BATS-R-US and CRASH User Manual

CSEM and CRASH team

November 23, 2024

Contents

1 Introduction **5**
- 1.1 Acknowledgments . . . 6
- 1.2 Web Pages for BATS-R-US and CRASH . . . 6

2 Using BATS-R-US and CRASH **7**
- 2.1 Quick Start . . . 7
  - 2.1.1 Installing the code . . . 10
  - 2.1.2 Testing the code . . . 12
  - 2.1.3 Configuring BATS-R-US . . . 12
  - 2.1.4 Compilation options . . . 13
  - 2.1.5 Compilation . . . 14
  - 2.1.6 Creating a run directory . . . 14
  - 2.1.7 Creating input files . . . 15
  - 2.1.8 Running the code . . . 16
  - 2.1.9 Restarting the run . . . 16
  - 2.1.10 Postprocessing . . . 17
  - 2.1.11 Recompilation and clean up . . . 18
- 2.2 Hardware and Software Requirements . . . 18
  - 2.2.1 Source Code and Compilers . . . 18
  - 2.2.2 Parallelism, Speed Up and Scaling . . . 18
  - 2.2.3 Memory and Disk Requirements . . . 18
- 2.3 Installation and Compilation . . . 19
  - 2.3.1 Directory Structure . . . 19
  - 2.3.2 Setting Grid Structure Before Compiling BATS-R-US . . . 20
  - 2.3.3 The Main Makefile . . . 23
  - 2.3.4 Compiler Flags . . . 24
  - 2.3.5 Run Directory Structure . . . 24
- 2.4 Running a Simulation . . . 26
  - 2.4.1 Before Running the Code . . . 26
  - 2.4.2 Interactive Execution . . . 26
  - 2.4.3 Queues and Scripts . . . 27

3 Input Parameters **29**
- 3.1 PARAM.in . . . 29
- 3.2 Included Files, `#INCLUDE` . . . 29
- 3.3 Commands, Parameters, and Comments . . . 30
- 3.4 Sessions . . . 30
- 3.5 The Order of Commands . . . 31
- 3.6 Command Defaults . . . 32

3.7 Iterations and Frequency of Output . . . 33
3.8 Input Commands for the BATSRUS: GM, EE, SC, IH and OH Components . . . 38
3.8.1 Stand alone mode . . . 38
3.8.2 Planet parameters . . . 41
3.8.3 User defined input . . . 44
3.8.4 Testing and timing . . . 45
3.8.5 Initial and boundary conditions . . . 49
3.8.6 Grid geometry . . . 65
3.8.7 Initial time . . . 69
3.8.8 Time integration . . . 70
3.8.9 Implicit parameters . . . 73
3.8.10 Stopping criteria . . . 78
3.8.11 Output parameters . . . 79
3.8.12 Eruptive event generator . . . 102
3.8.13 Amr parameters . . . 105
3.8.14 Scheme parameters . . . 116
3.8.15 Coupling paramaters . . . 130
3.8.16 Pic coupling . . . 134
3.8.17 Physics parameters . . . 138
3.8.18 Corona specific commands . . . 150
3.8.19 Threaded low solar corona . . . 156
3.8.20 Heliosphere specific commands . . . 158
3.8.21 Wave specific commands . . . 160
3.8.22 Particles . . . 160
3.8.23 Script commands . . . 161

4 Output files **163**
4.1 Restart files . . . 163
4.1.1 Conversion of binary restart files with ConvertRestart.pl . . . 164
4.2 Logfiles . . . 164
4.3 Satellite Files . . . 165
4.4 Plotfiles . . . 165
4.5 Postprocessing the IDL plot files . . . 166
4.5.1 Conversion of binary .out files with FixEndian.pl . . . 167
4.6 Postprocessing the TEC plot files . . . 167

5 Visualization **173**
5.1 Tecplot . . . 173
5.2 IDL . . . 173
5.3 MATLAB . . . 173
5.4 Julia . . . 174
5.5 VisIt . . . 174
5.6 ParaView . . . 174
5.7 Python . . . 174

6 The Synoptic Solar Wind Model **175**
6.1 General Description of the Model . . . 175
6.2 The potential field approximation . . . 175
6.3 Semi-empirical Model for the Solar Wind . . . 178
6.4 Model Parameterization . . . 180

Chapter 1

Introduction

BATS-R-US stands for Block Adaptive Tree Solar-wind Roe Upwind Scheme. This name, while not complete in describing the code, especially in its newer incarnations, points out some of BATS-R-US' main features. Specifically, BATS-R-US originally solved the MHD equations using a finite volume upwind Roe-type scheme. Currently there are several different solvers available. The computational region in BATS-R-US is made up of logically Cartesian blocks of cells that can be adaptively refined to give higher resolution in a restricted part of the domain. The division of blocks into smaller blocks creates a tree like structure of blocks, where a divided block has eight children (in 3D), and the blocks are connected to other blocks much like the branches of a tree. Finally, BATS-R-US is most commonly run to model the solar wind interaction with solar system bodies.

More recently, BATS-R-US has been extended for high energy density plasma (HEDP) applications. While the code is essentially the same, the applications are drastically different, so the code used for HEDP is named CRASH (Code for Radiative Shock Hydrodynamics) and the corresponding development is supported and carried out by the Center for Radiative Shock Hydrodynamics.

The BATS-R-US (CRASH) code is a first principles hydrodynamic (HD) and magnetohydrodynamic (MHD) model which has been used to simulate the Earth's magnetosphere, the solar convection zone, corona, inner and outer heliospheres, the magnetosphere of most of the planets, several moons and various comets. The code can be extended for use to any problem for which the hydrodynamic and MHD equations are a reasonable physical model. The CRASH code is used to model radiative shocks generated by extremely strong laser pulses in laboratory plasmas.

The BATS-R-US code is the most important building block of the Space Weather Modeling Framework (SWMF). The SWMF executes and couples a number physics models, components as a single model. BATS-R-US is used in multiple roles in SWMF: it can model the Solar and Lower Corona (SC and LC components), the Inner and Outer Heliosphere (IH and OH components) and the the Global Magnetoshere (GM component). A lot of effort was spent on making sure that the very same source code, scripts, test suites and makefiles are used in the stand alone BATS-R-US and in the various components. We tried to change the behavior of the standalone version as little as possible.

This document is aimed at providing the user detailed information about installation, compilaton and execution of the code, and how one can change the physical and numerical parameters to achieve the desired result. The tools provided for visualization are also described.

The physics and the numerics contained in the code are described in the DESIGN document. That document should help the user understand the design philosophy behind the code, the available physics that the code contains and the numerical algorithms that make the code work. This document may not be fully up-to-date regarding software implementation. For more detailed description of the equations and algorithms read the published papers. A review paper describing both the SWMF and BATS-R-US is Toth et al., 2011, Journal of Computational Physics, doi:10.1016/j.jcp.2011.02.006. The algorithms of the CRASH code are described by van der Holst et al., 2011, Astrophysical Journal Supplements, doi:10.1088/0067-0049/194/2/23. Both papers contain numerous references to earlier publications.

1.1 Acknowledgments

BATS-R-US was developed at the University of Michigan starting in 1996 with funding under the NASA High Performance Computing and Communications (HPCC) Earth and Space Sciences (ESS) program (NASA ESS Cooperative Agreement Number: NCCS5-146). Continued work is funded by various grants from NFS, NASA, AFOSR, DoD, and for the CRASH code by DoE. The Center for Space Environment Modeling is lead by Tamas Gombosi, while the Center for Radiative Shock Hydrodynamics is lead by Paul Drake.

Contributions to the development of BATS-R-US fall roughly into three categories: theoretical development, code development and scientific investigations. The principle players and their involvement is as follows:

Principle Investigators and Theory

Tamas Gombosi	1994-	Principle Investigator, MHD theory
Aaron Ridley	2007-	Principal Investigator, magnetospheric physics
Paul Drake	2008-	Principal Investigator, radiative shock theory
Ken Powell	1994-	Co-Principle Investigator, numerics and algorithm development
Quentin Stout	1994-	Co-Principle Investigator, parallel architecture and computer science

BATS-R-US/CRASH Code Development

Darren DeZeeuw	1996-
Hal Marshall	1996-1998
Clinton Groth	1997-1999
Gabor Toth	1999-
Igor Sokolov	2001-
Bart van der Holst	2008-
Lars Daldorff	2010-

Code Development and Science

Aaron Ridley	1999-	Magnetosphere
Gabor Toth	1999-	Magnetosphere, Extended MHD, CRASH
K.C. Hansen	1999-	Planets, Moons, Comets
Chip Manchester	2000-	Eruptive Events, Corona, Helioshpere
Ilia Roussev	2001-2004	Corona, Heliosphere
Igor Sokolov	2001-	Corona, Heliosphere, CRASH
Bart van der Holst	2008-	Eruptive Events, Corona, Heliosphere, CRASH

1.2 Web Pages for BATS-R-US and CRASH

The web page of the Center for Space Environment Modeling can be found at

`http://csem.engin.umich.edu/`

The web page of the Center for Radiative Shock Hydrodynamics is at

`http://aoss-research.engin.umich.edu/crash/`

The SWMF, which includes BATS-R-US/CRASH, is publicly available at

`http://csem.engin.umich.edu/swmf`

Chapter 2

Using BATS-R-US and CRASH

2.1 Quick Start

The installation instructions are described in the README.md file. To keep this user manual more up-to-date and consistent, the README.md file is quoted verbatim below.

```
Copyright (C) 2002 Regents of the University of Michigan,
portions used with permission.
For more information, see http://csem.engin.umich.edu/tools/swmf

This document outlines how to install the stand-alone BATSRUS code
on your system and how to create and access the documentation.
Note that BATSRUS is part of the SWMF. If you already have the SWMF,
you can use BATSRUS inside the SWMF to build the stand-alone BATSRUS code.

To learn more about the BATSRUS, including how to compile and run the code,
please consult the user manual.  To install the BATSRUS and create the
user manual please follow the instructions below.

# Obtain BATSRUS

Get the full source code from GitHub/SWMFsoftware after
uploading your machine's ssh key to github.

The minimum requirement is the 'BATSRUS' repository.

# Getting SWMF from GitHub/SWMFsoftware

Read the
[Git instructions](https://github.com/SWMFsoftware/SWMF/blob/master/doc/Git_instructions.pdf)
about (optional) registration, passwordless access, mail notifications, and
using the [gitclone](https://github.com/SWMFsoftware/share/blob/master/Scripts/gitclone) script.

## Clone the BATSRUS repository

'''
cd {where_you_want_to_have_batsrus}
gitclone BATSRUS
'''
```

```
You may also need the open-source `SWMF_data` repository that contains
large data files in GM/BATSRUS and SC/BATSRUS subdirectories for BATSRUS.
The other subdirectories in SWMF_data can be removed to save disk space.
The `SWMF_data` should be put into the home directory:

'''
cd
gitclone SWMF_data
'''

Some data files used by the Center for Radiative Shock Hydrodynamics (CRASH)
are in the `CRASH_data` repository that is available to registered users.
If needed, it has to be placed into the home directory.

## Clone the CRASH_data repository into the home directory if needed
'''
cd
gitclone CRASH_data
'''

# Install BATSRUS

Many machines used by UofM are already recognized by the
`share/Scripts/Config.pl` script, which is called by the `Config.pl`
scripts in BATSRUS.
For these platform/compiler combinations installation is very simple:
'''
cd BATSRUS
./Config.pl -install
'''
On other platforms the Fortran (and C) compilers should be explicitly given.
To see available choices, type
'''
./Config.pl -compiler
'''
Then install the code with the selected Fortran (and default C) compiler, e.g.
'''
./Config.pl -install -compiler=gfortran
'''
A non-default C compiler can be added after a comma, e.g.
'''
./Config.pl -install -compiler=mpxlf90,mpxlc
'''
For machines with no MPI library, use
'''
./Config.pl -install -nompi -compiler=....
'''
This will only allow serial execution, of course. Like with most scripts
in the SWMF/BATSRUS, type
'''
./Config.pl -help
'''
for a comprehensive description of options and examples.

The ifort compiler (and possibly others too) use the stack for temporary
```

````
arrays, so the stack size should be large. For csh/tcsh add the following
to `.cshrc`:
```
unlimit stacksize
```
For bash/ksh/zsh add the following to `.bashrc` or equivalent initialization
file:
```
ulimit -s unlimited
```

# Create the manuals

Please note that creating the PDF manuals requires that LaTeX
(available through the command line) is installed on your system.

To create the PDF manuals for BATSRUS and CRASH type
```
make PDF
cd util/CRASH/doc/Tex; make PDF
```
in the BATSRUS directory. The manuals will be in the `Doc/` and
`util/CRASH/doc/` directories, and can be accessed by opening
`Doc/index.html` and `util/CRASH/doc/index.html`. Note that
the CRASH application is only usable in the full SWMFsoftware version.

The input parameters of BATSRUS/CRASH are described in the `PARAM.XML`
in the main directory. This is the best source of information when
constructing the input parameter file and it is used to generate the
"Input Parameters" section of the manual.

## Cleaning the documentation
```
cd Doc/Tex
make clean
```
To remove all the created documentation type
```
cd Doc/Tex
make cleanpdf
```
As for most Makefile-s in the SWMF/BATSRUS, type
```
make help
```
for a comprehensive list of make targets and examples.

# Read the manuals

All manuals can be accessed by opening the top index file
```
open Doc/index.html
```
You may also read the PDF files directly with a PDF reader.
The most important document is the user manual in
````

```
```
Doc/USERMANUAL.pdf
```

# Running tests

You can try running the standard test suite by typing
```
make -j test
```
in the main directory. The `-j` flag allows parallel compilation.
This requires a machine where `mpiexec` is available.
The tests run with 2 MPI processors and 2 threads by default.
Successful passing of the test is indicated by empty `*.diff` files.

To run the tests on more (up to 8) cores use
```
make -j test NP=4
```
You can also run an individual test. The list of available SWMF tests
can be listed with
```
make test_help
```
For example, to run `test_shocktube` in a serial mode (without MPI)
but multithreaded with 4 threads (using OpenMP):
```
make -j test_shocktube MPIRUN= NTHREAD=4
```

# Compiling for GPUs

We are working on merging the GPU version of the message passing module with
the CPU version. Currently, you will need a development implementation of
the message passing module to run in parallel on GPUs. Replace the file
before compilation:
```
cp srcBATL/BATL_pass_cell_gpu_parallel.f90 srcBATL/BATL_pass_cell.f90
```
On clusters, load a version of the Nvidia compiler. Some versions may be
unstable. A known stable version is nvhpc/20.7. A newer version that works
on Pleiades is nvhpc/24.3-nompi paired with mpi-hpe/mpt. Switch on the -acc
flag:
```
Config.pl -acc
```
and Compile:
```
make BATSRUS
```
```

2.1.1 Installing the code

In SWMF the components are installed and uninstalled automatically at the same time as the framework is installed. The Makefile.def and Makefile.conf make files in GM/BATSRUS will simply include the files with the same name in

the SWMF directory. The rest of this subsection describes the installation procedure for the stand alone code.

