||_\mathrm{Sp}} &= \frac{1.84\cdot 10^{-10}}{Z\ln\Lambda}T^{5/2} \\ \kappa_{\perp_\mathrm{Sp}} &= 8.04\cdot 10^{-33}\overline{\mu}^{-3/2}Z^3\left(\frac{\ln\Lambda}{m_u}\right)^2\frac{\varrho^2}{T^3B^2}\kappa_\mathrm{Sp} \end{aligned}$$

  measured in SI units with $\ln\Lambda$ (preset to 30) the Coloumb logarithm and Z (preset to 1) the mean ionic charge number. Validity of the perpendicular coefficient is restricted to the strong magnetic field case. In the weak field case theory breaks down and heat conduction approches isotropy. In the implementation this is accounted for by modification of $\kappa_{\perp_\mathrm{Sp}}$ according to $\kappa_{\perp_\mathrm{Sp}} = \min\{\kappa_{\perp_\mathrm{Sp}}, \kappa_{||_\mathrm{Sp}}\}$ which avoids singular behaviour when $\mathbf{B}\longrightarrow 0$. If $\kappa_{||} = \kappa_\perp$ the isotropic form of heat conduction is therefore recovered. In the absence of magnetic fields heat conduction is generically isotropic

$$\mathbf{F}_\mathrm{C} = -\kappa\nabla T$$

  where κ is defined in the code by prescribing $\kappa_{||}$ only.

- saturated heat flux according to Cowie & McKee (ApJ 211,135; 1977) given by

$$|\mathbf{F}_{\mathrm{C,sat}}| = 5\Psi\varrho c_s^3$$

  where c_s is the sound speed and $\Psi = \mathcal{O}(1)$. The actual heat flux is limited to this canonical value if temperature gradients become so large that the classical diffusion approximation breaks

down. Expressed in terms of a conductivity saturation effects can be included by defining modified coefficients

$$\tilde{\kappa} = \left(\frac{1}{\kappa} + \frac{1}{\kappa_{\rm sat}} \right)^{-1}$$

where

$$\kappa_{\rm sat} = \begin{cases} \frac{|\mathbf{F}_{\rm C,sat}|}{|\nabla T|} & \text{isotropic case} \\ \frac{|\mathbf{F}_{\rm C,sat}|}{|(\nabla T \cdot \hat{\mathbf{B}})|} & \text{anisotropic case, parallel coefficient} \\ \frac{|\mathbf{F}_{\rm C,sat}|}{|\nabla T - (\nabla T \cdot \hat{\mathbf{B}})\hat{\mathbf{B}}|} & \text{anisotropic case, perpendicular coefficient} \end{cases}$$

Ambipolar diffusion. NIRVANA can handle ambipolar diffusion in weakly ionized gases within the single fluid approach assuming the strong coupling approximation. Here, the ambipolar diffusion field $\mathbf{E}_{\rm AD}$ appearing in the induction equation and energy equation is given by

$$\mathbf{E}_{\rm AD} = \frac{1}{\mu} \eta_{\rm AD} \left[(\nabla \times \mathbf{B}) \times \mathbf{B} \right] \times \mathbf{B}$$

where $\eta_{\rm AD}$ is the ambipolar diffusion coefficient. A model for $\eta_{\rm AD}$ which usually depends on the ionization degree and coupling parameter must be defined by the user.

Dual energy formalism. NIRVANA implements a dual energy formalism to moderate the problems arising with high Mach number/low plasma-β flows. In the applied dual energy approach the total energy equation is solved simultaneously with the thermal energy equation given by

$$\partial_t \varepsilon + \nabla \cdot (\varepsilon \mathbf{v}) + p \nabla \cdot \mathbf{v} = \frac{1}{2\nu} tr(\tau^2) + \frac{\eta}{\mu} |\nabla \times \mathbf{B}|^2 + \frac{\eta_{\rm AD}}{\mu^2} |\mathbf{B} \times (\nabla \times \mathbf{B})|^2 - \nabla \cdot \mathbf{F}_{\rm C} \,.$$

The actual thermal energy density is then reset

$$\varepsilon \longrightarrow \max\{\varepsilon, e - \varrho \mathbf{v}^2/2 - \mathbf{B}^2/2\mu\} \quad \text{if} \quad (e - \varrho \mathbf{v}^2/2 - \mathbf{B}^2/2\mu)/e < \text{DUAL\_ENERGY\_SW}$$

where DUAL_ENERGY_SW is a user parameter (code variable _C.energy_dual_sw) which controls the transition.

2.1.2 Numerical methods

Hyperbolic part of equations. The hyperbolic part of the equations (MHD) is solved with an unsplit FV method within a methods-of-lines integration framework. After space discretization the resulting system of ODEs is solved with a time-explicit RK method. The standard second-order RK scheme (RK2) or the more stable third-order RK version (RK3) of Shu & Osher (JCP 77, 439 (1988)) is available. In order to compute fluxes and the electric field two different Godunov-type solvers combined with a CT method for a divergence-free evolution of the magnetic field are implemented:

- **CU_CCT.** Second-order version of the Central-Upwind scheme of Kurganov etal., SIAM J. Sci. Comput. 23 (2001) 707, applied to the Euler equations with Lorentz-force term combined with a CT scheme for the induction equation. The electric field is computed from a genuinely 2D central-upwind procedure (CCT) based on the evolution-projection method by Bryson & Levy, JCP 189 (2003) 63. For a comprehensive description of the CU_CCT scheme see Ziegler, JCP 230 (2011) 1035.
- **HLLD_CCT.** HLLD approximate Riemann solver of Miyoshi & Kusano, JCP 208 (2005) 315, applied dimenson-by-dimension in 2D/3D to the Euler equations with Lorentz-force term combined with a CT scheme for the induction equation. The electric field is computed like in the CU_CCT scheme.

- **HLLD_CT.** HLLD approximate Riemann solver of Miyoshi & Kusano, JCP 208 (2005) 315, applied dimenson-by-dimension in 2D/3D to the MHD equations combined with a CT scheme for the induction equation. The electric field is computed by face-to-edge interpolation from the HLLD electric field fluxes using the entropy-wave-sensitive upwind correction of Gardiner & Stone, JCP 227 (2008) 4123.

NOTE:

- In case of HD the HLLD Riemann solver reduces to the HLLC Riemann solver.
- The HLLD_CT scheme does not work in conjunction with 'F'-type BCs.

Parabolic part of equations. The dissipation terms are spatially discretized within the FV framework and make use of second-order finite-difference approximations of the dissipative fluxes. The magnetic diffusion solver and ambipolar diffusion solver retain the divergence-free condition for $\mathbf{B}$. Two different time integrators are available:

- **STD.** Classical explicit time integration based on the RK2/RK3 schemes as used for the integration of the hyperbolic part of equations. Coupling to the hyperbolic part is unsplit.
- **RKL.** Stabilized Runge-Kutta-Legendre integrator according to Meyer et al., MNRAS 422 (2012) 2102. This scheme is specially designed for midly-stiff parabolic equations allowing effective Courant numbers (much) higher than with STD (i.e. Super-TimeStepping). Coupling to the hyperbolic part is via Strang-splitting.

Poisson equation. The Poisson equation is solved – in case of AMR – with a conservative multigrid method (V-cycle iteration, GS-RB smoother, 2nd-order restriction/prolongation operators) employing elliptic matching at grid interfaces, – in case of a uniform mesh – with a two-grid method using a SOR-RB coarse-level solver.

NOTE:

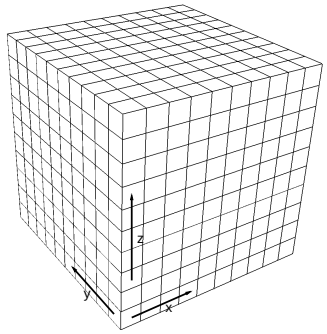
- The Poisson solver only works in 3D Cartesian geometry.

Heatloss term. Time integration of the heatloss term $\mathcal{L}(T, \varrho)$ in the energy equation is done with an exponential Rosenbrock method of order 3 with embedded error estimator. Coupling to the hyperbolic part is via Strang-splitting.