The installation requires the selection of a specific Fortran compiler. For many platforms used by CSEM and CRASH, the script can figure out the default settings all by itself. On these systems, the code can be installed simply by running the command

```
Config.pl -install
```

in the main directory. This

- creates `Makefile.def` with the correct absolute path to the base directory,
- creates `Makefile.conf` that contains the operating system and compiler specific part of the Makefile
- creates the compiler specific `Makefile.RULES` from `Makefile.RULES.all` in the various source directories,
- attempts to create a symbolic link `data` to `SWMF_data/GM/BATSRUS/data`
- attempts to create a symbolic link `dataCRASH` to `CRASH_data`
- executes `make install` which does further installation steps specific to BATS-R-US.

If the platform/machine is not listed at the beginning of the

```
share/Scripts/Config.pl
```

script, or a non-default compiler is desired, then the compiler must be specified explicitly with the `-compiler` flag. Type

```
Config.pl -compiler
```

to see available choices. Then install the code with the selected compiler, for example

```
Config.pl -install -compiler=gfortran
```

On a Linux system this will copy the

```
share/build/Makefile.Linux.gfortran
```

file into Makefile.conf.

In case your platform and/or compiler is not supported yet, you will have to create a

```
share/build/Makefile.OS.COMPILER
```

file where 'OS' is the name of the operating system returned by the Unix command `uname`, while 'COMPILER' is the name of the F90 compiler used or something closely related to that.

To uninstall BATS-R-US type

```
Config.pl -uninstall
```

If the uninstallation fails (this can happen if some makefiles are missing) do reinstallation with

```
Config.pl -install
```

and then try uninstalling the code again. When BATS-R-US is installed, its configuration can be checked with

```
Config.pl
```

with no arguments. To get a complete description of the usage of the `Config.pl` script type

```
Config.pl -h
```

2.1.2 Testing the code

The fastest way of testing the installed code is to run test problems. The full BATS-R-US test suite can be run with

```
make -j test NP=4
```

on a machine that allows both compilation and parallel execution. These tests are typically run on 1 to 8 processors, the default is 2. If only serial execution is available, the tests can be run serially with

```
make test MPIRUN=
```

Running the complete test suite can take anywhere from one to several hours depending on the speed and number of processors, as well as on the speed of the Fortran compiler. The tests are run in the `run_test_*` directories, one after the other.

A test is successful if the results agree with the reference solution to the required accuracy. This is checked by a script, typically share/Scripts/DiffNum.pl, that compares the output with the reference solution. The test passes if the resulting difference file is empty. See Makefile.test for details.

The full set of available tests is shown by

```
make test_help
```

A specific test can be run by typing, for example,

```
make -j test_shocktube
```

This will do a simple shock tube test that takes a few minutes to complete. Like most other tests, this one consists of 4 stages:

1. make -j test_shocktube_compile: configuration and compilation
2. make test_shocktube_rundir: create run directory with input files
3. make test_shocktube_run: run the test
4. make test_shocktube_check: check the results

These stages can be done separately. For example the code can be configured and compiled and the run directory can be created on the head node of a super computer, and then the run phase can be done on a compute node, and the results can be checked on the head node again. The reference solution of the shock tube test is stored in

```
Param/SHOCKTUBE/TestOutput
```

and the difference file will be

```
test_shocktube.diff
```

2.1.3 Configuring BATS-R-US

The first step is selecting the equation and user modules. For example

```
Config.pl -e=Hd -u=Waves
```

will select the hydrodynamics equation module and the 'waves' user module that can perturb variables with various waves, which is useful for tests. This is essentially the same as executing

```
cp srcEquation/ModEquationHd.f90 src/ModEquation.f90
cp srcUser/ModUserWaves.f90 src/ModUser.f90
```

but using Config.pl is simpler and safer, because the previous user module is saved into `src/ModUser.f90.safe` in case it was replaced accidentally. The available equation and user modules can be listed with

```
Config.pl -e -u
```

New equation and user modules can be easily added into the srcEquation/ and srcUser/ directories, respectively.

For some equation modules the number of materials and/or wave groups can be set, for example,

```
Config.pl -e=Crash -nMaterial=3 -nWave=30
```

will select the CRASH user module and set the number of materials to 3 and the number of radiation energy groups to 30.

The adaptive grid parameters can be set, for example, as

```
Config.pl -g=4,4,4 -ng=2
```

This will create 3D grid blocks consisting of $4 \times 4 \times 4$ cells with 2 ghost cell layers (set by -ng). Note that the number of grid blocks should be set in the PARAM.in file with the #GRIDBLOCK and #GRIDBLOCKALL commands. The number of cells per block, however, cannot be changed during run time, which helps the compiler to properly optimize the inner loops on the grid cell indexes.

If the fifth order scheme is to be used, it requires three ghost cells and at least 6 cells in all used dimension:

```
Config.pl -g=6,6,6 -ng=3
```

The grid can also be 2 or even 1 dimensional. For 2D grids the number of cells in the third dimension is 1, for example,

```
Config.pl -g=10,10,1 -ng=3
```

will 10×10 grid cells with 3 ghost cell layers in 2 spatial dimensions. For 1D grids both the second and third numbers are 1, e.g.,

```
Config.pl -g=100,1,1 -ng=2
```

The current grid settings can be inquired with

```
Config.pl -g
```

which shows it in the same compact format. A more verbose and complete configuration information can be obtained with

```
Config.pl -s
```

2.1.4 Compilation options

Check the SWMF test page for the currently used/recommended compiler versions on the various machines. On many systems the `module` command can be used to see the currently loaded compilers (`module list`), the available compiler versions (`module avail`) and to unload old and load new versions (`module unload SOMETHING` and `module load SOMETHINGELSE`).

The optimization level can be set, for example, with

```
Config.pl -O2
```

For debugging it is best to use

```
Config.pl -O0 -debug
```

The NAG Fortran compiler is especially powerful for debugging, as it can check for uninitialized real variables and it provides line numbers for any failure of the code. In some cases the MPI library does not allow the error messages to show up on the terminal. If the problem can be reproduced with a serial run, it is best to set

```
Config.pl -O0 -debug -nompi -noopenmp
```

The -nompi flag will compile the util/NOMPI library and modifies Makefile.conf so that the code is linked with the NOMPI library instead of the real MPI library. The code compiled with the NOMPI library can only be run in serial mode, however, debugging can become much easier. The -noopenmp flag disables compiling the OpenMP directives and the use of the OpenMP library for multi-threaded runs.

To undo these settings, for production runs, use

```
make clean
Config.pl -O4 -nodebug -mpi -openmp
```

Note that make clean is necessary when the optimization level or debugging flags are changed. If only the NOMPI is replaced with the usual MPI library (or the other way around), it is sufficient to delete the executable, e.g. src/BATSRUS.exe, and simply relink the code using make.

The code can be compiled with single precision using Config.pl -single, but this is neither recommended nor supported. On 64-bit platform the default double precision executes almost as fast as the single precision, and the enhanced arithmetic accuracy is required by many algorithms. The single precision option wiil likely be removed in the future.

2.1.5 Compilation

Once the compilation options are set, compile BATS-R-US with

```
make -j
```

in the main directory. This will produce the executable src/BATSRUS.exe. The CRASH code, which is BATS-R-US compiled together with the util/CRASH library, can be created with

```
make -j CRASH
```

and the executable will be src/CRASH.exe.

For post-processing IDL output, the executable src/PostIDL.exe have to be compiled with

```
make PIDL
```

The multiple snapshots included in a single IDL output file can be manipulated with the src/SNAPSHOT.exe code, which can be compiled with

```
make SNAPSHOT
```

2.1.6 Creating a run directory

The next step is to create a run directory with

```
make rundir
```

This will create a directory named run/. This can be renamed or even moved to another disk (e.g. the scratch disk of a super computer), if necessary. The name of the run directory can also be specified like this

```
make rundir RUNDIR=/scratch1/MYSELF/run15
```

Note that an absolute path is used. In some cases it is useful to create a run directory that mimics the use of BATS-R-US as some specific component of the SWMF. For example, for the solar corona component use

```
make rundir COMPONENT=SC
```

This will create a run/SC subdirectory instead of the usual run/GM subdirectory.

The freshly created run directory contains several files, symbolic links, and directories. The most important one is a link to the executable BATSRUS.exe. If the run directory is created with

```
make rundir DEFAULT_EXE=CRASH.exe
```

then it will contain a link to the CRASH.exe code. There are also links to the PostIDL.exe and PostSPH.exe codes used for post-processing the output. It is often a good idea to remove the links and copy the executables into the run directory. This allows recompiling the code without having a possibly unwanted effect on the run directory, or a submitted run that is still in the job queue.

The IO2 subdirectory will contain the output files, while the restartIN and restartOUT subdirectories are used for input and output restart files, respectively. The PostProc.pl, pIDL, pTEC, and Restart.pl scripts are copied into the run directory to allow processing the output files and restarting the code. The link to the PostIDL.exe code is for postprocessing.

If the machine name (disregarding numbers at the end of the name) matches one of the job files in

```
share/JobScripts
```

then the job file will be copied into the run directory. For example, on `pfe8` (one of the head nodes of Pleiades), the `job.pfe` job script file and the `qsub.pfe.pl` and `watch.pfe.pl` scripts will be copied. The job script file serves as a template only. It has to be edited and modified before submitting a job. On Pleiades the `qsub.pfe.pl` script can be used to submit jobs for multiple node types, see the help message (-h) for details.

2.1.7 Creating input files

The run directory also contains a ”default” PARAM.in file, but this should always be replaced with another file, in practice. The link to the BATSRUS/Param directory is provided, so that the files in this directory can be copied into PARAM.in as a starting point. Some of the files contain small segments of the input file, e.g. the grid description for some application. These segments can also be included into PARAM.in with the `#INCLUDE` command.

It is important to check the correctness of the input file, especially before submitting large jobs on a super computer. In the main BATS-R-US directory type

```
CheckParam.pl
```

which will check `run/PARAM.in`. The number of processors to be used and the name of the input parameter file can also be specified, for example

```
CheckParam.pl -n=256 run_large/PARAM.in
```

Another way to construct and check the PARAM.in file is to run the parameter editor (this can be done best on your local desktop or laptop machine):

```
share/Scripts/ParamEditor.pl
```

This will open `run/PARAM.in` in a web browser with a user interface. The name of the parameter file can also be specified, e.g.,

```
share/Scripts/ParamEditor.pl run_large/PARAM.in
```

The parameter editor provides an integrated user interface with manual, editing, and checking options.

The PARAM.in file is always required for a run. Several applications read additional files, for example for initialization, boundary condition, satellite trajectories to extract data, and in case of a restarted run, restart files. These all have to be in place before the code is run. The parameter checking also checks for the existence of many of these input files.

2.1.8 Running the code

Now you are ready to run the code. For an interactive run on 8 processors with 4 threads (if the code was compiled with an MPI library and OpenMP enabled) type

```
cd run
export OMP_NUM_THREADS=4; mpiexec -np 8 BATSRUS.exe # bash, ksh, zsh
setenv OMP_NUM_THREADS 4; mpiexec -np 8 BATSRUS.exe # csh, tcsh
```

It is often a good idea to save the output and error messages into a file, both for runs done interactively and in the background:

```
mpiexec -np 8 BATSRUS.exe |& tee runlog
mpiexec -np 8 BATSRUS.exe >& runlog &
```

Using `runlog` as the filename allows the `PostProc.pl` script to find it and collect it together with other output files.

On many machines you have to submit the run to a batch queue. First edit the appropriate job script file and then submit the job, for example on Pleiades

```
cd run
emacs job.pfe
./qsub.pfe.pl job.pfe Run8
```

The content of the job scripts and submission commands depend on the queuing system.

2.1.9 Restarting the run

There are several reasons for restarting a run

- a run may be too long for a single job submission,
- a run may fail for some reason, and it can be restarted with different parameters,
- a parameter study is done with varied parameters starting from the same state,
- debugging some problem that happens a few steps after the restart.

The `Restart.pl` script makes it quite easy to restart a run. Assuming that there are restart files saved, simply type

```
Restart.pl
```

in the run directory. This will move the files from the `restartOUT` directory into a restart directory tree, and the files in the restart directory tree will be linked to the `restartIN` directory. The default name of the restart directory tree is based on the simulation time for time accurate runs, or the number of iterations for steady state runs. For example, if the restart files were saved at 6 hours of simulation time, the restart directory will be named

```
RESTART_t006.00h
```

The name of the restart tree can also be specified, e.g.,

```
Restart.pl RESTART_debug
```

It is also possible to select an already existing tree, e.g. something that was saved by PostProc.pl (see next subsection):

```
Restart.pl -i NewResults/RESTART
```

By default the restart files overwrite each other, however, the `Restart.pl` script can create multiple trees while the code is running, by simply checking if there are new restart files in restartOUT and moving those into a tree. For example

```
Restart.pl -r=600 -t=h &
```

will check for new restart files every 600 seconds (wall clock time), and the restart trees will be named using hours as time units. To read about more options, type

```
Restart.pl -h
```

2.1.10 Postprocessing

After the code has successfully run (possibly with multiple restarts) the output files found in `run/IO2` need some post processing before they can be visualized. The most convenient way to do this is to run

```
PostProc.pl
```

in the run directory. This will process both IDL and Tecplot output files. For long runs it is a good idea to periodically post process the output, e.g.,

```
PostProc.pl -r=600 >& PostProc.log &
```

will post process the output every 600 seconds, and the various output (and possibly error) messages will be collected into the `PostProc.log` file.

When the run is finished, the IDL output files can be merged into a single "movie" file with the `-M` switch, while the logfiles and satellite output files from multiple restarts can be concatenated using the `-cat` swithc:

```
PostProc.pl -M -cat
```

When the run is complete, it is a very good idea to create an output directory tree that contains the `PARAM.in` file, the `runlog` file, the plot files as well as restart files together. This can be done by providing the name of the directory tree as an argument:

```
PostProc.pl -M -cat RESULTS/latest
```

When there are a huge number of output files to process, the post processing can become slow. The post processing of the IDL files can be done in parallel on multicore machines using the `-n` switch

```
PostProc.pl -n=4
```

This can also be combined with the `-r` switch for continuous post-processing when the serial post processing cannot keep up with the amount of data produced by the running job. The complete set of options can be obtained with

```
PostProc.pl -h
```

There are several lower level scripts that can be used directly if desired. For example the IDL output files can be processed with

```
pIDL
```

This command will produce the new `run/IO2/*.out` files, and delete the `run/IO2/*.idl` files and the corresponding header files `run/IO2/*.h`. In case you want to keep the raw data, e.g. to be safe, type

```
pIDL -k
```

For complete usage information type `pIDL -h`.

The Tecplot files are processed with the `pTEC` script. See `pTEC -h` for options.

Visualization of the resulting output files with IDL and TecPlot will be described in detail in sections **??** and **??**, respectively.

2.1.11 Recompilation and clean up

If you change some source files, e.g. the user module `src/ModUser.f90`, the code can be simply recompiled with the `make` command. However, if the compiler flags or the precision are changed, you need to clean the object files with

```
make clean
```

before recompilation. This will remove all the object files and forces recompilation.

To delete all files created during installation and compilation, use

```
Config.pl -uninstall
```

In addition to the object files, this command will remove the executables `src/*.exe`, the libraries `src*/*.a` and all other temporary files.

For a complete list of possible make targets type

```
make help
```

2.2 Hardware and Software Requirements

2.2.1 Source Code and Compilers

BATS-R-US is written completely in standard Fortran 90 with calls to standard MPI libraries. You must assure that both of these are available in order to use BATS-R-US. Compilers seem to be readily available for F90, although there are very few free F90 compilers. Most parallel machines have MPI libraries in residence, although most Linux distributions do not include F90 versions of the MPI libraries. The code has been run with the free distribution of the MPI-CH libraries under Linux. BATS-R-US has been run successfully on the following platforms: Cray T3E, IBM SP, SGI Origin, Altix, Darwin (Mac OS) and several different Beowulf clusters.

If the user wishes to run BATS-R-US on a single node, no MPI libraries are required to be in residence. The user must make the `NOMPI` library which is included in the util/NOMPI utility (see section 2.3.3)

2.2.2 Parallelism, Speed Up and Scaling

While it is designed to run on massively parallel platforms, the code will run on any number of nodes from a single processor to as many as the user has access to. We have run BATS-R-US on up to half a million CPU cores using a hybrid MPI + OpenMP approach. As described in the DESIGN document, blocks are distributed on different processors and must communicate with each other through message passing. The communication time between processors, or latency, will have an effect on the efficiency of the code.

Two ways to measure a code's performance are to look at "speed up" or strong scaling and "weak scaling". Speed up is measured by taking a fixed size problem and running it on more processors. If the code runs twice as fast when run on twice as many processors then the code has perfect speed up. Communication time is likely a major part of the work load if more processors does not mean more speed. Scalability is measured by doubling the size of the problem and the number of processors at the same time and asking if running the code takes the same amount of time as before. Both are important measures of code performance.

2.2.3 Memory and Disk Requirements

BATS-R-US is a block based simulation code (see the DESIGN document) and as such, memory requirements are most easily discussed in terms of the number of blocks in a simulation.

To establish the computation requirements needed to run the code, the user must determine how large of a simulation region and what resolution is needed for that run. This will establish the total number of simulation blocks needed, which will in turn determine the memory requirements of the system. Table 2.1 gives examples of a number

Table 2.1: Examples of memory requirements for some commonly run simulation types.

Simulation	Number of Bytes for a Real Number	Minimum Number of cells	Maximum Number of cells	Total Memory for Maximum
Saturn	8	600,000	1,000,000	6.6 GB
Heliosphere	8	500,000	2,000,000	13.2 GB
Earth	8	250,000	1,000,000	6.6 GB
Earth	4	250,000	1,000,000	3.3 GB

of different simulations which are commonly run, and their memory requirements. Note that the memory requirement depends on the size of real numbers used at compilation time. This can be changed by changing compile flags in the makefiles. For a 4 byte real, each 4x4x4 cell block takes approximately 0.2 MB of memory. For an 8 byte real, each 4x4x4 cell block takes approximately 0.4 MB of memory. The trade-off between the two are discussed in section 2.3.4.

For example, simulations of the Earth's magnetosphere are typically run with between 250,000 cells and 1,000,000 cells and with a 4x4x4 grid structure within each block. This means that the simulations are run with between 4000 and 16000 blocks. Using a 4 byte real number, each block takes approximately 0.2 MB of memory. Therefore the simulations would require between 850 and 3300 MB of memory. This memory can obviously be distributed across all of the CPUs which are available to the run. So, if these runs were conducted on a 16 node computer, each node would have to have approximately 200 MB of memory (for the large run). On a different computer, the memory per node would scale accordingly.

The amount of disk space required to run the code varies dramatically between different types of simulations, time and spatial resolution required, and the number of variables to be saved for plotting. Specifically, users need to be aware of the size of restart files and the size of output files for plotting.

Restart files, used to restart the code from the middle of a simulation, store the entire three dimensional cell centered data used by the code. Because the code has other overhead, this number is smaller than the memory needed when actually running the code. The restart files are roughly one tenth (1/10) the size of the code image in memory.

Output files for plotting vary greatly in size depending on the nature of the output and the frequency of the output. Full 3-D output files of the entire simulation domain, not yet post processed, can be a factor of 2 or more larger than the restart files because they contain more variables and are not binary. One common method for saving disk space is to only write out cut planes and not the full 3-D simulation region. This has the advantage of saving vast amounts of disk space, but reduces the amount of physical insight which can be gained from the simulation. Typical (binary) cut plane files are approximately 1.0 MB once they are post processed. The biggest influence on the amount of memory required is the frequency at which plot files are written during a run. Clearly, if data is output often (say every 1-30 seconds of simulation time of a 10 hour simulation), the amount of data can grow astronomically.

2.3 Installation and Compilation

2.3.1 Directory Structure

Upon installing BATS-R-US, you should find the following files and directories inside the main BATS-R-US directory. The subdirectories are included without listing all of the files that they contain.

```
File or Directory:          Notes:

Config.pl                   Script to (un)install and configure the code
Configure.pl                Script to configure the source code
Configure.options           Default configuration options
Doc/                        Documentation
   Tex/                        Latex source code for documentation
   HTML/                       HTML form of the documentation
Idl/                        Source code for IDL post processing
```

```
Makefile                  Main makefile
Makefile.def              Place holder for the configured Makefile.def
PARAM.XML                 Description of input parameters in text and XML
PARAM.pl                  PARAM.XML converted to a Perl file
Param/                    Sample PARAM.in files, include files
Scripts/                  Contains scripts to run the code on
   ConvertRestart.pl         Script to convert the endianness of restart files
   IDL/                      Scripts used to process IDL files
   Run/                      Scripts used to run BATSRUS and process output
      AIX/                      -IBM SP
      Darwin/                   -Macintosh
      IRIX/                     -SGI Origin
      IRIX64/                   -SGI Origin
      Linux/                    -Linux (Beowulf clusters, SGI Altix)
      OSF1/                     -Compaq
   TEC/                      Scripts used to process TECPLOT files
share/                    Scripts and source code shared with SWMF
src/                      Source code for BATSRUS
srcInterface/             Source code for interfaces with other components
srcPostProc/              Post processing source code
TestBatsrus.pl            Script to test BATSRUS
TestCompare.pl            Script to compare test results
TestCovariant             Script to test BATSRUS in 'covariant' configuration
TestParam.pl              Script to test the input parameter file
TestSuite.pl              Script to run a whole test suite
util/                     Utilities NOMPI and TIMING shared with SWMF
```

2.3.2 Setting Grid Structure Before Compiling BATS-R-US

Part of the design of BATS-R-US is the conscious decision to limit the usage of allocateable variables. This is because allocateable memory was found to slow the code considerably and the choice was made to maximize code speed. This means that the most of the arrays of BATS-R-US must be dimensioned with a fixed size in the source code. Since platforms vary widely, there is no way to set these in a universal way and the user will have to set them. There are two main sets of parameters that must be changed.

Number of Blocks per Processor: `nBLK`

As described in the DESIGN document, the main unit of mesh used for computation in BATS-R-US is the Cartesian block. The major part of memory is dimensioned in this block structure. Therefore, the number of blocks basically determines the amount of memory the code will use. The number of blocks that any given processor can handle depends on the amount of memory that each node has available and must be changed in the file `src/ModSize.f90`. This is normally done with the `Config.pl` script. The maximum number of blocks is defined by the variable `nBLK`.

Number of Cells in Each Block: `nI`, `nJ`, `nK`

Typically, BATS-R-US is run with blocks that are cubes, but this does not have to be the case. The size of blocks is determined in `ModSize.f90` by the parameters `nI`, `nJ`, `nK`, the numbers of cells in a block in each dimension. This number does not include ghost cells which the code takes care of automatically. The user should set these numbers with the Config.pl script according to their needs based on the following restrictions.

- `nI,nJ,nK` variables should not be less than 4 and must be set as even integers (4, 6, 8, 10 ...). If the grid is uniform (all cells the same size) the code should run with `nI,nJ,nK` set to 2.

- Smaller blocks means a larger ratio of ghost cells to computational cells. For example a 4x4x4 block has a 64 cell computation region, while it has 8x8x8 = 512 total cells counting ghost cells. In other words, the majority of the storage is ghost cells (#Ghost/#Computation = 7). If the user used 8x8x8 blocks, the computation region would have 512 cells while the total number of cells is 1728 and the ratio of ghost cells to computation cells is only 2.4.
- The time the code spends message passing depends on the number of ghost cells in a block and on the number of blocks. With larger blocks the message passing time may be reduced.
- Larger blocks mean more wasted cells when doing AMR to resolve features of the solution. If the user uses 4x4x4 block he or she will have much better control over where the resolution is located. With larger blocks the user will have to resolve a larger area to get the interesting area resolved.

The user should note that as usual, in numerical simulations there is a trade off in efficiency in resolving the solution and the amount of storage. In general, larger blocks mean less "wasted" storage but will inevitably lead to more cells to resolve the same features.

Getting the Grid you Want

The initial grid (before any refinement) contains a number of blocks determined by the `proc_dims(i)` variable, where `i` is the direction (1=x, 2=y, 3=z). If the values are set to

```
proc_dims(1) = 2
proc_dims(2) = 1
proc_dims(3) = 1
```

then the top level blocks form a brick consisting of 2 blocks arranged along the x axis. This variable is called `proc_dims` because in the original design of BATS-R-US the initial number of blocks could not exceed the number of processors. In the current version there is no such restriction, and the name is kept for historical reasons only. A more descriptive name would be `nRootBlock_D` which refers to the number of root blocks in the 3 coordinate directions.