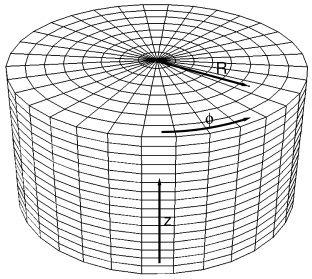
2.1.3 Geometries

The equations can be solved in Cartesian, cylindrical and spherical coordinates. A general nomenclature is used to describe metric space. Code coordinates `(x,y,z)`, spatial increments $(\delta x, \delta y, \delta z)$ and metric scale factors (h_x, h_y, h_z) of the space metric $\mathcal{H} = diag(h_x^2, h_y^2, h_z^2)$ have the following meaning depending on the selected geometry:

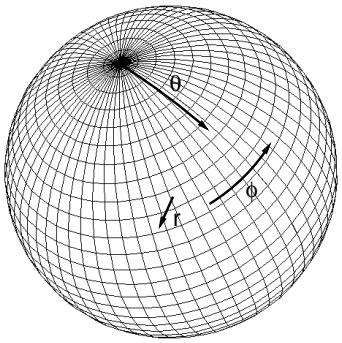
Cartesian geometry: `(x,y,z)` $\longrightarrow (x, y, z)$
$(\delta x, \delta y, \delta z) = (\delta x, \delta y, \delta z)$
$(h_x, h_y, h_z) = (1, 1, 1)$



cylindrical geometry: `(x,y,z)` $\longrightarrow (z, R, \phi)$
$(\delta x, \delta y, \delta z) = (\delta z, \delta R, \delta\phi)$
$(h_x, h_y, h_z) = (1, 1, R)$



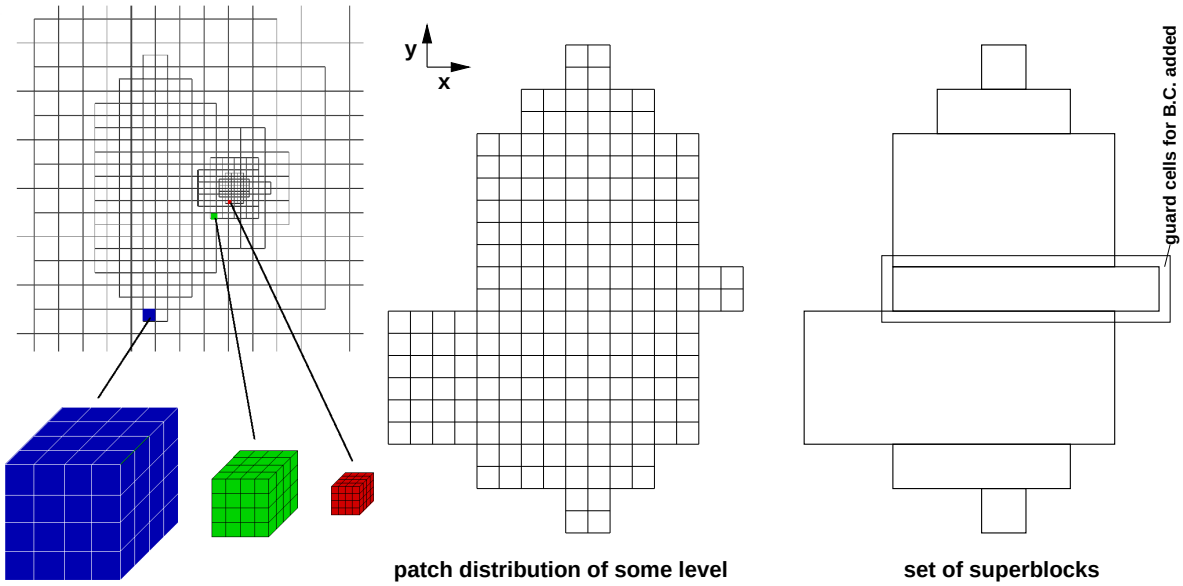
spherical geometry: `(x,y,z)` $\longrightarrow (r, \theta, \phi)$
$(\delta x, \delta y, \delta z) = (\delta r, \delta\theta, \delta\phi)$
$(h_x, h_y, h_z = h_y \cdot h_{zy}) = (1, r, r \cdot \sin\theta)$



2.1.4 Grid structure and AMR

Mesh refinements on a given base grid are realized by hierarchically nested blocks of size 4^3 cells (in 3D; 4^2 cells in 2D) with half grid spacing in each direction compared to their parent blocks. All blocks with the same grid spacing build a refinement level. The base level itself is comprised out of such generic blocks and the full grid hierarchy is structured like an oct-tree (left panel in the figure below), however, with incomplete refined blocks allowed in order to improve mesh adaptivity. For good computational efficiency the solver operates not on generic blocks but on superblocks generated by a clustering algorithm with the clustering of blocks separately done for each refinement level $\ell = 0...\ell_{\max}$ (right panels in the figure below). Generic blocks do not contain ghost zones for BCs. Those are automatically added in the construction of superblocks.

Mesh refinement may be controlled by different types of refinement criteria:



- gradient-based/second-derivatives-based criterion given by

$$\left[\alpha\frac{|\delta U|}{|U|+U_{\rm ref}}+(1-\alpha)\frac{|\delta^2 U|}{|\delta U|+{\rm FILTER}\cdot(|U|+U_{\rm ref})}\right]\left(\frac{\delta x^{(\ell)}}{\delta x^{(0)}}\right)^{\xi}\left\{\begin{array}{lll}>\mathcal{E}_U & \exists U & \text{refinement}\\ <0.8\mathcal{E}_U & \forall U & \text{derefinement}\end{array}\right.$$

  where U is a set of primary variables the criterion is applied to. Undivided first (δU) and second ($\delta^2 U$) differences of U are computed and both contributing terms are averaged over directions. $\alpha\in[0,1]$ is a switch between a purely gradient-based criterion ($\alpha=1$) and second-derivatives-based criterion ($\alpha=0$), $U_{\rm ref}$ are reference values, $\mathcal{E}_U$ are thresholds and ξ is a parameter giving some control on level dependence. FILTER (preset to 10^{-2}) is a filter to suppress refinement at small-scale wiggles. The criterion is checked on a generic block octant-wise including a 2-cell wide buffer zone around the octant.

- Jeans-length-based criterion in selfgravitating gas simulations given by

$$\left(\frac{\pi}{G}\frac{c_s^2}{\varrho}\right)^{1/2}\cdot\mathcal{E}_{\rm Jeans}\left\{\begin{array}{ll}<\delta s & \text{refinement}\\ >1.25\delta s & \text{derefinement}\end{array}\right.$$

  where $\mathcal{E}_{\rm Jeans}$ is a user-specific threshold and $\delta s=min\{\delta x,h_y\delta y,h_z\delta z\}$.

- Field-length-based criterion to track thermally unstable gas in simulations with combined heatloss- and conduction term given by

$$2\pi\left(\frac{\kappa T}{\max\left\{T\left(\frac{\partial\mathcal{L}}{\partial T}\right)_{\varrho}-\varrho\left(\frac{\partial\mathcal{L}}{\partial\varrho}\right)_{T},EPS\right\}}\right)^{1/2}\cdot\mathcal{E}_{\rm Field}\left\{\begin{array}{ll}<\delta s & \text{refinement}\\ >1.25\delta s & \text{derefinement}\end{array}\right.$$

  where $\mathcal{E}_{\rm Field}$ is a user-specific threshold.

2.1.5 Parallel computing

Code parallelization is based on the MPI library. The code infrastructure provides routines for mesh (re)partitioning and data transfer among processors. Mesh repartitioning in the AMR case is based on space-filling curve techniques enabling load balancing with an optimized intra-level data locality. Data

locality across refinement levels is realized by block-sharing between neighboring partitions introducing an (irreducible) overlap region at partition borders. A simple block domain decomposition is also possible for uniform grids. The data transfer model for mesh syncronization processes and mesh repartitioning is heavily coarse-grained collecting all data in a first step and then communicating them in a second step in two sweeps: meta data and physical data.

2.2 Usage

2.2.1 User-controllable modules

The following modules allow the user to control a simulation:

nirvana.in – specification of main parameters
User.h – user-controllable macros
configUser.c – definition of ICs
bcrhoUser.c, bcmUser.c, bceUser.c, bcbUser.c, ... – user-defined BCs
viscosityCoeffUser.c – implementation of a dynamic viscosity coefficient
diffusionCoeffUser.c – implementation of a magnetic diffusion coefficient
conductionCoeffUser.c – implementation of a conductivity coefficient
APdiffusionCoeffUser.c – implementation of a ambipolar diffusion coefficient
eosUser.c – implementation of an analytic EOS (AU)
forceUser.c – definition of an external force
createTabUser.c – specification of sample data for look-up tables/tabulated EOS (AU)
checkDomainUser.c – construction of user-defined regions of mesh refinement (AU)
analysisDataUser.c – data analysis during run time
modifyFluxUser.c – flux modification (AU)
modifyConfigUser.c – data modification in restarts (AU)
sourceHeatingUser.c – implementation of a heating function
sourceCoolingUser.c – implementation of a cooling function

2.2.2 Specification of main parameters

The file **nirvana.in** serves as user interface for the specification of main code parameters. Parameters are changed by editing this file. **nirvana.in** is devided into different parameter sections:

SIMULATION I/O
GEOMETRY
DOMAIN SETTINGS
BOUNDARY CONDITIONS
MESH REFINEMENT
USER-SPECIFIC PARAMETERS
SOLVER SPECIFICATIONS
PHYSICS SPECIFICATIONS
SPECIES

Except the first two parameters `mode` and `fname` in section SIMULATION I/O parameters described in **nirvana.in** are globally defined in the code. These parameters (among others) are summarized in a variable named `_C` of struct type `CTL` declared in **nirvana.h**. Any parameter `X` in `CTL` can be accessed by `_C.X` everywhere. In the following each parameter section of **nirvana.in** is explained in detail. The description is preceded an example section. The parameter name `X` and its C type is given in brackets. For some parameters the allowed values are stated as $\in\{.\}$ for a set or $\in[.]$ for a range. Text in **nirvana.in** which follows a $'>'$ sign in a line is interpreted as comment during parsing of the file.

```
>SIMULATION I/O ----------------------------------------------------------------
 NEW   ./start/mod100                        >mode:{NEW,CONT,REST,MODI,ANAL},fname
100000  0.20e+00                             >mod_max,time_max
 0010  0500  01000  91005  99999  1.00e+03  >freq_{log,raw,mod,ana,hdf},walltime_tmp
```

parameter 1 (`char* mode`): $\in$\{NEW,CONT,REST,MODI,ANAL\}.
Simulation running mode. NEW starts a new simulation using ICs provided by the user in **configUser.c**. CONT continues a simulation taking model data from the file with filename stored in the CONTINUE

file. The CONTINUE file is generated and updated during a simulation and contains the filename of the latest available full-precision snapshot. The update frequency of CONTINUE is given by the parameters `_C.freq_mod` and `_C.walltime_tmp` (see below). REST restarts a simulation taking model data from the file specified by the parameter `fname` (see next). MODI is as REST but, in addition, the function **modifyConfigUser** is called after data read-in and allows its modification by the user. ANAL renders a post-analysis of (DATA_MOD-type) simulation data from the file specified by `fname`.