The sizes of cells in these top level blocks is determined by the initial number of blocks and also by the physical size of the computational domain and the values of `nI`, `nJ`, `nK`. As an example, we show a section of the input parameters used when running the code

```
#GRID
4                       nRootBlockX
2                       nRootBlockY
1                       nRootBlockZ
-32.                    xMin
 32.                    xMax
-8.                     yMin
 8.                     yMax
 0.                     zMin
 4.                     zMax
```

These input parameters, along with `nI,nJ,nK` define the initial grid used in the simulation. The values xMin, xMax, ... indicate the physical domain of the computation. In section 2.3.2 we indicated that the number of cells in a block in each direction is determined in `ModSize.f90`. Here we determine the actual shape of the computational domain as well as the actual shape of each cell. Typically, BATS-R-US is run with cells and blocks that are cubes. While this is not required, cells that have large aspect ratios may lead to less accurate computations. The shape of a cell is determined by the `nI,nJ,nK`, the `proc_dims` and the physical size of each dimension. The size of a cell in the top level blocks is given by

$$dx = \frac{\text{xMax} - \text{xMin}}{\text{nRootBlockX} * \text{nI}} \qquad dy = \frac{\text{yMax} - \text{yMin}}{\text{nRootBlockY} * \text{nJ}} \qquad dz = \frac{\text{zMax} - \text{zMin}}{\text{nRootBlockZ} * \text{nK}} \tag{2.1}$$

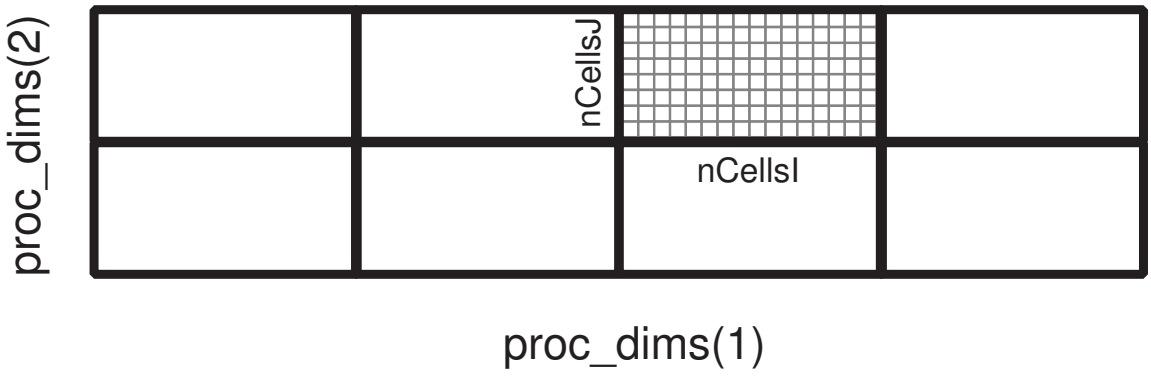

Figure 2.1: Initial block and grid structure for `proc_dims` of 4x2x1 and `nI,nJ,nK` of 16x8x4. The z dimension is not shown. Heavy line indicate blocks, lighter lines indicate individual cells.

We give three examples. First, with 4x4x4 blocks, if the physical dimension of the computation region is a cube, then setting the initial number of blocks in each dimension (`proc_dims`) equal will ensure cells that are cubes. For the second example, if one dimension is twice as large as the other two, then beginning with twice as many blocks in this dimension will again ensure cubic cells. Finally, combining xMin, xMax, yMin, yMax, zMin, zMax, nI, nJ, nK and proc_dims allows the user to make computation regions that are highly stretched in one dimension while still having cubic cells. If the user wanted to do a two dimensional problem, for example, he or she could choose parameters to minimize the numbers of cells in 1 dimension. Figure 2.1 shows a 2 dimensional view of a grid defined using the input shown above and with `nI` = 16, `nJ` = 8, `nK = 4`. In the figure, the dark lines show the locations of blocks, initially 4x2x1 (the z dimension is not shown). This grid would have to be run on at least 8 processors. The The lighter lines show the individual cells in one of the blocks. These are 16x8x4 (again, the z dimension is not shown). Notice that the cells are cubes (dx=dy=dz=1.0), but the computation region is stretched in the x and y directions.

THERE IS ONE OTHER RESTRICTION ON THE WAY THAT GRIDS CAN BE CREATED WHEN USING A CENTRAL BODY. Because the first body is created with its center at the origin, it is important that eight block corners meet at the origin. The user must think ahead so that the final desired refinement level at the body satisfies this restriction. While the code may run if this is not that case, Tecplot output will not be correct, initial refinement may not do what you thought it should and there may be other problems. We give parameters for the standard Earth case as an example

```
#GRID
2                       nRootBlockX
1                       nRootBlockY
1                       nRootBlockZ
-224.                   xMin
 32.                    xMax
-64.                    yMin
 64.                    yMax
 64.                    zMin
 64.                    zMax
```

Initially there are two blocks which meet at x=-96. The origin does not lie at block corners. Two refinements of these initial blocks will create 128 blocks and will give the required corners at the origin. Since this case is never run with a lower resolution than this we are okay.

2.3.3 The Main Makefile

The main executable and all the post processing executables along with run directories can all be set up from the main makefile. This is true for the framework and the stand alone code as well, and the makefile targets are very similar in both cases. Typing

```
make help
```

lists the available executables, directories and scripts which can be built or executed. The rest of this subsection is specifically about the stand alone code, i.e. the use of the Makefile in the BATS-R-US main directory. In stand alone mode the help list is the following:

```
You can ``make'' the following:

  <default> BATSRUS in stand alone mode, help in SWMF

  install   (create MAKEFILES, src/ModSize.f90, src/mpif90.h)
  MAKEFILE_DEF (create/correct Makefile.def)

  LIB     (Component library libGM for SWMF)
  BATSRUS (Block Adaptive Tree Solar-Wind Roe Upwind Scheme)
  NOMPI   (NOMPI library for compilation without MPI)
  PIDL    (PostIDL program creates 1 .out file from local .idl files)
  PSPH    (PostSPH program creates spherical tec file from sph*.tec files)

  help      (makefile option list)
  clean     (rm -f *~ *.o *.kmo *.mod *.T *.lst core)
  distclean (make clean; rm -f *exe Makefile Makefile.DEPEND)
  dist      (create source distribution tar file)

  PDF       (Make PDF version of the documentation)
  HTML      (Make HTML version of the documentation)

  rundir      (create run directory for standalone or SWMF)

  mpirun      (make BATSRUS and mpirun BATSRUS.exe on 8 PEs)
  mpirun NP=7 (make BATSRUS and mpirun BATSRUS.exe on 7 PEs)
  mprun NP=5  (make BATSRUS and mprun  BATSRUS.exe on 5 PEs)

  spherical_src  (Make SPHERICAL directory with BATSRUS on spherical grid)
  spherical_conf (Make SPHERICAL directory and link it to BATSRUS_conf)
  covariant_src  (Make COVARIANT directory with BATSRUS on covariant grid)
  corelax_src    (Make CORELAX directory for the covariant version
              of the magnetogram-driven solar wind)
  cartesian_src  (removes source code for covariant grid)
  relax_src      (Make RELAX directory for the Cartesian
                  version of the magnetogram-driven solar wind)
```

As an example, in order to make `BATSRUS.exe`, simply type

```
make
```

To make a distribution (tar file) of the code like the one used for installation, type

```
make dist
```

To make a directory with everything setup to run code

```
make rundir
```

The executables reside in the `src` directory. When a run directory is created it is located in the root directory of the installation and will have links to the executables, will have copies of the appropriate scripts for the current platform and will have the directory structure for running the code setup.

The Makefile in the root directory of the distribution calls the different makefiles in the `src/`, `srcPostProc/`, `share/` and `util/` directories in order to do the actual compilations.

2.3.4 Compiler Flags

The makefile `Makefile.conf` in the main directory (of SWMF or the stand alone BATS-R-US) contains the system dependent flags. In this file, the user will find several sets of commented out flags. The default version will be the one that should be used for running the code to do production type runs. Other sets of flags include those for debugging (which does more error checking), among others. As mentioned above, compile flags are not trivial to set and if the user has to change them, they will likely have to consult the manual pages of the F90 compiler.

One main flag, **PRECISION** determins the number of bytes a real number occupies. On most systems the default size real is 4 bytes. In the Makefiles, the flag **PRECISION** can be changed to indicate the number of bytes to use for a real. When debugging it is often necessary to use 8 byte real to differentiate round off error from true errors. For production runs, 8 byte reals should be more accurate, but may not always be necessary. Using 8 byte real will double the size of the code. The trade off is of course loss of some accuracy with 4 byte reals and increased code and restart file size for 8 byte reals. In addition, on 32 bit machines (such as most PC's and Suns), the code will run slower (about 30 % on our PC based Beowulf) when using 8 byte reals.

Another compiler flag that the user may need to change is the **MPILIB** flag. As mentioned in section 2.2.1, the code can be run on a single processor with our `NOMPI` library, which the user can make (see section 2.3.3). To run with this library the **MPILIB** needs to be changed from calling the system libraries to the NOMPI library. As an example, the `Makefile.IRIX64`, is the makefile for the SGI Origin. In this makefile the user will find the following lines:

```
MPILIB = -lmpi
#MPILIB = -L. -lNOMPI
```

To use the `NOMPI` library, the first line must be commented out (place a # in front of the line) and the # should be removed from the second line. With the `NOMPI` library the code can run on a single processor only.

2.3.5 Run Directory Structure

The `run` directory is setup with `make rundir`. The purpose for creating run directories is to separate runs. You can rename them

```
mv run run_CME
```

to do a simulation of a CME, for example. You can also move the `run` directory to another directory or disk, but this is not recommended, because it will be difficult to relate the source code and the actual run.

When /BATSRUS/ is used inside the framework, it gets a subdirectory named by the component. For example if /BATSRUS/ is used as the SC component, it will read files from and write files into the

```
run/SC
```

directory. The following directories links and files are created in `run/SC`:

```
IO2/
restartIN/
restartOUT/
PostIDL.exe -> /home/USER/SWMFDIR/bin/PostIDL.exe
PostSPH.exe -> /home/USER/SWMFDIR/bin/PostSPH.exe
pIDL
pTEC
```

In the stand alone mode of BATS-R-US the subdirectory name is always GM (independent of the value of the STANDALONE variable in Makefile.def). To facilitate access to the subdirectories and scripts, a symbolic link is provided in the `run` directory to all the items in `run/GM`.

In SWMF the run directory itself contains all the component subdirectories, and the following files, links and subdirectories

```
core
job.MACHINE
LAYOUT.in
Param       -> ../Param
PARAM.in
SWMF.exe    -> /home/USER/SWMFDIR/bin/SWMF.exe
STDOUT/
```

In the stand alone BATS-R-US, other than the links to the items in the run/GM subdirectory, the following files and links are created

```
BATSRUS.exe -> /home/USER/bats/src/BATSRUS.exe
core
job.MACHINE
PARAM.in
```

The 'core' file is there to prohibit the creation a huge core file due to a run time error. The job.MACHINE (if any) file contains an example script to submit a job on the given MACHINE. The PARAM.in file is copied from Param/PARAM.DEFAULT and should be normally modified or replaced before running the code.

This directory is now setup to run the code. Links have been made to `BATSRUS.exe` which is the main MHD executable, as well as the IDL post processing executable `PostIDL.exe` and the Tecplot post processing executable `ProcessIO.exe`. The script `pTEC` is for processing the Tecplot files while `pIDL` is for processing the IDL plot files. The scripts `job.XXXXX` contain sample scripts for submitting jobs to the various machines that we have run the code on.

The `run` directory contains subdirectories or links to directories such as `restartOUT/`, `IO2/`, and `Param/`. The first three are needed to run the code, while the fourth contains input files which can be included from the `PARAM.in` file, as described below. Normally one needs at least the `IO2/` and `restartOUT` directories to store plot files and restart files. These are the default names, which can be changed in the PARAM.in together with the directory names. If a needed subdirectory is missing, the code will stop with an error message.

`IO2/` is the most important directory, since it contains output of the MHD run. The `restartOUT/` directory is where restart files are written. These files allow the code to be restarted if it either crashes or if the code must be run on a queue system in which the run takes longer than the length of the run time allowed on the machine. For example, on some machines the codes are allowed to run for two hours. Since most time accurate runs take much longer than this, restart files can be saved near the end of the 2 hours and the code can be restarted after waiting in the queue again.

The `restartIN/` directory is needed to actually restart the run from a previous run. Often the `restartIN/` directory is simply a link to the `restartOUT/` directory. This, however, can be dangerous because if something "bad" occurs during the writing of the restart files, the last save may be destroyed. It is wiser to move the `restartOUT/` directory to some other location, link the `restartIN/` directory to this directory, and create a new `restartOUT/` directory. This will ensure that the old save will be secure, while restarting correctly.

The PARAM.in file contains the input parameters for the BATS-R-US code. The details about this file and the input parameters are given in section 3.

2.4 Running a Simulation

This section describes how to run the stand alone BATS-R-US. To run BATS-R-US as part of the framework, read the SWMF manual.

As described above, the makefile will help the user to set up a directory in which to run the main executable. This directory will have links made to executables, will have the correct directories for input and output and will copy the appropriate scripts (if any) for the current machine. Unfortunately, systems for running the code vary widely. For example the NASA Ames SGI Origin and the NCSA SGI Origin have different queuing systems and therefore require different scripts to run the code. If the user is running the code on a system which is different than the ones listed in section 2.3.3, he or she will have to create the appropriate scripts and make the corresponding changes to the makefiles.

2.4.1 Before Running the Code

Before submitting a job or running the code interactively there are several necessary steps.

- Use `make` to make the `BATSRUS.exe` executable.
- Use `make rundir` to make a run directory.
- Edit the `PARAM.in` file in the `run` directory to contain all the important code input (see chapter 3).
- Check the `PARAM.in` file with the TestParam.pl script.
- Prepare the proper job script for the users system.