parameter 2 (`char* fname`):
Name (including path relative to the NIRVANA I/O directory specified in the job script *runnirvana*) of the file containing model data to (re)start a simulation in case `mode`∈{REST,MODI} or for post-analysis purposes in case `mode`=ANAL. In case of MPI simulations `fname` must be without any threadnumber postfix (e.g. instead of mod#.# `fname` should be mod#; read also 2.3). Note: in a restart simulation all the following parameters in **nirvana.in** are ignored but those are recovered from stored model data! If input parameters need to be changed use mode=`MODI` instead of `REST` and overwrite them in **modifyConfigUser**.

line 2, parameter 1 (`int _C.mod_max`):
Maximum number of time-steps a simulation runs. However, if physical evolution time reaches `_C.time_max` (see next) the simulation stops before `_C.mod_max` time-steps are done.

line 2, parameter 2 (`double _C.time_max`):
Maximum physical evolution time a simulation runs. However, if `_C.mod_max` time-steps are done the simulation stops before evolution time `_C.time_max` is reached.

line 3, parameters 1-5 (`int _C.freq_log,_C.freq_raw,_C.freq_mod,_C.freq_ana,_C.freq_hdf`):
Various frequencies of data output in units of time-steps e.g. data of type DATA_RAW is stored in **raw#** files every `_C.freq_raw` time-step. For the different data types of NIRVANA see 2.3. The parameter `_C.freq_ana` denotes the frequency the function **analysisDataUser** is called for data analysis during run time.

line 3, parameter 6 (`double _C.walltime_tmp`):
Wall-time interval in seconds for DATA_MOD data output in the **tmp#** file.

```
>GEOMETRY ------------------------------------------------------------------------
 CART  0.00e+00  0.00e+00  0.00e+00      >geometry,omega[0-2]
```

parameter 1 (`flag_t _C.geometry`): ∈{CART,CYL,SPH}.
Choice of coordinate system.

parameters 2-4 (`double _C.omega[0]...C.omega[2]`):
Angular velocity of the rotating frame of reference with respect to the inertial frame. If zero, equations are solved in the inertial frame of reference. If non-zero, equations are solved in the corotating frame. In Cartesian geometry `_C.omega[0-2]` have the meaning of usual vector components for $\mathbf{\Omega}_0$. In cylindrical/spherical geometry only the first component `_C.omega[0]` has a meaning and gives the angular velocity of rotation axis which is assumed to coincide with the geometric axis ($\mathbf{\Omega}_0 \parallel \mathbf{e}_z$)!

```
>DOMAIN SETTINGS -----------------------------------------------------------------
  0.000000e+00  3.200000e+00  0096       >lo[0],up[0],dim[0]
  0.000000e+00  1.000000e+00  0030       >lo[1],up[1],dim[1]
  0.000000e+00  1.000000e+00  0000       >lo[2],up[2],dim[2]
BLOCK 2 2 4                              >ptype:{SPACE_FILLING_CURVE,BLOCK # # #}
```

line 1, parameters 1-2 (`double _C.lo[0],_C.up[0]`):
Lower/upper x-coordinate of the computational domain.

line 1, parameter 3 (`int _C.dim[0]`):
Number of base level grid points in x-direction. `_C.dim[0]` must be a multiple of 4 (recall that the base level is composed out of generic grid blocks having 4 cells per coordinate direction)! Ghost cells needed by the solver are not counted here and are added automatically!

line 2 (`double _C.lo[1],_C.up[1]; int _C.dim[1]`):
Same as line 1 but for the y-direction. In case of spherical geometry `_C.lo[1]` and `_C.up[1]` define the range of angle θ measured from the geometric northpole in units of π.

line 3 (`double _C.lo[2],_C.up[2]; int _C.dim[2]`):
Same as line 1 but for the z-direction. In case of cylindrical/spherical geometry `_C.lo[2]` and `_C.up[2]` define the range of azimuth ϕ measured in units of π. If `_C.dim[2]=0`, the simulation is assumed 2D respective axissymetric in case of cylindrical/spherical geometry!

line 4 (`char* _C.partitioning_type; int _C.bnx,_C.bny,_C.bnz`):
Domain decomposition type in parallel simulations. If `_C.partitioning_type=SPACE_FILLING_CURVE` space-filling curve techniques are used for mesh (re)partitioning (the only possible for AMR). If `_C.partitioning_type=BLOCK` the domain is decomposed into `_C.bnx`×`_C.bny`×`_C.bnz` equally-sized subdomains where `_C.bnx (_C.bny,_C.bnz)` is the number of those subdomains in $x(y,z)$-direction. The block domain decomposition must be chosen consistently with the number of processors and base level dimensions, i.e. a subdomain's size must match an integral number of base level generic grid blocks!

```
>BOUNDARY CONDITIONS ------------------------------------------------------------
UUUUPP                                        >bc[0-5]:{I,O,A,M,R,C,P,D,F,U}
```

parameters 1-6 (`char _C.bc[0]...C.bc[5]`): ∈{I,O,A,M,R,C,P,D,F,U}.
Definition of (MHD) boundary conditions for the LOWER-X, UPPER-X, LOWER-Y, UPPER-Y, LOWER-Z, UPPER-Z domain boundary specified by a capital each:

- I: *Inflow* - the fluid is allowed to flow into the domain but not to flow out; vanishing gradient of scalars (total energy density set such that gradient in thermal energy density vanishes); vanishing gradient of parallel components plus divergence-free extrapolation of the magnetic field.
- O: *Outflow* - the fluid is allowed to flow out of the domain but not to flow in; vanishing gradient of scalars (total energy density set such that gradient in thermal energy density vanishes); vanishing gradient of parallel components plus divergence-free extrapolation of the magnetic field.
- M: *mirror symmetry* - reflecting conditions in all variables.
- A: *antimirror symmetry* - same as M for HD variables; the magnetic field is forced to have dipole parity with respect to the domain boundary.
- R: *reflection-on-axis* - reflecting conditions in all variables at the geometric axis in cylindrical/spherical coordinates. Meaningless in Cartesian coordinates!
- C: *reflection-at-center* - reflecting conditions in all variables at the coordinate origin in spherical coordinates. Meaningless in Cartesian/cylindrical coordinates!
- P: *periodicity*
- D: *default* - vanishing gradient in HD variables and normal-to-boundary condition (pseudo-vacuum) for the magnetic field.
- F: *full geometry* - for cylindrical/spherical geometry only. uses boundary conditions natural for the geometric axis at LOWER-Y ($y = 0$) in cylindrical geometry and at LOWER-X ($x = 0$), LOWER-Y ($y = 0$) and UPPER-Y ($y = \pi$) in spherical geometry. If the geometric axis is in fact excluded from the domain, boundary conditions are of pseudo-axis type.

U: *user-defined* - boundary conditions specified by the user in **bcrhoUser.c**, **bceUser.c**, **bcmUser.c**, **bcbUser.c**, **bctrUser.c** etc.

```
>MESH REFINEMENT ------------------------------------------------------------
  0  0                                          >smr,amr
  2.0e-01  4.0e-01  3.0e-01 -2.0e-01  2.0e-01  >amr_eps[0-4] / refinement thresholds
  0.0e-00  1.0e-01  0.0e+00  1.0e-01  1.0e-02  >rhoref,vref,eref,Bref,Cref
  0.6e+00  0.0e+00                             >amr_{d1,exp}
  0.0e+00  0.0e+00  0.0e+00                    >amr_{Jeans,dJeans,Field}
```

line 1, parameter 1 (`flag_t _C.smr`):
|`_C.smr`| denotes the maximum grid refinement level in user-defined refinement regions (see 2.2.10). If `_C.smr<0` regions are *initially* refined and are later subject to the usual refinement criteria. If `_C.smr>0`, on the other hand, regions are *permanently* refined and are not subject to derefinement in the course of simulation.

line 1, parameter 2 (`flag_t _C.amr`):
Requested maximum grid refinement level in AMR simulations. `_C.amr` is limited by the value of the macro MAXLEVEL in **User.h**.

line 2, parameters 1-5 (`double _C.amr_eps[0]...._C.amr_eps[4]`):
Thresholds for the mass density (`_C.amr_eps[0]`), momentum densities (`_C.amr_eps[1]`), energy density (`_C.amr_eps[2]`), magnetic field (`_C.amr_eps[3]`) and tracer (`_C.amr_eps[4]`) in the derivatives-based mesh refinement criterion. If `_C.amr_eps[#]<=0` the corresponding criterion is disabled. The typical range is $[0.1, 0.5]$.

line 3, parameter 1 (`double _C.rhoref`):
Density reference value in the mesh refinement criterion. Can be savely set to zero.

line 3, parameter 2 (`double _C.vref`):
Velocity reference value in the mesh refinement criterion. Typically, this value should be chosen a small fraction of a characteristic velocity of the problem.

line 3, parameter 3 (`double _C.eref`):
Energy density reference value in the mesh refinement criterion. Can be savely set to zero.