To check the parameters in the run/PARAM.in file type

```
./TestParam.pl
```

To check another parameter file named run/PARAM.in.other, type

```
./TestParam.pl run/PARAM.in.other
```

This script reads in the input parameter file and all the included files and checks that they conform with the XML description given in the PARAM.XML (or the PARAM.pl file on systems which do not have the XML-PARSER::EasyTree Perl package installed). Error and warnings are printed on the screen. If there are no errors the script runs silently.

Before submitting a long job into the queue, it is generally a good idea to test the code interactively or in a short queue to make sure that it is working as planned.

2.4.2 Interactive Execution

On most systems, running the code for an extended period of time on many nodes must happen through a queuing system. Test runs, however, can often be done interactively. Although there is no universal form for running codes under MPI, on a number of platforms (e.g. SGI or IRIX and Linux) MPI executables can be run with the

```
mpirun -np N BATSRUS.exe
```

command, where `N` is the number of processors to run the code on. On the Cray T3E the command would be

```
mpprun -n N BATSRUS.exe
```

and for the IBM SP

```
BATSRUS.exe -procs N
```

For any given system, there will be a limit on the value of `N` that is allowed in the interactive queue. **Note that the above two examples may not work on some systems.**

Many platforms require additional information in a script such as the length of time the run will last, the memory usage, names of error files, and many others. To run the code, the user typically needs to prepare a script that meets the system requirements and then submit this script to the queuing system in the appropriate way.

2.4.3 Queues and Scripts

On most platforms, the code must be run through a queue system because there are many users that share the resources on a machine. The `Scripts/Run` directory contains scripts for running BATS-R-US on the currently recognized platforms listed in section 2.3.3 at specific institutions. For example, the `IRIX64` subdirectory contains scripts to run on an SGI Origin on machines at The University of Michigan, NASA Ames and NCSA. Even though the user's system is an SGI Origin, the available scripts may not work for the user's queuing system. The user will have to develop scripts for his or her specific machine. These should be placed in the appropriate subdirectory of `Scripts`. If either your queuing system or your platform is not recognized you will have to create a subdirectory and scripts. The Makefile in the root of the distribution copies files out of this directory when creating run directories. See section 2.3.3 to get more details on the steps to take for a new platform.

Sample scripts for all of the platforms and queuing systems that the code is commonly run are found in the `Scripts` directory and are copied into the run directory.

Chapter 3

Input Parameters

3.1 PARAM.in

The input parameters for the BATS-R-US code are read from the `PARAM.in` file which must be located in the run directory. The file controls all of the BATS-R-US functionality. When BATS-R-US runs as part of the SWMF, the parameters for the component represented by BATS-R-US (for example GM) are given between the

```
#BEGIN_COMP GM
...
#END_COMP GM
```

commands. We refer to the lines starting with a # character as commands.

There are several features of the input parameter file syntax that allow the user to easily run the code in a variety of modes while at the same time being able to keep a library of useful parameter files that can be used again and again.

The user should be aware of and become intimately attached to the `PARAM.XML` file located in the main BATS-R-US directory. This file contains the most detailed description and complete list of the all the input parameters used by BATS-R-US. The same file is used to produce much of this manual with the aid of the share/Scripts/XmlToTex.pl script. The `TestParam.pl` script also uses the `PARAM.XML` file to check the PARAM.in file. Copying small segments of the `PARAM.XML` file into `PARAM.in` can speed up the creation or modification of a parameter file.

If the command string

```
#END
```

is present, it indicates the end of the run and lines following this command are ignored. If the #END command is not present, the end of the file indicates the end of the run.

3.2 Included Files, `#INCLUDE`

The `PARAM.in` file can include other parameter files with the command

```
#INCLUDE
include_parameter_filename
```

The include files serve two purposes: (i) they help to group the parameters; (ii) the included files can be reused for other parameter files. An include file can include another file itself. Up to 10 include files can be nested. The include files have exactly the same structure as `PARAM.in`. The only difference is that the

```
#END
```

command in an included file means only the end of the include file, and not the end of the run, as it does in `PARAM.in`.

The user can place his/her included parameter files into the main run directory or in any subdirectory as long as the correct path to the file from the run directory is included in the `#INCLUDE` command. There are many include files in the `Param` directory. These can be included into the `PARAM.in` files, or they can serve as examples.

3.3 Commands, Parameters, and Comments

As can be seen from the above examples, parameters are entered with a combination of a **command** followed by specific **parameter(s)**, if any. The **command** must start with a hashmark (#), which is followed by capital letters and underscores without space in between. Any characters behind the first space or TAB character are ignored (the #BEGIN_COMP and #END_COMP commands are the only exception, but these are markers rather than commands). The parameters, which follow, must conform to requirements of the command. They can be of four types: logical, integer, real, or character string. Logical parameters can be entered as `.true.` or `.false.` or simply `T` or `F`. Integers and reals can be in any of the usual Fortran formats. All these can be followed by arbitrary comments, typically separated by space or TAB characters. In case of the character type input parameters (which may contain spaces themselves), the comments must be separated by a TAB or by at least 3 consecutive space characters. Comments can be freely put anywhere between two commands as long as they don't start with a hashmark.

Here are some examples of valid commands, parameters, and comments:

```
#TIMEACCURATE
F                       DoTimeAccurate

Here is a comment between two commands...

#INNERBOUNDARY
ionosphereB0            TypeBcInner (3 spaces or TAB before the comment)

#STOP
-1.                     tSimulationMax
100         MaxIteration

#RUN ------------ last command of this session -----------------

#BORIS
T                       UseBorisCorrection
0.10                    BorisClightFactor
```

Note that in the SWMF some of these commands would be in the part of the PARAM.in file which is intended for the control module CON, while the ones specific to BATS-R-US (e.g. the #INNERBOUNDARY and #BORIS commands would be enclosed between the #BGIN_COMP and #END_COMP markers. In the detailed description of all commands, it is explicitly stated if a command can only be used in the stand alone mode.

3.4 Sessions

A single parameter file can control consecutive **sessions** of the run. Each session looks like

```
#SOME_COMMAND
param1
param2
```

```
...

#STOP
max_simulation_time_for_this_session
max_iter_for_this_session

#RUN
```

while the final session ends like

```
#STOP
max_simulation_time_for_final_session
max_iter_for_final_session

#END
```

The purpose of using multiple sessions is to be able to change parameters during the run. For example one can obtain a coarse steady state solution with a low order scheme in the first session, improve on the solution with a better scheme and finer grid in the second session, then switch to time accurate mode in the third session. The code remembers parameter settings from all previous sessions, so in each session one should only set those parameters which change relative to the previous session. Note that the maximum number of iterations given in the `#STOP` command is meant for the entire run, and not for the individual sessions. On the other hand, when a restart file is read, the iterations prior to the current run do not count.

The `PARAM.in` file and all included parameter files are read into a buffer at the beginning of the run, so even for multi-session runs, changes in the parameter files have no effect once `PARAM.in` has been read.

3.5 The Order of Commands

In essence, the order of parameter commands (`#COMMAND`) is arbitrary, but there are some important restrictions. We should note that the order of the parameters following the command are not however arbitrary and must exactly match what the code requires.

If you want all the input parameters to be echoed back, the first command in `PARAM.in` should be

```
#ECHO
T                  DoEcho
```

If the run starts from scrath (not restarted) the next commands should be

```
#PROBLEMTYPE
problem_type_number
```

and

```
#GRID
...
```

The problem type sets the default values for many parameters, which may be overwritten by the following commands. If the problem type was set later it could overwrite the previous settings with the default values! The grid command sets the size of the computational domain, which is used in the #SAVEPLOT command, for example. The rest of the commands can be in arbitrary order.

If the code starts from restart files, it reads in the `#PROBLEMTYPE` and `#GRID` commands from the restart header file. This should be done with an #INCLUDE command, which should be at the beginning of the PARAM.in file. For example

```
#INCLUDE
GM/restartIN/restart.H
```

In older versions of BATS-R-US, the same effect was achieved with the #RESTART command. For sake of backwards compatibility the #RESTART command is still supported in the stand alone mode as long as the input restart directory is the default one. It is recommended, however, to use the #INCLUDE command instead, because it does not have to be changed when a stand alone PARAM.in file is used in an SWMF input file, and it also allows to use a non-default directory name.

There are several parameters which should not be changed once the code has begun to execute. In other words, these parameters can be defined only during the first session. These parameters typically either involve geometries which cannot be changed, or physical parameters for which there is no reasonable reason to change. These commands are marked with an `if=''$_IsFirstSession''` conditional in the PARAM.XML file. Some of the more commonly used first-session-only commands are

```
#PROBLEMTYPE
#GRID
#AMRINIT
#GAMMA
#SOLARWIND
#MAGNETOSPHERE
#BODY
#SECONDBODY
```

If any of these parameters are attempted to be changed in later sessions, a warning is printed on the screen, and the command is ignored.

In several commands the frequency or 'time' of some action has to be defined. This is usually done with a pair of parameters. The first defines the frequency or time in terms of the number of time steps, and the second one in terms of the simulation time. A negative value for the frequency means that it should not be taken into account. For example, in time accurate mode,

```
#SAVERESTART
T            DoSaveRestart
2000         DnSaveRestart
-1.          DtSaveRestart
```

means that a restart file should be saved after every 2000th time step, while

```
#SAVERESTART
T            DoSaveRestart
-1           DnSaveRestart
100.0        DtSaveRestart
```

means that it should be saved every 100 second in terms of physical time. Defining positive values for both frequencies might be useful when switching from steady state mode to time accurate mode. In the steady state mode the DnSaveRestart parameter is used, while in time accurate mode the DtSaveRestart if it is positive. But it is more typical and more intuitive to either use the frequencies based on time steps in all sessions, or to explicitly repeat the command in the first time accurate session with the time frequency set.

3.6 Command Defaults

A quick glance the the `PARAM.XML` file shows that there is an overwhelmingly large number of input parameters to BATS-R-US. Especially daunting is the long list of parameters which controls details of the numerical methods that the code uses. Fortunately, many of these will never need to be set by the beginning user.

BATS-R-US sets many of the parameters to reasonable values in the source file MH_set_parameters.f90. The two routines set_defaults and set_problem_defaults set the appropriate values. The defaults have been chosen because they work for the vast majority of problems for which the code has been run. The PARAM.XML file defines which commands are required with the required=''T'' attribute of the <command...> tag. In general, commands which deal with details of numerics are defaulted to reasonable values. These may (and should) need to be changed eventually by most users to achieve the desired code speed or solution accuracy, but will work reasonably well with the defaults for most problems. Physics parameters are also defaulted by the problem type and should work fine when running the code on a previously run problem. This will be commonly changed by users as they begin to run their own simulations. Finally, geometry, problem type, flow control (time stepping, start and stops, etc.) and plotting output are generally not defaulted.

3.7 Iterations and Frequency of Output

The purpose of this section is to try to help the user understand what at first may seem like an inconsistent use of stopping frequencies and output frequencies. After using BATS-R-US over several years, it is clear to the authors that the code is used in specific ways and the frequencies are very specifically designed to meet these needs and is the most reasonable implementation. The main "inconsistencies" come into play when the code is restarted, but let us begin by elaborating on time stepping and output frequencies in the code.

We begin by defining several different quantities and the variables that represent them in the code. The variable nITER, represents the number of "iterations" that the simulation has taken since it began running. This number starts at zero every time the code is run, even if beginning from a restart file. This is reasonable since most users know how many iterations the code can take in a certain amount of CPU time and it is this number that is needed when running in a queue. The quantity n_step is a number of "time steps" that the code has taken in total. This number starts at zero when the code is started from scratch, but when started from a restart file, this number will start with the time step at which the restart file was written. This implementation lets the user output data files at a regular interval, even when a restart happens at an odd number of iterations. The quantity time_simulation is the amount of simulated, or physical, time that the code has run. This time starts when time accurate time stepping begins. When restarting, it starts from the physical time for the restart. Of course the time should be cumulative since it is the physically meaningful quantity. We will use these three phrases("iteration", "time step", "time") with the meanings outlined above.

As outlined in section 3.5, commands that ask for a frequency of doing something (say writing the restart file) or for ending a session asks for either a number of iterations, time steps or a physical (simulated) time, depending on whether the time stepping mode is local or time accurate. With the #SAVERESTART command, for example, in local time stepping mode the command with its parameters would be

```
#SAVERESTART
T            DoSaveRestart
2000         DnSaveRestart
-1.          DtSaveRestart
```

and a restart file would be written to disk every 2000 time steps. A negative value for the time frequency means that it should not be taken into account. The previous command works in time accurate mode too, but it ismore typical to define the frequency in physical time:

```
#SAVERESTART
T            DoSaveRestart
-1           DnSaveRestart
100.0        DtSaveRestart
```

which means that the restart files should be saved every 100 seconds. Defining positive values for both frequencies is possible but not very easy to read. In steady state mode the time step frequency, while in time accurate mode the physical time frequency will be used.