line 3, parameter 4 (`double _C.bref`):
Magnetic field reference value in the mesh refinement criterion. Typically, this value should be chosen a small fraction of a characteristic magnetic field strength of the problem.

line 3, parameter 5 (`double _C.Cref`):
Tracer reference value in the mesh refinement criterion.

line 4, parameter 1 (`double _C.amr_d1`): $\in[0,1]$.
Controls the transition between a pure gradient-based (`_C.amr_d1=1`) and pure second-derivatives-based (`_C.amr_d1=0`) check of the derivatives-based refinement criterion (the quantity α there).

line 4, parameter 2 (`double _C.amr_exp`):
Allows for additional control in the derivatives-based mesh refinement criterion (the exponent ξ there) in the sense that refinement becomes more difficult (easier) with increasing refinement level if `_C.amr_exp>0` (`_C.amr_exp<0`). A typical range may be $[-1, 1]$.

line 5, parameter 1 (`double _C.amr_Jeans`):
Jeans-based refinement criterion in selfgravitating flow simulations. `_C.amr_Jeans` (typically <= 0.2) defines the threshold in the ratio of local grid spacing to local Jeans length above which mesh refinement is triggered. If `_C.amr_Jeans<0` the Jeans criterion is disabled. If the maximum level `_C.amr` is already reached no further refinement takes place regardless whether the Jeans length is resolved or not!

line 5, parameter 2 (`double _C.amr_dJeans`):

Allows for a variation of the Jeans threshold with refinement level according to _C.amr_Jeans $\longrightarrow$ _C.amr_Jeans-level*_C.amr_dJeans (i.e. higher resolved Jeans length on higher refinement levels if _C.amr_dJeans>0).

line 5, parameter 3 (double _C.amr_Field):
Field-length-based refinement criterion in simulations with both cooling/heating and heat conduction. _C.amr_Field (typically <= 0.2) defines the threshold in the ratio of local grid spacing to local Field length above which mesh refinement is triggered. If _C.amr_Field<0 the Field criterion is disabled. If the maximum level _C.amr is already reached no further refinement takes place regardless whether the Field length is resolved or not!

```
>USER-SPECIFIC PARAMETERS ------------------------------------------------------
 0.00000e+00  0.00000e+00  0.00000e+00   >param[0-2] /
 0.00000e+00  0.00000e+00  0.00000e+00   >param[3-5] /
 0.00000e+00  0.00000e+00  0.00000e+00   >param[6-8] /
 0.00000e+00  0.00000e+00  0.00000e+00   >param[9-11] /
 0.00000e+00  0.00000e+00  0.00000e+00   >param[12-14] /
  0  0  0  0  0  0  0  0                 >flag[0-7] /
```

lines 1-5 (double _C.param[0]...._C.param[14]):
Free user parameter of double type.

line 6 (int _C.flag[0]..._C.flag[7]):
Free user parameter of integer type.

```
>SOLVER SPECIFICATIONS ---------------------------------------------------------
 RK3 CU CCT 4.0e-01                    >mhd_solver_{time,flux,ef},mhd_courant
 STD 4.0e-01                           >viscosity_{solver,courant}
 STD 4.0e-01                           >diffusion_{solver,courant}
 STD 4.0e-01                           >conduction_{solver,courant}
 STD 4.0e-01                           >APdiffusion_{solver,courant}
 1.0e+00                               >heatloss_courant
 1.0e-06                               >gravity_tol
 000                                   >delay_step
```

line 1, parameter 1 (flag_t _C.mhd_solver_time): $\in$\{RK2,RK3\}.
Time integrator for hyperbolic MHD solver. Choices available are RK2 for the standard second-order accurate Runge-Kutta method and RK3 for a third-order accurate, strong-stability-preserving Runge-Kutta integrator.

line 1, parameter 2 (flag_t _C.mhd_solver_flux): $\in$\{CU,HLLD\}.
Method of flux computation in MHD solver. Choices available are CU for a central-upwind scheme and HLLD for a Riemann solver.

line 1, parameter 3 (flag_t _C.mhd_solver_ef): $\in$\{CT,CCT\}.
Method of electric field computation in the constrained-transport MHD solver. Choices available are CT for a flux-to-edge interpolation of dimension-by-dimension upwinded base solver electric field fluxes and CCT for a central-type, genuinely 2D upwinding procedure of the electric field.

line 1, parameter 4 (double _C.mhd_courant):
CFL number for MHD.

line 2, parameter 1 (flag_t _C.viscosity_solver): $\in$\{STD,RKL\}.
Time integrator for the viscosity term. Choices available are STD for standard (unsplit RK) time

integration and RKL for Super-TimeStepping.

line 2, parameter 2 (`double _C.viscosity_courant`):
CFL-like number for the viscosity term. Typically < 0.5 for STD but values > 1 are possible for RKL.

line 3, parameter 1 (`flag_t _C.diffusion_solver`): $\in$\{STD,RKL\}.
Time integrator for the magnetic diffusion term. Choices available are STD for standard (unsplit RK) time integration and RKL for Super-TimeStepping.

line 3, parameter 2 (`double _C.diffusion_courant`):
CFL-like number for the magnetic diffusion term. Typically < 0.5 for STD but values > 1 are possible for RKL.

line 4, parameter 1 (`flag_t _C.conduction_solver`): $\in$\{STD,RKL\}.
Time integrator for the thermal conduction term. Choices available are STD for standard (unsplit RK) time integration and RKL for Super-TimeStepping.

line 4, parameter 2 (`double _C.conduction_courant`):
CFL-like number for the thermal conduction term. Typically < 0.5 for STD but values > 1 are possible for RKL.

line 5, parameter 1 (`flag_t _C.APdiffusion_solver`): $\in$\{STD,RKL\}.
Time integrator for the ambipolar diffusion term. Choices available are STD for standard (unsplit RK) time integration and RKL for Super-TimeStepping.

line 5, parameter 2 (`double _C.APdiffusion_courant`):
CFL-like number for the ambipolar diffusion term. Typically < 0.5 for STD but values > 1 are possible for RKL.

line 6 (`double _C.heatloss_courant`):
CFL-like number for the heatloss term. Values > 1 are possible.

line 7 (`double _C.gravity_tol`):
Error tolerance in the gravity solver. A typical value is 10^{-6}. If the number of iterations needed to reach `_C.gravity_tol` exceeds the value of the macro MG_ITMAX defined in **User.h** the multigrid solver terminates indicating non-convergence.

line 8 (`int _C.delay_step`):
Allows CFL numbers to grow linearly from 0 to its specified values within `_C.delay_step` time-steps at the beginning of a simulation.

```
>PHYSICS SPECIFICATIONS ---------------------------------------------------
 N  1.00e+00                            >mf,permeability_rel
 N  0.00e+00                            >viscosity{,_coeff}
 N  0.00e+00                            >diffusion{,_coeff}
 N  0.00e+00  0.00e+00  0.00e+00        >conduction{,_coeff,_coeff_perp,_coeff_sat}
 N  0.00e+00                            >APdiffusion{,_coeff}
 Y  1.00e-03                            >energy,energy_dual_sw:[0,1]
 N                                      >force
 N                                      >gravity
 ADIA  1.40e+00  1.00e+00  1.00e+00     >eos,gamma,poly_const,temperature
 N                                      >heatloss
 0                                      >tracer
 0                                      >fields
```

line 1, parameter 1 (`flag_t _C.mf`): $\in$\{Y,N\}.
Switch for including magnetic fields. Y renders MHD simulations whereas N stands for pure HD simulations. In the latter the Lorentz force is dropped from the momentum equation and the induction

equation is not solved.

line 1, parameter 2 (`double _C.permeability_rel`):
Relative magnetic permeability μ_{rel} ($\mu_{rel} = \mu/\mu_0, \mu_0 = 4\pi \cdot 10^{-7}$). To mimic Gaussian cgs unit system set `_C.permeability_rel` $= 10^7$!

line 2, parameter 1 (`flag_t _C.viscosity`): $\in$\{Y,N,U\}.
Viscosity term switch. N disables viscosity. Y enables viscosity assuming a constant dynamic viscosity coefficient given by the parameter `_C.viscosity_coeff`. U enables viscosity assuming a user-defined dynamic viscosity coefficient coded in **viscosityCoeffUser.c**.

line 2, parameter 2 (`double _C.viscosity_coeff`):
Constant dynamic viscosity coefficient.

line 3, parameter 1 (`flag_t _C.diffusion`): $\in$\{Y,N,U\}.
Magnetic diffusion term switch. N disables magnetic diffusion. Y enables magnetic diffusion assuming a constant diffusion coefficient given by the parameter `_C.diffusion_coeff`. U enables magnetic diffusion assuming a user-defined diffusion coefficient coded in **diffusionCoeffUser.c**.

line 3, parameter 2 (`double _C.diffusion_coeff`):
Constant magnetic diffusion coefficient.

line 4, parameter 1 (`flag_t _C.conduction`): $\in$\{Y,N,U,S\}.
Thermal conduction term switch. N disables thermal conduction. Y enables thermal conduction assuming constant conduction coefficients given by the parameters `_C.conduction_coeff` and `_C.conduction_coeff_perp`. U enables thermal conduction assuming user-defined conduction coefficients coded in **conductionCoeffUser.c**. S enables thermal conduction using the Spitzer conductivity model.

line 4, parameters 2-4 (`double _C.conduction_coeff,_C.conduction_coeff_perp,_C.conduction_coeff_sat`):
`_C.conduction_coeff` and `_C.conduction_coeff_perp` are constant thermal conduction coefficients parallel and perpendicular to the magnetic field vector for classical heat conduction. In the absence of magnetic fields the isotropic conduction coefficient is assumed to be `_C.conduction_coeff` and `_C.conduction_coeff_perp` becomes meaningless. `_C.conduction_coeff_sat` is an adjustable parameter (Ψ) in the saturated heat flow model of Cowie & McKee (see 2.1.1).

line 5, parameter 1 (`flag_t _C.APdiffusion`): $\in$\{Y,N,U\}.
Ambipolar diffusion term switch. N disables ambipolar diffusion. Y enables ambipolar diffusion assuming a constant ambipolar diffusion coefficient given by the parameer `_C.APdiffusion_coeff`. U enables ambipolar diffusion assuming a user-defined ambipolar diffusion coefficient coded in **APdiffusionCoeffUser.c**.

line 5, parameter 2 (`double _C.APdiffusion_coeff`):
constant ambipolar diffusion coefficient.

line 6, parameter 1 (`flag_t _C.energy`): $\in$\{Y,N,DUAL\}.
Energy equation switch. N skips the total energy equation from the set of equations assuming that the thermal pressure can be computed solely from the density by an appropriate EOS (eg. isothermal, polytropic). Y enables the total energy equation without dual energy support. DUAL, in addition, activates dual energy support. In the dual energy formalism the equations of total energy and thermal energy are computed simultaneously. If the ratio of thermal to total energy falls short some prescribed threshold `_C.energy_dual_sw` (see next) then the pressure is computed locally more savely from the thermal energy equation than the total energy equation.

line 6, parameter 2 (`double _C.energy_dual_sw`): $\in [0, 1]$.
Threshold in the dual energy formalism.

line 7, parameter 1 (`flag_t _C.force`): $\in$\{Y,N\}.
External force term switch. Y (N) enables (disables) the use of an external force.

line 8, parameter 1 (`flag_t _C.gravity`): ∈{Y,N}.
Selfgravity switch. Y (N) enables (disables) selfgravity.

line 9, parameter 1 (`flag_t _C.eos`): ∈{ADIA,ISO,POLY,USER,TAB}.
Choice of the EOS: ADIA = adiabatic EOS, ISO = isothermal EOS, POLY = polytropic EOS, USER = user-defined analytic EOS, TAB = user-defined tabulated EOS.

line 9, parameter 2 (`double _C.gamma`):
Ratio of specific heats in case of an adiabatic EOS. Polytropic exponent in case of a polytropic EOS.

line 9, parameter 3 (`double _C.polytropic_constant`):
Polytropic constant in case of a polytropic EOS.

line 9, parameter 4 (`double _C.temperature`):
System temperature in case of an isothermal EOS.

line 10 (`flag_t _C.heatloss`): ∈{N,ISM,U}.
Heatloss term switch. N disables the heatloss term. ISM enables the heatloss term using predefined cooling/heating functions for the interstellar medium. U enables the heatloss term with cooling/heating functions to be defined by the user in modules **sourceCoolingUser.c** and **sourceHeatingUser.c**.

line 11 (`int _C.tracer`):
Number of tracer variables.

line 12 (`int _C.fields`):
Number of fields variables. The FIELDS code component provides the infrastructure (not numerical method) for the solution of induction-like equations (AU).

```
>SPECIES: WEIGHT
+H        1.00
```

parameter 1 (`double _C.mean_molecular_weight`):
Mean molecular weight $\overline{\mu}$ of the gas. Do not remove the '+' sign in front of the species name!

```
>>END INPUT ------------------------------------------------------------------
```

2.2.3 Defining initial conditions

ICs have to be defined in module **configUser.c**. Defining ICs means assignment of required – depending on the physics specification – basic variables on the grid. Basic variables are the mass density ϱ (`double ***rho`), the total energy density e (`double ***e`), x-momentum density $m_x = \varrho v_x$ (`double ***mx`), y-momentum density $m_y = \varrho v_y$ (`double ***my`), canonical z-momentum density (=angular momentum density in cylindrical/spherical geometry) $m_z = h_z \varrho v_z$ (`double ***mz`), magnetic field $\mathbf{B}$ (`double ***bx,***by,***bz`), and tracer variables C (`double ****C`). In case of AMR the grid consists of a set of superblocks of different size and resolution. A uniform grid is represented by just one superblock, on the other hand. The AMR grid hierarchy is organized in refinement levels $\ell = 0$...`_C.level` starting with the base level $\ell = 0$ (its the only level if there is no mesh refinement) and ending with the highest resolved level `_C.level`. Each refinement level consists of a set of superblocks of different size but identical grid spacings. The full grid hierarchy is accessible through the (doubly) master pointer `**gc` of struct type `GRD`. `gc` is the only argument passed to **configUser.c**. From the master pointer `**gc` individual refinement levels are accessible by pointers `*gc[`ℓ`]`. All the superblocks of a refinement level ℓ are collected in a linked list. A superblock is represented by a pointer `*g` of the same struct type `GRD`. The first superblock in this list is given by just `g=gc[`ℓ`]`. Follow-up superblocks in the list are reached consecutively via links `g->gp`. A `NULL` pointer (`g->gp==NULL`) marks the list's end. The struct type `GRD` contains all necessary information about a superblock, i.e. metric information, its size and grid spacings,

coordinates, arrays for basic variables, etc.. This information is accessible through dereferencing `g->`X where X is a variable in `GRD`:

X :	description
`nx,ny,nz` :	indices of the last grid cell in a superblock; `nz=0` in 2D!
`nx1,ny1,nz1` :	`nx-1,ny-1,nz-1`; `nz1=1` in 2D!
`ixs,iys,izs` :	starting indices for active grid cells
`ixe,iye,ize` :	ending indices for active grid cells
`nbx,nby,nbz` :	indices of the last generic block in a superblock
`x[ix],y[iy],z[iz]` :	nodal coordinates of grid cells
`xc[ix],yc[iy],zc[iz]` :	volume-averaged cell coordinates
`xf[ix]` :	y, z-face-averaged x-coordinate of cells
`dx,dy,dz` :	cell spacings
`dxi,dyi,dzi` :	`1/dx,1/dy,_C.idz/dz`
`hy[ix]` :	metric scale factor h_y at nodal coordinates
`hyc[ix]` :	metric scale factor h_y at volume-averaged cell coordinates
`hzy[iy]` :	metric scale factor h_{zy} at nodal coordinates
`hzyc[iy]` :	metric scale factor h_{zy} at volume-averaged cell coordinates
`dv[iy][ix]` :	cell volume
`rho[iz][iy][ix]` :	mass density array
`e[iz][iy][ix]` :	total energy density array
`mx[iz][iy][ix]` :	x-momentum density array
`my[iz][iy][ix]` :	y-momentum density array
`mz[iz][iy][ix]` :	canonical z-momentum density array
`bx[iz][iy][ix]` :	magnetic field x-component array
`by[iz][iy][ix]` :	magnetic field y-component array
`bz[iz][iy][ix]` :	magnetic field z-component array
`phi[iz][iy][ix]` :	gravitational potential array
`C[ic][iz][iy][ix]` :	tracer variables array
...	

For the complete list of variables consult the struct type `GRD` declared in **nirvana.h**.

Cell indices `ix,iy,iz` of a superblock run from 0 to `g->nx,g->ny,g->nz` including ghost zones and from `g->ixs,g->iys,g->izs` to `g->ixe,g->iye,g->ize` for active zones excluding ghost zones. Index `ic=0..._C.tracer-1` counts tracer variables. Variables `rho, e, mx, my, mz, C` are defined at volume-averaged cell coordinates `(xc,yc,zc)` (identical to geometric cell centers in Cartesian geometry). The magnetic field variables `bx, by, bz` are face-centered at position `(x,yc,zc)` for `bx`, `(xf,y,zc)` for `by` and `(xf,yc,z)` for `bz` where `xf` is the y, z-face-centered x-coordinate (identical to geometric face center in Cartesian geometry) and `(x,y,z)` are the coordinates of cell nodes.

NOTE:

- Superblocks may include up to 3 ghost zones per side in each coordinate direction depending on the physics/numerics. The number of required ghost zones per side is stored in `_C.ngc`: 3 zones per side are needed for *MHD on cylindrical/spherical grids* and in simulations with selfgravity; 2 zones per side otherwise. Its not a necessity to initialize ghost zones by the user in **configUser.c**.
- The variable `_C.idz` indicates whether a problem is 3D (`_C.idz`=1) or 2D (`_C.idz`=0; axissymmetry in cylindrical/spherical geometry).
- Magnetic field variables `bx,by,bz` are not declared if `_C.mf`=0. Likewise, the tracers variable `C` is not declared if `_C.tracer`=0.
- The different magnetic field components are defined at different cell faces. Due to this staggering of components the range of active zone faces is by one larger for `bx` (`by`, `bz`) in $x(y,z)$-direction: `ix=g->ixs...g->ixe+1` (`iy=g->iys...g->iye+1`,`iz=g->izs...g->ize+_C.idz`).
- In MHD simulations the initial magnetic field configuration must be formulated divergence-free on the grid. This may be a non-trivial task, in particular on an initially refined mesh. One way is along the magnetic vector potential $\mathbf{A}$, if available, utilizing the integral form of $\mathbf{B} = \nabla \times \mathbf{A}$ in order to compute face-averaged magnetic field components. Another way is direct integration of the analytical magnetic field components over the respective cell faces. See the test problems in the paper Ziegler (2011) for examples. Warning: the code can not check for consistency of the user-defined magnetic field configuration. If inconsistencies exist in AMR setups it can lead to magnetic monopole generation during run time even not visible initially!