Now, what happens when the user has more than one session and he or she changes the frequencies. Let us examine what would happen in the following sample of part of a `PARAM.in` file. For the following example we will assume that when in time accurate mode, 1 iteration simulates 1 second of time. Columns to the right indicate the values of `nITER`, `n_step` and `time_simulation` at which restart files will be written in each session.

```
                                          Restart Files Written at:
==SESSION 1                          Session   nITER   n_step   time_simulation
#TIMEACCURATE                         --------  ------  -------  --------------
F            DoTimeAccurate

#SAVERESTART                              1       200      200             0.0
T            DoSaveRestart                1       400      400             0.0
200          DnSaveRestart
-1.0         DtSaveRestart

#STOP
400          MaxIteration
-1.          tSimulationMax

#RUN ==END OF SESSION 1==

#SAVERESTART                              2       600      600             0.0
T            DoSaveRestart                2       900      900             0.0
300          DnSaveRestart
-1.0         DtSaveRestart

#STOP
1000         MaxIteration
-1.          tSimulationMax

#RUN ==END OF SESSION 2==

#TIMEACCURATE
T            DoTimeAccurate

#SAVERESTART                              3      1100     1100           100.0
T            DoSaveRestart                3      1200     1200           200.0
-1           DnSaveRestart                3      1300     1300           300.0
100.0        DtSaveRestart

#STOP
-1           MaxIteration
300.0        tSimulationMax

#RUN ==END OF SESSION 3==

#SAVERESTART                              4      1400     1400           400.0
T            DoSaveRestart                4      1800     1800           800.0
-1           DnSaveRestart                4      2000     2000          1000.0
400.0        DtSaveRestart

#STOP
-1           MaxIteration
1000.0       tSimulationMax

#END  ==END OF SESSION 4==
```

Now the question is how many iterations will be taken and when will restart file be written out. In session 1 the code will make 400 iterations and will write a restart file at time steps 200 and 400. Since the iterations in the #STOP command are cumulative, the #STOP command in the second session will have the code make 600 more iterations for a total of 1000. Since the timing of output is also cumulative, a restart file will be written at time step 600 and at 900. After session 2, the code is switched to time accurate mode. Since we have not run in this mode yet the simulated (or physical) time is cumulatively 0. The third session will run for 300.0 simulated seconds (which for the sake of this example is 300 iterations). The restart file will be written after every 100.0 simulated seconds or in other words at time steps 1100, 1200 and 1300. The #STOP command in Session 4 tells the code to simulate 700.0 more seconds for a total of 1000.0 seconds. The code will make a restart file when the time is a multiple of 400.0 seconds or at 400.0 and 800.0 seconds (1400 and 1800 time steps). Note that a restart file will also be written at time 1000.0 seconds (time step 2000) since this is the end of a run.

Hopefully it is clear how the simulation is stopped and when output is written. Unfortunately, when restarting, things change just slightly. Let us say that we want to restart running from where our previous example left off. We had written a final restart file at 1000.0 seconds of simulated time, which was 2000 time steps. We want to have the following PARAM.in file executed.

```
                                          Restart Files Written at:
==SESSION 1                            Session   nITER   n_step   time_simulation
                                        --------  ------  -------  --------------
#INCLUDE                                              0     2000           1000.0
GM/restartIN/restart.H

#TIMEACCURATE
F            DoTimeAccurate

#SAVERESTART                                  1       200     2200           1000.0
T            DoSaveRestart
1100         DnSaveRestart
-1.0         DtSaveRestart

#STOP
500          MaxIteration
-1.          tSimulationMax

#RUN ==END OF SESSION 1==

#SAVERESTART                                  2       700     2700           1000.0
T            DoSaveRestart                    2      1000     3000           1000.0
300          DnSaveRestart
-1.0         DtSaveRestart

#STOP
1000         MaxIteration
-1.          tSimulationMax

#END ==END OF SESSION 2==
```

Since we switched to local time stepping we only have to worry about iteration number and time steps. Here we notice the difference in the #STOP command when restarting and looking at the iteration number. With a restart, the #STOP command does not consider the cumulative number of time steps but starts again. However, the output frequency is based on the cumulative time step. The simulation will make 500 iterations in the first session. This

would cumulatively be 2500 time steps. The restart file will be written out at 2200 cumulative time steps or at 200 iterations into this session. The second session will make 500 more iterations for a total of 1000 in this run or 3000 time steps over all. A restart file will be written out at multiples of 300 time steps taken relative to the cumulative total number of time steps. In other words at 2700 and 3000 time steps over all, which is at 700 and 1000 iterations in this run.

This example shows how iterations for stopping are cumulative within a run but are not at restart, but that output is always based on the cumulative number of time steps.

Now let us take one last example. We want to restart from 1000.0 seconds (2000 time steps) just as in the previous example, but we want to continue with a time accurate run.

```
                                                  Restart Files Written at:
==SESSION 1                                    Session   nITER   n_step   time_simulation
                                                --------  ------  -------  --------------
#INCLUDE                                                     0     2000           1000.0
GM/restartOUT/restart.H

#TIMEACCURATE
T            DoTimeAccurate

#SAVERESTART                                         1       200     2200           1200.0
T            DoSaveRestart
-1           DnSaveRestart
600.0        DtSaveRestart

#STOP
-1           MaxIteration
1500.0       tSimulationMax

#RUN ==END OF SESSION 1==

#SAVERESTART                                         2       700     2700           1500.0
T            DoSaveRestart                           2      1000     3000           2000.0
-1           DnSaveRestart
750.0        DtSaveRestart

#STOP
-1           MaxIteration
2000.0       tSimulationMax

#END ==END OF SESSION 2 =
```

In this example, we see that in time accurate mode the simulated, or physical, time is always cumulative. To make 500.0 seconds more simulation, the original 1000.0 seconds must be taken into account. In this example, since each second is 1 iteration, the restart file would be written at the same time steps as in the previous example. The final output (at 2000.0 seconds) in this case is not because a frequency was hit but because the run ended.

Throughout this section, we have used the frequency of writing restart files as an example. The frequencies of writing plot files, writing logfiles and doing AMR work similarly. When some of these files are written, they have in the file name a time step number. This number is always the cumulative number of time steps.

3.8 Input Commands for the BATSRUS: GM, EE, SC, IH and OH Components

List of MH (GM, EE, SC, IH, and OH) commands used in the PARAM.in file

3.8.1 Stand alone mode

#COMPONENT command

```
#COMPONENT
GM                      NameComp
```

This command can be used in the stand-alone mode to make BATSRUS behave as if it was the Global Magnetosphere (GM), Eruptive Event (EE), Solar Corona (SC), Inner Heliosphere (IH) or Outer Heliosphere (OH) component of the SWMF. The NameComp variable contains the two-character component ID of the selected component. If NameComp is different from the default component value, then the default values for all parameters (including the component dependent defaults, like coordinate system) are reset, therefore it should occur as the first command if it is used to change the behavior of BATSRUS. The default behavior is Global Magnetosphere (GM) for the stand-alone BATSRUS.

The command is also saved into the restart header files.

In the SWMF the BATSRUS codes are configured to the appropriate components, so the default components should not be changed by this command.

#DESCRIPTION command

```
#DESCRIPTION
This is a test run for Jupiter with no rotation.
```

This command is only used in the stand alone mode.

The StringDescription string can be used to describe the simulation for which the parameter file is written. The #DESCRIPTION command and the StringDescription string are saved into the restart file, which helps in identifying the restart files.

The default value is “Please describe me!”, which is self explanatory.

#ECHO command

```
#ECHO
T                       DoEcho
```

This command is only used in the stand alone mode.

If the DoEcho variable is true, the input parameters are echoed back. The default value for DoEcho is .false., but it is a good idea to set it to true at the beginning of the PARAM.in file.

#PROGRESS command

```
#PROGRESS
10                      DnProgressShort
100                     DnProgressLong
```

The frequency of short and long progress reports for BATSRUS in stand alone mode. These are the defaults. Set -1-s for no progress reports.

#TIMEACCURATE command

```
#TIMEACCURATE
F               IsTimeAccurate
```

This command is only used in stand alone mode.

If IsTimeAccurate is set to true, BATSRUS solves a time dependent problem. If IsTimeAccurate is false, a steady-state solution is sought for. It is possible to use steady-state mode in the first few sessions to obtain a steady state solution, and then to switch to time accurate mode in the following sessions. In time accurate mode saving plot files, log files and restart files, or stopping conditions are taken in simulation time, which is the time relative to the initial time. In steady state mode the simulation time is not advanced at all, instead the time step or iteration number is used to control the frequencies of various actions.

In steady-state mode BATSRUS uses different time steps in different grid cells (limited only by the local stability conditions) to accelerate the convergence towards steady state.

The default is time accurate mode.

#TIMEWARP command

```
#TIMEWARP
T                       UseTimeWarp
10.0                    uWarpDim (read if UseTimeWarp is true)
```

The goal is to solve the equations in a coordinate system where time is shifted with the d = X, Y, Z, or R coordinate as t' = t + d/uWarpDim, where uWarpDim is a positive speed larger than the fastest characteristic speed in dimensional units. Time warping works only if the flow is supersonic/superfast at the outer boundaries in the positive warp direction (see #WARPDIM) and the warp speed exceeds the fastest wave speed.

The default is UseTimeWarp false.

#WARPDIM command

```
#WARPDIM
1                       iDimWarp
```

Direction of time warping if it is switched on (see #TIMEWARP). The iDimWarp=0 setting means radial direction, while 1 to 3 corresponds to the X, Y, and Z directions, respectively.

Default is iDimWarp = 0.

#WARPCMAX command

```
#WARPCMAX
F                       UseWarpCmax
```

If UseWarpCmax is true then use the maximum speed of the warped variables times the jump in the warped variables in the numerical flux. If UseWarpCmax is false, use the original maximum speed with the jump in the original variables.

Default value is true.

#WARPSCHEME command

```
#WARPSCHEME
1e-8                    Tolerance
20                      MaxIteration
1                       DnJacobian
1e-6                    EpsRel
1e-8                    EpsAbs
```

The conversion from the warped variables W=U-F_r/uWarp back to normal state variable U uses a Newton iterative scheme.

The Tolerance parameter sets if the iterative solution Witer is close enough to W, abs(W-Witer).le.Tolerance*[abs(W)+abs(Witer)], to finish the iteration.

MaxIteration limits the number of iterations performed to reach the tolerance. If MaxIteration is 1 then the scheme is linearized in dU/dW.

DnJacobian sets how often the dU/dW Jacobian is calculated, which requires calculating dW/dU and an LU decomposition. Performing this less frequently saves computation time per iteration, but it may increase the required number of iterations.

EpsRel and EpsAbs are used to set the perturbation of U for calculating a row of the dW/dU matrix for a given variable u as dW/du=[W(u+Eps)-W]/Eps where Eps=EpsRel*abs(u)+EpsAbs.

Default values are shown above.

#SUBCYCLING command

```
#SUBCYCLING
T               UseSubcycling (rest read if UseSubcycling is true)
T               UseMaxTimeStep
10.0            DtLimitDim
```

This command controls how the time stepping works in time accurate mode.

If UseSubcycling is true, the time step size in each grid block can be different. This algorithm is sometimes called ”subcycling” because some of the blocks will take several small time steps during a single global time step. This should not be confused with the ”steady state” mode (see the TIMEACCURATE command) where each grid cell takes different time steps and the result is only valid if a steady state is reached.

If UseMaxTimeStep is true, each blocks takes the time step determined by the local stability condition but limited by the DtLimitDim parameter.

If UseMaxTimeStep is false, then the local time step will be set by the AMR level. For Cartesian grids the time step will be proportional to the physical cell size, which is optimal if the wave speeds are roughly constant in the whole domain. Note that the global time step is set so that the stability conditions hold in every grid block. A conservative flux correction is applied at the resolution changes. On the other hand, the normal velocity, normal magnetic/electric field etc. used in some source terms are not ”corrected”, which is different from the default uniform time step algorithm.

The DtLimitDim parameter sets an upper limit on the time step for all the grid blocks in dimensional time units (typically seconds). Setting this parameter to a reasonable value can greatly improve the accuracy and robustness of the scheme with minimal effect on the computational speed, since typically there are relatively few blocks that would allow very large time steps. Setting DtLimitDim to a very large value will result in a global time step based on the block with the largest stable time step.

Currently the subcycling algorithm is either first or second order accurate in time depending on the value of nStage set in the #TIMESTEPPING command.

For spherical grids the #FIXAXIS command does not work with the subcycling algorithm, on the other hand the #COARSENAXIS command can be used.

See also the #PARTSTEADY, #PARTLOCALTIMESTEP and #TIMESTEPLIMIT commands for related time stepping algorithms.

The default is using a uniform time step for the whole domain.

#BEGIN_COMP command

This command is allowed in stand alone mode only for the sake of the test suite, which contains these commands when the framework is tested.

#END_COMP command

This command is allowed in stand alone mode only for the sake of the test suite, which contains these commands when the framework is tested.

#RUN command

```
#RUN
```

This command is only used in stand alone mode.

The #RUN command does not have any parameters. It signals the end of the current session, and makes BATSRUS execute the session with the current set of parameters. The parameters for the next session start after the #RUN command. For the last session there is no need to use the #RUN command, since the #END command or simply the end of the PARAM.in file makes BATSRUS execute the last session.

#END command

```
#END
```

The #END command signals the end of the included file or the end of the PARAM.in file. Lines following the #END command are ignored. It is not required to use the #END command. The end of the included file or PARAM.in file is equivalent with an #END command in the last line.

3.8.2 Planet parameters

The planet commands can only be used in stand alone mode. The commands allow to work with an arbitrary planet. It is also possible to change some parameters of the planet relative to the real values.

By default Earth is assumed with its real parameters. Another planet (moon, comet) can be selected with the #PLANET (#MOON, #COMET) command. The real planet parameters can be modified and simplified with the other planet commands listed in this subsection. These modified commands cannot precede the #PLANET command!