Below, two simple examples of how to define ICs in **configUser.c** are shown i) for a uniform grid and ii) for an AMR mesh with refinements generated before by module **checkDomainUser.c**. More examples can be found in the problem collection at /nirvana/project/....

The uniform grid example:

```
void configUser(GRD **gc)
{ /*
  // NIRVANA v3.8, copyright by udo ziegler, AIP
   *
   * @func: definition of ICs.
   *
   * @param: gc = mesh master pointer
   *
   * @author: udo ziegler, AIP
   *
   * @modified: MHD/problem2: Orszag-Tang vortex
   *
   *-------------------------------------------------------------------------*
   */
  GRD *g;
  double rho0,x0,e0,b0,bx,by,bz,pih;
  int ix,iy,iz,ic;
```

```
_C.permeability=1.;

pih=0.5/_C.permeability;
x0=2.*PI/(_C.up[0]-_C.lo[0]);
rho0=2.77778;e0=5./3./(_C.gamma-1.);b0=1.;

for(g=gc[0]; (g); g=g->gp) {
    for(iz=0; iz<=g->nz; iz++)
      for(iy=0; iy<=g->ny; iy++)
        for(ix=0; ix<=g->nx+1; ix++)
          g->bx[iz][iy][ix]=-b0*sin(x0*g->yc[iy]);
    for(iz=0; iz<=g->nz; iz++)
      for(iy=0; iy<=g->ny+1; iy++)
        for(ix=0; ix<=g->nx; ix++)
          g->by[iz][iy][ix]=b0*sin(2.*x0*g->xc[ix]);
    for(iz=0; iz<=g->nz+_C.idz; iz++)
      for(iy=0; iy<=g->ny; iy++)
        for(ix=0; ix<=g->nx; ix++)
          g->bz[iz][iy][ix]=0.;
    for(iz=0; iz<=g->nz; iz++)
      for(iy=0; iy<=g->ny; iy++)
        for(ix=0; ix<=g->nx; ix++) {
            g->rho[iz][iy][ix]=rho0;
            g->mx[iz][iy][ix]=-rho0*sin(x0*g->yc[iy]);
            g->my[iz][iy][ix]=rho0*sin(x0*g->xc[ix]);
            g->mz[iz][iy][ix]=0.;
            bx=0.5*(g->bx[iz][iy][ix]+g->bx[iz][iy][ix+1]);
            by=0.5*(g->by[iz][iy][ix]+g->by[iz][iy+1][ix]);
            bz=0.5*(g->bz[iz][iy][ix]+g->bz[iz+_C.idz][iy][ix]);
            g->e[iz][iy][ix]=e0+0.5/rho0
              *SPQR(g->mx[iz][iy][ix],g->my[iz][iy][ix],g->mz[iz][iy][ix])
              +SPQR(bx,by,bz)*pih;
        }
}

return;
}
```

The AMR example:

```
void configUser(GRD **gc)
{ /*
  // NIRVANA v3.8, copyright by udo ziegler, AIP
   *
   * @func: definition of ICs.
   *
   * @param: gc = mesh master pointer
   *
   * @author: udo ziegler, AIP
   *
```

```
   * @modified: MHD/problem17: shock-cloud interaction
   *
   *-------------------------------------------------------------------------*
   */
  GRD *g;
  double g1,x,y,z,dx,dy,x0,r2,vxR,eL,eR,byL,byR,bx,by,bz,pih;
  int ix,iy,iz,il;


_C.permeability=1.;pih=0.5/_C.permeability;
g1=_C.gamma-1.;_C.time_max=0.05;
x0=0.1;
r2=0.1*0.1;
vxR=-11.2536,eL=167.345/g1,eR=1./g1;
byL=2.1826182;byR=0.56418958;

for(il=_C.level; il>=0; il--) /* LOOP OVER REFINEMENT LEVELS */
  for(g=gc[il]; (g); g=g->gp) { /* LOOP OVER SUPERBLOCKS */

      if(_C.mf) { /* MAGNETIC FIELD */
         for(iz=0; iz<=g->nz; iz++) {
             for(iy=0; iy<=g->ny; iy++)
               for(ix=0; ix<=g->nx+1; ix++)
                 g->bx[iz][iy][ix]=0.;
             for(iy=0; iy<=g->ny+1; iy++)
               for(ix=0; ix<=g->nx; ix++)
                 g->by[iz][iy][ix]=(g->x[ix+1]<x0 ? byL
                   : (g->x[ix]>x0 ? byR
                                  : ((g->x[ix+1]-x0)*byR+(x0-g->x[ix])*byL)*g->dxi));
         }
         for(iz=0; iz<=g->nz+_C.idz; iz++)
           for(iy=0; iy<=g->ny; iy++)
             for(ix=0; ix<=g->nx; ix++)
               g->bz[iz][iy][ix]=(g->x[ix+1]<x0 ? -byL
                 : (g->x[ix]>x0 ? byR
                                : ((g->x[ix+1]-x0)*byR-(x0-g->x[ix])*byL)*g->dxi));

      }

      for(iz=0; iz<=g->nz; iz++) {
          z=g->zc[iz];
          for(iy=0; iy<=g->ny; iy++) {
              y=g->yc[iy];
              dy=(y-g->y[iy])*g->dyi;
              for(ix=0; ix<=g->nx; ix++) {
                  x=g->xc[ix];
                  g->mx[iz][iy][ix]=g->my[iz][iy][ix]=g->mz[iz][iy][ix]=0.;
                  if(x<x0) {
                     g->rho[iz][iy][ix]=3.86859;
                     g->e[iz][iy][ix]=eL;
                  }
                  else {
```

```
                  g->rho[iz][iy][ix]=(x-0.3)*(x-0.3)+y*y+z*z<r2 ? 10.
                                                               : 1.;
                  g->mx[iz][iy][ix]=g->rho[iz][iy][ix]*vxR;
                  g->e[iz][iy][ix]=eR+0.5*vxR*vxR*g->rho[iz][iy][ix];
               }
               if(_C.mf) {
                  bx=g->bx[iz][iy][ix]+(g->bx[iz][iy][ix+1]
                     -g->bx[iz][iy][ix])*(g->xc[ix]-g->x[ix])*g->dxi;
                  by=g->by[iz][iy][ix]+(g->by[iz][iy+1][ix]
                     -g->by[iz][iy][ix])*dy;
                  bz=0.5*(g->bz[iz][iy][ix]+g->bz[iz+_C.idz][iy][ix]);
                  g->e[iz][iy][ix]+=pih*SPQR(bx,by,bz);
               }
            }
         }
      }
  }

return;
}
```

2.2.4 Defining boundary conditions

Standard BCs. Standard BCs are selected in **nirvana.in** in parameter section BOUNDARY CONDITIONS.

User-defined BCs. A user can define own BCs by implementation of the modules **bcrhoUser.c** (mass density), **bceUser.c** (total energy density), **bcmUser.c** (momentum densities), **bcbUser.c** (magnetic field), **bcetUser.c** (thermal energy density; necessary in case of dual-energy mode), **bctrUser.c** (tracer) etc.. User-defined BCs for the gravitational potential Φ are not possible. A domain boundary is assumed to have user-defined BCs by setting the corresponding parameter `_C.bc[#]=U` in **nirvana.in** (section BOUNDARY CONDITIONS).

Examples for user-defined BCs can be found in the NIRVANA problem collection.

NOTE:

- user-defined BCs for a certain variable have to be implemented only if required by the physics of the problem, e.g. **bcbUser.c** is not needed in pure HD simulations.

- elliptic-type BCs for the Poisson equation are derived from the MHD BCs assuming the following relationship:

  P $\Longrightarrow$ P for Φ
  A,M,R,C $\Longrightarrow$ Neumann condition for Φ
  I,O,D,U $\Longrightarrow$ Dirichlet condition for Φ

  elliptic BCs of Dirichlet type are obtained by a multipole expansion of the gravitational potential for the given density distribution. At a ghost cell with coordinate $\mathbf{x}_b$

$$\begin{aligned}\Phi(\mathbf{x}_b) &= \int \frac{\varrho d^3\mathbf{x}'}{|\mathbf{x}_b - \mathbf{x}'|} \\ &\approx -\frac{GM}{|\mathbf{x}_b|} - G\sum_{i,j}\frac{3x_{b,i}x_{b,j} - |\mathbf{x}_b|^2\delta_{ij}}{3|\mathbf{x}_b|^5}Q_{ij} \\ &\quad -G\sum_{i,j,k}\frac{5x_{b,i}x_{b,j}x_{b,k} - |\mathbf{x}_b|^2(x_{b,i}\delta_{jk} + x_{b,j}\delta_{ki} + x_{b,k}\delta_{ij})}{2|\mathbf{x}_b|^7}O_{ijk}\end{aligned}$$

  where

$$Q_{ij} = \frac{1}{2}\int (3x'_i x'_j - |\mathbf{x}'|^2\delta_{ij})\varrho(\mathbf{x}')d^3\mathbf{x}'$$

  is the (traceless) quadrupole tensor and

$$O_{ijk} = \int x'_i x'_j x'_k \varrho(\mathbf{x}')d^3\mathbf{x}'$$

  is the (primitive) rank-3 octopole moment. The computation is carried out relative to the center-of-mass given by

$$\mathbf{x}_{cm} = \frac{1}{M}\int \mathbf{x}\varrho d^3\mathbf{x}, \quad M = \int \varrho d^3\mathbf{x}$$

  so that the dipole component identically vanishes. Note that this truncated expansion is accurate only in the far-zone limit where $|\mathbf{x}_b|$ is sufficiently larger than the size of mass concentration.

- in the absence of Dirichlet boundaries (e.g. pure periodic BCs) the modifed Poisson equation,

$$\nabla^2\Phi = 4\pi G(\varrho - \langle\varrho\rangle)$$

  with $\langle\varrho\rangle$ the mean density, is solved thereby fulfilling the combatibility condition.

- combinations of periodic BCs with Dirichlet BCs are not supported.

2.2.5 User-controllable macros

The file **User.h** contains the following user-controllable macros:

numerics-related macros:

macro :	description
SPACE_ORDER :	∈{2}. Spatial approximation order (currently restricted to second-order).
LIM(.):	∈{MM(.),MC(.),VL(.)}. Slope limiter used in the solution reconstruction step: MM=MinMod, MC=Monotonized-Centered, VL=Van Leer.
LIM_MULTI_DIM :	∈{YES,NO}. Switch for additional multi-dimensional limiter in the reconstruction step.
PSI :	∈[0.5,1]. Parameter for multi-dimensional limiter.
MAXLEVEL :	Maximum number of refinement levels.
MG_ITMAX :	Maximum number of iterations in the multigrid solver before termination.
MG_TYPE :	∈{MULT}. Multigrid type (currently restricted to MULT).
RKL_COURANT_EXPL :	< 0.5. Internally used explicit Courant number in RKL solver.
RKL_MAX_COURANT :	Maximum Courant number in RKL solver.
RKL_DT_LIM :	< 1. limits the RKL timestep to the specified fraction of the dynamical timestep.
HEATLOSS_TOL :	Error tolerance in exponential Rosenbrock method for the heatloss term.
HEATLOSS_DT_LIM :	limits the heatloss timestep to the specified fraction of the dynamical timestep.

physics-related macros:

macro :	description
CENTRIFUGAL_FORCE :	∈{YES,NO}. Switch for the centrifugal force term. If set to NO, the centrifugal force (not Coriolis force) is ignored in the momentum equation.
COND_FORCE_ISO :	∈{YES,NO}. Switch for forcing isotropy of heat conduction in MHD simulations.

IONISATION(rho): allows to define the ionisation degree as a function of density or simply a fixed value.

PUSR(rho,u), CS2USR(rho,u), TUSR(rho,u): define the pressure, square of sound speed and temperature as a function of density (first argument rho) and *specific* thermal energy (second argument u) for an analytic EOS (see below).

The struct type `user` declared in **User.h** can be freely adjusted/extended by the user and is globally accessible through the code variable `_U` as `_U.`X where X is any variable in this structure. Except for pointers all information in `_U` is recovered in a simulation restart!

2.2.6 User-defined analytic/tabulated EOS

Analytic EOS (AU):
A user can implement own code for an analytic EOS in **eosUser.c**. It requires definition of the thermodynamic quantities pressure, temperature, sound speed squared and thermal energy density as a function of density and total energy density. It furthermore requires analog definition of the macros PUSR(rho,u), CS2USR(rho,u) and TUSR(rho,u) in **User.h**. These define the pressure $p = p(\varrho, u)$ (square of sound speed, temperature) as a function of density ϱ (rho) and *specific* thermal energy $u = \varepsilon/\varrho$ (u). Definition of the macros is mandatory because it are needed in numerical solver routines! It can be used

in **eosUser.c** to define necessary thermodynamics quantities. Application of the analytic EOS then requires to choose `_C.eos`=USER in **nirvana.in** (section SOLVER SPECIFICATIONS). If the solution of an energy equation is obsolete, one has to set `_C.energy=N` in **nirvana.in**! A barotropic EOS of the form $p = c_0^2 \varrho (1 + (\varrho/\varrho_{\rm crit})^{4/3})^{1/2}$, $c_0 = 200$, $\varrho_{\rm crit} = 10^{-10}$ is predefined in modules **User.h** (the macros) and **eosUser.c**.

Tabulated EOS (AU):
A tabulated EOS requires definition of sample data for the logarithms of pressure, $p = p(\varrho, u)$, and temperature, $T = T(\varrho, u)$, as a function of density ϱ and *specific* thermal energy u. Such sample data for $\{log(p), log(T)\}$ on a (ϱ, u)-grid has to be specified in module **createTabUser.c** (see further description in **createTabUser.c**). Generation of look-up tables for $\{log(p), log(T)\}$ are based on cubic interpolation techniques provided by the code. Application of the tabulated EOS requires to choose `_C.eos`=TAB in **nirvana.in** (section SOLVER SPECIFICATIONS). If the solution of an energy equation is obsolete, i.e. p and T are functions of ϱ only, set `_C.energy=N`.

2.2.7 User-defined external force

The module **forceUser.c** serves for the definition of an external *specific* force $\mathbf{f}_e$ (body force) which enters the momentum equation as source term. This force term is enabled if `_C.force`=Y in **nirvana.in** (section SOLVER SPECIFICATIONS).

2.2.8 User-defined coefficients for dissipation terms

The modules **viscosityCoeffUser.c**, **diffusionCoeffUser.c**, **conductionCoeffUser.c** and **APdiffusionCoeffUser.c** serve as templates to define user-specific coefficients for viscosity, magnetic diffusion, thermal conduction and ambipolar diffusion, respectively. These coefficients are assumed scalar quantities with point values defined by the user at the volumetric cell center `(xc,yc,zc)`. In case of anisotropic thermal conduction, a parallel- and perpendicular coefficient with respect to the magnetic field direction must be defined. In the isotropic case only the parallel coefficient is of relevance.

NOTE:

- In MHD simulations thermal conduction, by default, is assumed anisotropic. However, the user can force isotropy by setting the macro COND_FORCE_ISO in **User.h** to YES.

2.2.9 User-defined heatloss term

The files **sourceCoolingUser.c** and **sourceHeatingUser.c** allow to specify separately a cooling function $\mathcal{L}_{\rm cool}$ and heating function $\mathcal{L}_{\rm heat}$ which sum to the heatloss term $\mathcal{L}$ in the energy equation, $\mathcal{L} = \mathcal{L}_{\rm cool} + \mathcal{L}_{\rm heat}$. $\mathcal{L}_{\rm cool} = \mathcal{L}_{\rm cool}(\varrho, T)$ and $\mathcal{L}_{\rm heat} = \mathcal{L}_{\rm heat}(\varrho, T)$ may be functions of density and temperature. Code routines also require the partial derivatives $\partial\mathcal{L}_{\rm cool}/\partial\varrho$, $\partial\mathcal{L}_{\rm cool}/\partial T$, $\partial\mathcal{L}_{\rm heat}/\partial\varrho$, $\partial\mathcal{L}_{\rm heat}/\partial T$. These can be explicitely defined by the user in **sourceCoolingUser.c** and **sourceHeatingUser.c**. In this case the `*deriv` parameter in the functions must be set to YES. If `*deriv` is set to NO, the code approximates the derivatives by finite differencing.

2.2.10 User-defined regions of mesh refinement

The file **checkDomainUser.c** serves for the definition of spatial regions which should initially or permanently be refined to some prescribed refinement level |`_C.smr`|. The regions have to be declared via mathematical relations in the MESH_INIT part of **checkDomainUser.c**. The simple idea here is to check whether virtual cells of a virtual new child block of size $[xl, xu] \times [yl, yu] \times [zl, zu]$ lie inside user-specified regions. If true, the child block is generated in reality at this position of the parent block

(block here refers to generic blocks). Note that mesh adaptivity to arbitrary-shaped regions is limited due to the block-structured nature of AMR. It is therefore useful to consider a buffer zone in order to cover all the region of interest! For example, to refine a spherical region (radius R, center at x0,y0,z0) with refinement level 2 add code like:

```
case MESH_INIT:
dsx=(xu-xl)/4.;
dsy=(yu-yl)/4.;
dsz=(zu-zl)/4.;
for(iz=0; iz<=4; iz++)
 { z=zl+iz*dsz;
   for(iy=0; iy<=4; iy++)
     { y=yl+iy*dsy;
       for(ix=0; ix<=4; ix++)
         { x=xl+ix*dsx;
           if((x-x0)*(x-x0)+(y-y0)*(y-y0)+(z-z0)*(z-z0)<1.5*R*R) /* WITH BUFFER ZONE */
             value=2; /* 2 REFINED LEVELS */
         }
     }
 }
break;
```

The MESH_ADAPTIVE section of **checkDomainUser.c** allows control where mesh refinement should take place during a simulation. Like for MESH_INIT the user can define regions via mathematical relations (see further description in **checkDomainUser.c**).

2.2.11 User-defined data analysis

The file **analysisDataUser.c** is thought as a template for user-defined code allowing runtime data analysis. The function is called every `_C.freq_ana` time-step. Post-run analysis of data of DATA_MOD type is also possible by setting parameter `mode`=ANAL and specifying the data filename `fname` in **nirvana.in** (section SIMULATION I/O). In the ANAL mode the code rebuilds the stored configuration, calls **analysisDataUser** for data analysis and quits without further computation.