#PLANET command

```
#PLANET
NEW                     NamePlanet (rest of parameters read for unknown planet)
6300000.0               RadiusPlanet [m]
5.976E+24               MassPlanet   [kg]
0.000000199             OmegaPlanet  [radian/s]
23.5                    TiltRotation [degree]
DIPOLE                  TypeBField
11.0                    MagAxisThetaGeo [degree]
289.1                   MagAxisPhiGeo   [degree]
-31100.0E-9             DipoleStrength  [T]
```

The NamePlanet parameter contains the name of the planet with arbitrary capitalization. In case the name of the planet is not recognized, the following variables are read: RadiusPlanet is the radius of the planet, MassPlanet is the mass of the planet, OmegaPlanet is the angular speed relative to an inertial frame, and TiltRotation is the tilt of the rotation axis relative to ecliptic North, TypeBField, which can be ”NONE” or ”DIPOLE”. TypeBField=”NONE” means that the planet does not have magnetic field. If TypeBField is set to ”DIPOLE” then the following variables are read: MagAxisThetaGeo and MagAxisPhiGeo are the colatitude and longitude of the north magnetic pole in corotating planetocentric coordinates. Finally DipoleStrength is the equatorial strength of the magnetic dipole field. The units are indicated in the above example, which shows the Earth values approximately.

The default value is NamePlanet=”Earth”. Although many other planets and some of the moons are recognized, some of the parameters, like the equinox time are not yet properly set.

#ROTATIONAXIS command

```
#ROTATIONAXIS
T                       IsRotAxisPrimary (rest of parameters read if true)
23.5                    RotAxisTheta
198.3                   RotAxisPhi
```

If the IsRotAxisPrimary variable is false, the rotational axis is aligned with the magnetic axis. If it is true, the other two variables are read, which give the position of the rotational axis at the initial time in the GSE coordinate system. Both angles are read in degrees and stored internally in radians.

The default is to use the true rotational axis determined by the date and time given by #STARTTIME.

#ROTATION command

```
#ROTATION
T                       UseRotation
24.06575                RotationPeriod [hour] (read if UseRotation is true)
```

If UseRotation is false, the planet is assumed to stand still, and the OmegaPlanet variable is set to zero. If UseRotation is true, the RotationPeriod variable is read in hours, and it is converted to the angular speed OmegaPlanet given in radians/second. Note that OmegaPlanet is relative to an inertial coordinate system, so the RotationPeriod is not 24 hours for the Earth, but the length of the astronomical day.

The default is to use rotation with the real rotation period of the planet.

#MAGNETICAXIS command

```
#MAGNETICAXIS
T                       IsMagAxisPrimary (rest of parameters read if true)
34.5                    MagAxisTheta [degree]
0.0                     MagAxisPhi   [degree]
```

If the IsMagAxisPrimary variable is false, the magnetic axis is aligned with the rotational axis. If it is true, the other two variables are read, which give the position of the magnetic axis at the initial time in the GSE coordinate system. Both angles are read in degrees and stored internally in radians.

The default is to use the true magnetic axis determined by the date and time given by #STARTTIME.

#MAGNETICCENTER command

```
#MAGNETICCENTER
0.1                     MagCenterX
-0.02                   MagCenterY
0.0                     MagCenterZ
```

Shifts the magnetic center (e.g. the center of the dipole) to the location given by the three parameters. The default is no shift (at least for most planets).

#MONOPOLEB0 command

```
#MONOPOLEB0
16.0                    MonopoleStrengthSi [Tesla]
```

The MonopoleStrengthSi variable contains the magnetic strength of the monopole B0 field at R=1 radial distance. The unit is Tesla unless the normalization is set to NONE (see #NORMALIZATION command), when it is just the normalized value.

The default value is zero.

#DIPOLE command

```
#DIPOLE
-3.11e-5                DipoleStrengthSi [Tesla]
```

The DipoleStrengthSi variable contains the magnetic equatorial strength of the dipole magnetic field in Tesla.

The default value is the real dipole strength for the planet. For the Earth the default is taken to be -31100 nT. The sign is taken to be negative so that the magnetic axis can point northward as usual.

#UPDATEB0 command

```
#UPDATEB0
0.0001                  DtUpdateB0
```

The DtUpdateB0 variable determines how often the position of the magnetic axis is recalculated. A negative value indicates that the motion of the magnetic axis during the course of the simulation is neglected. This is an optimization parameter, since recalculating the values which depend on the orientation of the magnetic field can be costly. Since the magnetic field moves relatively slowly as the planet rotates around, it may not be necessary to continuously update the magnetic field orientation.

The default value is 0.0001, which means that the magnetic axis is continuously followed.

#IDEALAXES command

```
#IDEALAXES
```

The #IDEALAXES command has no parameters. It sets both the rotational and magnetic axes parallel with the ecliptic North direction. In fact it is identical with the commands:

```
#ROTATIONAXIS
T               IsRotAxisPrimary
0.0             RotAxisTheta
0.0             RotAxisPhi

#MAGNETICAXIS
F               IsMagAxisPrimary
```

but much shorter.

#MULTIPOLEB0 command

```
#MULTIPOLEB0
T                       UseMultipoleB0
10                      MaxHarmonicDegree
planetharmonics.txt     NamePlanetaryHarmonicsFile
```

Using this command, you can specify the planetary magnetic field (B0) using the Spherical Harmonics expansion. This is useful for e.g. to model the IGRF or complicated planetary magnetic field. Planetary rotation is allowed when using this option. We suggest using the GSE coordinate system using the #COORDSYSTEM command. If UseMultipoleB0 is true, the #IDEALAXES command is enforced i.e. the dipole magnetic axis is aligned with the rotation axis. No matter what coordinate system you use in GM, the multipole B0 calculation is always done in the GEO coordinate system.

As of now this feature cannot be used with the IE solver, and should be used in standalone GM/BATSRUS only. Secular variation has not been implemented yet.

The planetharmonics.txt file should be of the form -

```
Header line (is not read) - n m g h (follow this specific order)
0  0   0.000000      0.000000
1  0  -29619.400000  0.000000
1  1  -1728.200000   5186.100000
2  0  -2267.700000   0.000000
2  1   3068.400000  -2481.600000
2  2   1670.900000  -458.000000
```

Where g and h are the Legendre coefficients in units of nT.

3.8.3 User defined input

#USERSWITCH command

```
#USERSWITCH
-all +init +ic -perturb +B0 +source +update +progress     StringSwitch
```

This command controls the use of user defined routines in src/ModUser.f90. The string contains a single-space separated list of switches starting with a + sign or a - sign for switching the routines on or off, respectively. This command can occur multiple times in the same session. Previous settings are preserved for the next session. The possible switches are (with alternative names):

```
all                     : switch all routines on or off
init, init_session      : initialize user module before running session
ic, initial_condition   : initial conditions
perturb, perturbation   : perturbation (default is false)
B0, get_b0              : user defined B0 field
source                  : user source terms (explicit and implicit)
Sexpl, source_expl      : explicit user source terms
Simpl, source_impl      : point-implicit user source terms
update, update_state    : user defined state update
progress, write_progress: user progress report
```

Default is that init is on all others are off. When the perturbation is switched on, it gets switched off after the perturbation is applied. The corresponding logicals can be changed in the user module.

#USERINPUTBEGIN command

```
#USERINPUTBEGIN
```

This command signals the beginning of the section of the file which is read by the subroutine user_read_inputs in the ModUser.f90 file. The section ends with the #USERINPUTEND command. There is no XML based parameter checking in the user section.

#USERINPUTEND command

```
#USERINPUTEND
```

This command signals the end of the section of the file which is read by the subroutine user_read_inputs in the ModUser.f90 file. The section begins with the #USERINPUTBEGIN command. There is no XML based parameter checking in the user section.

3.8.4 Testing and timing

#TESTINFO command

```
#TESTINFO
T               DoWriteCallSequence
```

If DoWriteCallSequence is set to true, the code will attempt to produce a call sequence from the stop_mpi subroutine (which is called when the code finds an error) by making an intentional floating point exception. This will work only if the compiler is able to and requested to produce a call sequence. The NAGFOR compiler combined with the -debug flag can do that.

Default is DoWriteCallSequence=F.

#TEST command

```
#TEST
read_inputs
```

A space separated list of subroutine names. Default is empty string.

Examples:
read_inputs - echo the input parameters following the #TEST line
project_B - info on projection scheme
implicit - info on implicit scheme
krylov - info on the Krylov solver
message_count- count messages
initial_refinement
...

Check the subroutines for call setoktest("...",oktest,oktest_me) to see the appropriate strings.

#TESTIJK command

```
#TESTIJK
1                       iTest           (cell index for testing)
1                       jTest           (read for nDim = 2 or 3)
1                       kTest           (read for nDim = 3)
1                       iBlockTest      (block index for testing)
0                       iProcTest       (processor index for testing)
```

The location of test info in terms of indices, block and processor number. Note that the user should set #TESTIJK or #TESTXYZ, not both.

The default test cell is shown by the example.

#TESTXYZ command

```
#TESTXYZ
1.5                     xTest           (X coordinate of cell for testing)
-10.5                   yTest           (Y coordinate for nDim=2 or 3)
-10.                    zTest           (Z coordinate for nDim=3)
```

The location of test info in terms of coordinates. Note that the user should set #TESTIJK or #TESTXYZ, not both.

The default test cell is described in #TESTIJK.

#TESTVAR command

```
#TESTVAR
12                    NameTestVar

#TESTVAR
p                     NameTestVar
```

Index or the name of the variable to be tested. The name should agree with one of the names in the NameVar_V array in ModEquation.f90 (case insensitive). If an index is given instead of a name, it should be in the range 1 to nVar.

Default is the first variable that is usually density.

#TESTDIM command

```
#TESTDIM
1                         iDimTest
```

Index of dimension/direction to be tested. Default is X dimension.

#TESTSIDE command

```
#TESTSIDE
0                         iSideTest (-1, 0, 1)
```

Select the side of the cell to be tested. -1 is for ”left” side, +1 is for right side, 0 is for both sides. Currently this is implemented in the UpdateStateFast code only, where the sides are done with multiple threads on the GPU. Default value is shown.

#TESTPIXEL command

```
#TESTPIXEL
200                       iPixTest
200                       jPixTest
```

Indexes of the test pixel of the LOS plot.

#STRICT command

```
#STRICT
T                         UseStrict
```

If true then stop when parameters are incompatible. If false, try to correct parameters and continue. Default is true, i.e. strict mode

#VERBOSE command

```
#VERBOSE
-1                        lVerbose
```

Verbosity level controls the amount of output to STDOUT. Default level is 1.
lVerbose ≤ -1 only warnings and error messages are shown.
lVerbose ≥ 0 start and end of sessions is shown.
lVerbose ≤ 1 a lot of extra information is given.
lVerbose ≤ 10 all calls of set_oktest are shown for the test processor.
lVerbose ≤ 100 all calls of set_oktest are shown for all processors.

#DEBUG command

```
#DEBUG
F                       DoDebug         (use it as if(okdebug.and.oktest)...)
F                       DoDebugGhost    (parameter for show_BLK in library.f90)
```

Excessive debug output can be controlled by the global okdebug parameter

#USERMODULE command

```
#USERMODULE
TEST PROBLEM Smith
```

Checks the selected user module. If the name differs from that of the compiled user module, a warning is written, and the code stops in strict mode (see #STRICT command). This command is written into the restart header file too, so the user module is checked when a restart is done. There are no default values. If the command is not present, the user module is not checked.

#EQUATION command

```
#EQUATION
MHD                     NameEquation
8                       nVar
```

Define the equation name and the number of variables. If any of these do not agree with the values determined by the code, BATSRUS stops with an error. Used in restart header files and can be given in PARAM.in as a check and as a description.

#RESTARTVARIABLES command

```
#RESTARTVARIABLES
Rho Mx My Mz Bx By Bz p           NameRestartVar
```

The NameRestartVar string contains a space separated list of variable names that are stored in a restart file. This command is saved automatically into the restart files. Other then useful information about the content of the restart file, it is also needed for the #CHANGEVARIABLES command.

The default assumption is that the restart file contains the same variables as the equation module that the code is compiled with.

#CHANGEVARIABLES command

```
#CHANGEVARIABLES
T                       DoChangeRestartVariables
```

This command allows reading restart files that were produced with different equation and user modules than what the restarted code is using. If DoChangeRestartVariables is set to true, the code attempts to copy the corresponding variables correctly. This typically works if the restart file contains all the variables that the restarted code is using. See subroutine match_copy_restart_variables in ModRestartFile.f90 for more detail.

The default is to use the same variables and equation modules during restart.

#SPECIFYRESTARTVARMAPPING command

```
#SPECIFYRESTARTVARMAPPING
T                       DoSpecifyRestartVarMapping
H2OpRho H2OpP           NameVarsRestartFrom
H3OpRho P               NameVarsRestartTo
```

This command allows specifying the mapping of variables when reading restart files in one equation/user file to another equation/user file. In the above example, the code will use the values of H2OpRho/H2OpP in the old equation/user file to initialize the variables H3OpRho/P in the new equation/user file.

This mapping applies after the default mapping which maps the variables with the same variable names, meaning that it will overwrite the default mapping algorithm. For example, if both the original equation/user file and the new equation/user file have the variable P, the code will initialize P in the new equation/user file with the values of P in the old equation/user file by default. However, in the above example, users choose to map H2OpP in the old equation/user file to the varaible P in the new equation/user file. The mapping of variables is also shown in the runlog in case the user wants to see how the variables are mapped.

The deafult is not to apply user specified mapping even a different equation/user file is used during restart.

#PRECISION command

```
#PRECISION
8                       nByteReal
```

Define the number of bytes in a real number. If it does not agree with the value determined by the code, BATSRUS stops with an error unless the strict mode is switched off. This is used in restart header files to store (and check) the precision of the restart files. It is now possible to read restart files with a precision that differs from the precision the code is compiled with, but strict mode has to be switched off with the #STRICT command. The #PRECISION command may also be used to enforce a certain precision.