2.3 Data output

NIRVANA produces various forms of output data during a simulation:

DATA_LOG:
Two different ascii files are produced during a run which inform the user about the status of a simulation: **nirvana.log** logs input parameters as specified in **nirvana.in** and logs basic characteristics of a simulation like physical time, the various kinds of time-steps, the various forms of energies, grid statistics, etc.. The file is updated every `_C.freq_log` time-step. **nirvana.log** can be viewed using any editor. In MPI simulations **nirvana.log** is located on the master process with the lowest rank. In case NIRVANA encounters a code-controlled exception during a run a message about the type of exception is written into **nirvana.log** before the job is terminated.
nirvana.mon logs basic characteristics like in **nirvana.log** but uses a more formatted way. Data is stored line-wise with a line added to the file every `_C.freq_log` time-step. A line entry corresponds to a model snapshot at a certain time and contains the following information: time-step number, physical time, individual time-steps (dynamical time-step, dissipation time-steps, heatloss time-step), mass, total (non-gravitational) energy, momentum components, thermal energy, gravitational energy, kinetic

energy components, magnetic energy components, relative change in mass, relative change in total energy, relative changes in momentum components, maximum magnetic divergence, average magnetic divergence, number of generic blocks in levels 0...MAXLEVEL. **nirvana.mon** can be accessed via IDL using the IDL procedure **readMON.pro** located in NIRVANA3.8/caivs/idl. In MPI simulations **nirvana.mon** is located on the master process with the lowest rank.

DATA_MOD:
Output data of this type contains complete information about a model with full digit precision in order to allow a simulation restart. For double safety there are two equivalent binary files named **mod**#.# and **tmp**#.#. **mod**#.# stores model information every `_C.freq_mod` time-step producing a series of **mod**#.# files during a run where the first # denotes the actual model number (code variable `_C.mod`) and the second # is the process rank (1 for a single processor run). On the other hand, **tmp**#.# is replaced every `_C.walltime_tmp` seconds.

DATA_RAW:
Output data of this type is primarily thought for visualization purposes. The corresponding files are named **raw#.#** and are produced every `_C.freq_raw` time-step where the first # denotes the model number (code variable `_C.mod`) and the second # is the process rank. **raw#.#** files are mixed ascii/binary files containing header information and binary model data in encoded compressed form reducing the necessary disk space. **raw#.#** files are essentially byte streams and are architecture-independent!

DATA_HDF:
Not part of the NIRVANA standard release.

2.4 Problem collection

NIRVANA comes with a large set of predefined problems. A complete list is given by the file COLLECTION.INFO located in NIRVANA3.8/nirvana/project. The problems are organized into subfolders:

/MHD: mhd problems
/VISC: viscosity problems
/COND: thermal conduction problems
/DIFF: magnetic diffusion problems
/APDIFF: ambipolar diffusion problems
/HEATLOSS: problems with heatloss term
/GRAVITY: self-gravity problems

For instance, to compile problem# in /MHD type

./makenirvana MHD/problem#

from directory NIRVANA3.8/nirvana/bin. To run problem# in /MHD type

./runnirvana MHD/problem#

from directory NIRVANA3.8/bin. To visualize results output data of DATA_RAW type (**raw#.#** files) or DATA_MOD type (**mod#.#** files) may be converted to IDL/SILO files with help of the CAIVS tool.

3 CAIVS tool

3.1 What is CAIVS?

CAIVS is a converter for NIRVANA output data of DATA_RAW type (**raw#.#** files) or DATA_MOD type (**mod#.#** files). It produces data files suitable for graphical postprocessing with IDL or a visualisation tool (e.g. VisIt, Paramesh) able to import SILO files.

3.2 Usage

File **caivs.in** serves as user interface for the specification of converter parameters. **caivs.in** is divided into the three parameter sections:

DATA I/O
MESH TRANSFORMER
COMPONENTS

Parameters are collected in the global variable `_CC` of struct type `CCTL` declared in **caivs.h**. Any variable `X` in `CCTL` is accessible by `_CC.X`.

```
>DATA I/O ------------------------------------------------------------------
RAW 00750 0000                      >format_in:{RAW,MOD},infile_start,infile_step
IDL 00750                           >format_out:{IDL,SILO},outfile_start
```

line 1, parameter 1 (`char* _CC.format_in`): ∈{RAW,MOD}.
Input data format. CAIVS accepts data of NIRVANA types DATA_RAW (raw#.# files) and DATA_MOD (mod#.# files).

line 1, parameter 2 (`int _CC.infile_start`):
Number X in the data file **raw**X.#/**mod**X.# to be imported for conversion.

line 1, parameter 3 (`int _CC.infile_step`):
Specification of processing mode. If `_CC.infile_step`=0 only the file **raw**X.#/**mod**X.# is imported where X=`_CC.infile_start`. If `_CC.infile_step`>0 the file sequence **raw**X.#/**mod**X.# with X=`_CC.infile_start`, `_CC.infile_start`+`_CC.infile_step`, `_CC.infile_start`+2∗`_CC.infile_step`, ... is imported and files are processed in sequential order.

line 2, parameter 1 (`char* _CC.format_out`): ∈{IDL,SILO}.
Output data format.

line 2, parameter 2 (`int _CC.outfile_start`):
IDL (SILO) output files get names X.idl (X.silo) where X=`_CC.outfile_start`, `_CC.outfile_start`+1 and so on. If `_CC.infile_step`=0 only one output file is produced.

```
>MESH TRANSFORMER ----------------------------------------------------------
 0                                      >max_level
 0 -0.50000e+00  0.50000e+00            >subrange[0],rangemin[0],rangemax[0]
 0 -0.50000e+00  0.50000e+00            >subrange[1],rangemin[1],rangemax[1]
 0 -0.50000e+00  0.50000e+00            >subrange[2],rangemin[2],rangemax[2]
 1 0                                    >unigrid,cell_centering
 0                                      >vector_notransform
```

line 1 (`int _CC.max_level`):
Maximum refinement level of the input grid to take into account in the conversion process.

line 2 (`double _CC.subrange[0],_CC.rangemin[0],_CC.rangemax[0]`):

_CC.subrange[0]$\in$ $\{0,1\}$ specifies whether conversion should be restricted to a x-subrange (_CC.subrange[0]=1) or applied to the full computational x-range (_CC.subrange[0]=0). If set to 1, the x-subrange is (approximately) given by [_CC.rangemin[0],_CC.rangemax[0]].

line 3 (double _CC.subrange[1],_CC.rangemin[1],_CC.rangemax[1]):
Same as line 2 but y-range. In case of spherical coordinates y – denotng the angle θ from the geometric northpole – is measured in units of π.

line 4 (double _CC.subrange[2],_CC.rangemin[2],_CC.rangemax[2]):
Same as line 2 but z-range. In case of cylindrical or spherical coordinates z – denoting the azimuth ϕ – is measured in units of π.

line 5, parameter 1 (int _CC.unigrid): $\in$\{0,1\}.
Switch for unigrid mapping. If _CC.unigrid=1 block-structured data (AMR) is transformed onto an unigrid. Conversion to IDL output automatically includes a transformation to an unigrid. Block-structured IDL output is currently not possible.

line 5, parameter 2 (int _CC.cell_centering): $\in$\{0,1\}.
Switch for cell-centering IDL output data. By default, output data is node-centered. SILO output is automatically forced to be node-centered.

line 6 (int _CC.vector_notransform): $\in$\{0,1\}.
Switch for vector transformation in cyl./sph. geometry. By default (_CC.vector_notransform=0), vector fields of a cyl./sph. input mesh are transformed to Cartesian components for hexahedral output meshes (SILO export). This transformation is not performed, if (_CC.vector_notransform=1).

```
>COMPONENTS ------------------------------------------------------------------
 0     >block_structure
 1     >log_density
 0     >velocity
 0     >log_pressure
 0     >log_temperature
 0     >log_species
 0     >tracer
 0     >gravity_potential
 0     >magnetic_field
```

lines 1-9 (int _CC.block_structure,_CC.comp[0]...CC.comp[7]): $\in$\{0,1\}.
Switch for block structure/components to take into account in the conversion process. Components are log of density, velocity, log of pressure, log of temperature, log of species densities, tracer, gravitational potential and magnetic field.

```
>>END INPUT ------------------------------------------------------------------
```

NOTE:

- CAIVS automatically recognizes multiple files from parallel simulations.
- Output data generated from DATA_RAW input is of type float. Output data generated from DATA_MOD input is of type double.

3.3 Data output

3.3.1 The caivs.log file

CAIVS generates the ascii file **caivs.log** which logs parameters as specified in **caivs.in**, stores grid information and min/max values of variables over the input file sequence. In case of a input file sequence (`_CC.infile_step>0`) each snapshot produces a row in **caivs.log** giving the time-step number, evolution time, number of blocks per refinement level (in case of AMR) and the total number of blocks.
In case a code-controlled exception occurs **caivs.log** contains information about that exception before *CAIVS* terminates.

3.3.2 IDL export

IDL output files produced by *CAIVS* are mixed ascii/binary files and get names #.idl where # is a running number according to the specification in **caivs.in**. IDL files are in a format which can be read by the IDL procedure **readIDL.pro** (see usage information there) provided by the CAIVS tool and located in folder /caivs/idl. IDL files are platform-dependent!

3.3.3 SILO export

SILO output files produced by *CAIVS* get names #.silo where # is a running number according to the specification in **caivs.in**. SILO files are designed platform-independent!

NOTE:

- The component "block_structure" is obsolete in SILO export. Visualzation of the mesh should be possible within the visualisation tool.
- Cylindrical and spherical meshes are transformed to quadrilateral (2D) or hexahedral (3D) meshes. Vector components are, by default, transformed to Cartesian components.