#CHECKGRIDSIZE command

```
#CHECKGRIDSIZE
  4                       nI
  4                       nJ
  4                       nK
576                       MinBlockAll
```

This command is typically used in the restart headerfile to check consistency. The nI, nJ, nK parameters provide the block size in terms of number of grid cells in the 3 directions. The code stops with an error message if nI, nJ, or nK differ from the values set with Config.pl -g=...

The MinBlockAll parameter stores the total number of grid blocks actually used at the time the restart file was saved. When doing a restart, it is used to set the number of grid blocks to be sufficient to coninue the run as long as no AMR is performed. To allocate more blocks, use the #GRIDBLOCKALL command.

This command can also be used directly in PARAM.in to check the block size and to set the total number of blocks at the same time.

#BLOCKLEVELSRELOADED command

```
#BLOCKLEVELSRELOADED
```

This command means that the restart file contains the information about the minimum and maximum allowed refinement levels for each block. This command is only used in the restart header file.

#TIMING command

```
#TIMING
T                       UseTiming       (rest of parameters read if true)
-2                      DnTiming        (-3 none, -2 final, -1 each session)
-1                      nDepthTiming    (-1 for arbitrary depth)
cumu                    TypeTimingReport    (cumu/list/tree + optional 'all')
```

This command can only be used in stand alone mode. In the SWMF the #TIMING command should be issued for CON.

If UseTiming=.true., the TIMING module must be on. If UseTiming=.false., the execution is not timed.

Dntiming determines the frequency of timing reports. If DnTiming .ge. 1, a timing report is produced every dn_timing step. If DnTiming .eq. -1, a timing report is shown at the end of each session. If DnTiming .eq. -2, a timing report is shown at the end of the whole run. If DnTiming .eq. -3, no timing report is shown.

nDepthTiming determines the depth of the timing tree. A negative number means unlimited depth. If TimingDepth is 1, only the full BATSRUS execution is timed.

TypeTimingReport determines the format of the timing reports: 'cumu' - cumulative list sorted by timings 'list' - list based on caller and sorted by timings 'tree' - tree based on calling sequence

If the word 'all' is added, the timing is done on all the CPU-s. One output file will be created for each processor.

The default values are shown above.

3.8.5 Initial and boundary conditions

#UNIFORMSTATE command

```
#UNIFORMSTATE
0.125           StateVar Rho
1.0             StateVar Ux
-0.5            StateVar Uy
0.0             StateVar Uz
1.0             StateVar p
```

The #UNIFORMSTATE command sets up a uniform initial state. This uniform state can be perturbed or modified by the user module. The command sets the primitive variables in the order defined in the equation module.

#SHOCKTUBE command

```
#SHOCKTUBE
1.              LeftState Rho
0.              LeftState Ux
0.              LeftState Uy
0.              LeftState Uz
0.75            LeftState Bx
1.              LeftState By
0.              LeftState Bz
1.              LeftState p
0.125           RightState Rho
0.              RightState Ux
0.              RightState Uy
0.              RightState Uz
0.75            RightState Bx
-1.             RightState By
0.              RightState Bz
```

```
0.1             RightState p
```

The #SHOCKTUBE command can be used to set up a shocktube problem. The left and right state values are given in terms of the primitive variables as defined in the equation module. The shock can be shifted and rotated by the #SHOCKPOSITION command.

By default the initial condition is uniform, and the values are determined by the #SOLARWIND command. The user module can be used to set up more complicated initial conditions.

#SHOCKPOSITION command

```
#SHOCKPOSITION
5.0                     ShockPosition
1/2                     ShockSlope
```

The ShockPosition parameter sets the position where the shock, ie. the interface between the left and right states given by the #SHOCKTUBE command, intersects the X axis. When ShockSlope is 0, the shock normal points in the X direction. Otherwise the shock is rotated around the Z axis, and the tangent of the rotation angle is given by ShockSlope. Possible values are

```
ShockSlope = 0., 1/4, 1/3, 1/2, 1., 2., 3., 4.
```

because these angles can be accurately represented on the grid. The default values are zero, ie. the shock is in the X=0 plane.

#WAVE command

```
#WAVE
Ux                      NameVar
10.0                    Width
0.1                     Amplitude
5.0                     LambdaX
-1.0                    LambdaY
-1.0                    LambdaZ
90.0                    Phase [deg]
```

Add a wave to the initial condition.

NameVar selects the primitive variable to be changed. Width limits the extent of the wave relative to the origin. Inside the width the following formula is applied:

```
Var = Var + Amplitude*cos(Phase + Kx*x + Ky*y + Kz*z)**Exponent
```

where Kx = max(2*pi/LambdaX, 0), so negative LambdaX results in Kx=0. Ky and Kz are calculated similarly. The exponent is given by the name of the command. For #WAVE it is 1, for #WAVE2, #WAVE4 and #WAVE6 it is 2, 4 and 6, respectively. The high power allows setting up tests for the fifth order scheme.

The wave vectors and the amplitudes of vector variables are rotated around the Z axis with the angle of the shock slope if it is not zero.

This command can be repeated to add different waves to different variables. There is no wave perturbation by default.

#BUMP command

```
#BUMP
Rho                     NameVar
0.1                     Amplitude
```

```
5.0                     WidthX
3.0                     WidthY
-1.0                    WidthZ
4.2                     CenterX
0.0                     CenterY
-0.2                    CenterZ
4                       nPower
```

Add a "bump" perturbation to a variable.

NameVar selects the primitive variable to be perturbed.

Amplitude sets the amplitude of the perturbation.

WidthX, WidthY and WidthZ define the spatial extent of the perturbation in the 3 directions. Negative value means that there is no restriction, so 1/Width is set to 0.

CenterX, CenterY and CenterZ define the center of the perturbation.

nPower is the power of the cosine functions, which sets the smoothness. nPower=0 defines a bump with a constant value with a sharp edge. nPower=2 is suitable for convergence studies up to 2nd order.

For a given point at x, y, z, the normalized radial distances from the center is

```
r = sqrt( ((x-CenterX)/WidthX)**2 + ((y-CenterY)/WidthY)**2 ...)
```

The perturbation is applied for r less than 0.5 as

```
Var = Var + Amplitude * cos(pi*r)**nPower
```

This command can be repeated multiple times to add perturbations to multiple variables. No perturbation is applied by default.

#SOLARWIND command

```
#SOLARWIND
5.0                     SwNDim  [n/cc]
100000.0                SwTDim  [K]
-400.0                  SwUxDim [km/s]
0.0                     SwUyDim [km/s]
0.0                     SwUzDim [km/s]
0.0                     SwBxDim [nT]
0.0                     SwByDim [nT]
-5.0                    SwBzDim [nT]
```

This command defines the solar wind parameters for the GM component. The default values are all 0.0-s.

#SOLARWINDFILE command

```
#SOLARWINDFILE
T                       UseSolarWindFile (rest of parameters read if true)
IMF.dat                 NameSolarWindFile
```

Default is UseSolarWindFile = .false.

Read IMF data from file NameSolarWindFile if UseSolarWindFile is true. The data file contains all information required for setting the upstream boundary conditions. Parameter TypeBcWest should be set to 'vary' for the time dependent boundary condition.

If the #SOLARWIND command is not provided then the first time read from the solar wind file will set the normalization of all variables in the GM component. Consequently either the #SOLARWIND command or the #SOLARWINDFILE command with UseSolarWindFile=.true. is required by the GM component.

The input files are strutured similar to the PARAM.in file. There are `#commands` that can be inserted as well as the data. The file containing the upstream conditions should include data in the following order:

```
yr mn dy hr min sec msec bx by bz vx vy vz dens temp
```

The units of the variables should be:

```
Magnetic field (b)     nT
Velocity (v)           km/s
Number Density (dens)  cm^-3
Temperature (Temp)     K
```

The input files can have the following optional commands at the beginning

```
#REREAD      Reread the file if the simulation runs beyond the final time
             See also the #REFRESHSOLARWINDFILE command

#COOR
GSM          The coordinate system of the data: GSM (default) or GSE

#VAR
rho ux uy uz bx by bz p pe

#PLANE       The input data represents values on a tilted plane
20.0         Angle to rotate in the XY plane [deg]
15.0         Angle to rotate in the XZ plane [deg]

#POSITION    Y-Z Position of the satellite (also origin of plane rotation)
20.0         Y location
30.0         Z location

#SATELLITEXYZ  3D Position of the satellite
65.0         X location
0.0          Y location
0.0          Z location

#ZEROBX
T            Bx is ignored and set to zero if true

#TIMEDELAY
3600.0       A constant delay added to the time in the file [s]
```

The #REREAD command tells BATS-R-US to reread the solarwind file when the simulation goes past the time of the last data in the current file. The default behavior is to keep using the last data point, but this can also be changed with the #REFRESHSOLARWINDFILE command.

The #VAR command allows reading an extended set of variables, e.g. densities of multiple species, electron pressure, etc.

Finally, the data should be preceded by a `#START`. The beginning of a typical solar wind input file might look like:

```
#COOR
GSM

#START
```

```
 2004  6  24   0   0  58   0  2.9  -3.1 - 3.7  -300.0  0.0  0.0  5.3  2.00E+04
 2004  6  24   0   1  58   0  3.0  -3.2 - 3.6  -305.0  0.0  0.0  5.4  2.01E+04
```

The maximum number of lines of data allowed in the input file is 50,000. However, this can be modified by changing the variable Max_Upstream_Npts in the file GM/BATSRUS/get_solar_wind_point.f90.

#REFRESHSOLARWINDFILE command

```
#REFRESHSOLARWINDFILE
T                       DoReadAgain
```

If DoReadAgain is set to true and the code is using a solar wind data file, the code will stop running when the time goes beyond the end of the last data point in the solar wind input file and wait until new data arrives (see #SOLARWINDFILE command). The same effect can be achieved with the #REREAD command put into the solar wind input file itself (see #SOLARWIND command).

Default is DoReadAgain false, so the code keeps running with the last value read.

#BODY command

```
#BODY
T                       UseBody (rest of parameters read if true)
3.0                     rBody
4.0                     rCurrents (only read for GM component)
1.0                     BodyNDim (/cc) for fluid 1
10000.0                 BodyTDim (K)   for fluid 1
0.01                    BodyNDim (/cc) for fluid 2
300.0                   BodyTDim (K)   for fluid 2

#BODY
T                       UseBody (rest of parameters read if true)
3.0                     rBody
4.0                     rCurrents (only read for GM component)
1.0                     BodyNDim (/cc) for species 1
0.01                    BodyNDim (/cc) for species 2
300.0                   BodyTDim (K)

#MAGNETOSPHERE
T                       UseBody (rest of parameters read if true)
2.5                     rBody
3.5                     rCurrents (only read for GM component)
28.0                    BodyNDim (/cc)
25000.0                 BodyTDim (K)
```

Note that the #BODY command is most useful for Cartesian grids so that a sphere can be cut out as the inner boundary. For spherical grids the cell based boundary at the minimum radius can be controlled with the #OUTERBOUNDARY and #BOUNDARYSTATE commands.

If UseBody is true, the inner boundary is a spherical surface with radius rBody. The rBody is defined in units of the planet/solar radius. It can be 1.0, in which case the simulation extends all the way to the surface of the central body. In many cases it is more economic to use an rBody larger than 1.

The rCurrents parameter defines where the currents are calculated for the GM-IE coupling as well as for calculating the FAC contribution of ground magnetic field perturbations.

The BodyNDim and BodyTDim parameters define the number density and temperature inside the body, respectively. For multifluid MHD the number density and temperature are given for all the fluids. For multispecies MHD the

number density is given for all species followed by the (common) temperature. The exact effect of these parameters depends on the settings in the #INNERBOUNDARY command.

The default is UseBody=F. Some typical settings are shown above.

#CORONA command

```
#CORONA
1.0                     rCorona
1.5E8                   CoronaNDim (/cc) for fluid 1
1.5E6                   CoronaTDim (K)
1.5E2                   CoronaNDim (/cc) for fluid 2
1.5E6                   CoronaTDim (K)
```

This command can be used to set physical parameters at the inner boundary of the solar domain. Unlike the #BODY command, this command does not switch on the face boundary. rCorona sets the radius of the inner boundary (typically 1 Rs). The rest of the parameters set the density and temperature for AWSoM(-R). For multi-fluid case, the values are repeated.

Default values are shown.

#STAR command

```
#STAR
1.0                     RadiusStar (in solar radius)
1.0                     MassStar   (in solar mass)
25.38                     RotationPeriodStar (in days)
```

Modify the parameters of the central star (when BATSRUS is running in heliospheric mode). Setting zero for the rotation period will switch off the rotation as shown by the example.

By default the Sun is the central star.

#ROTPERIOD command

```
#ROTPERIOD
0.0                     RotPeriodSI (in second)
```

If goal is to switch off (or modify) the effects of the star rotation FOR A GIVEN MODEL (SC, IH, OH, EE), the #STAR command is not applicable, since it also modifies the infrastructure (HGI to HGR stransformation matrix, Earth location, Carrington rotations etc). For this purpose #ROTPERIOD command works, which affects only the model, not the infrastructure.

#NORMALIZATION command

```
#NORMALIZATION
READ                    TypeNormalization
1000.0                  No2SiUnitX   (only read if TypeNormalization=READ)
1000.0                  No2SiUnitU   (only read if TypeNormalization=READ)
1.0e-6                  No2SiUnitRho (only read if TypeNormalization=READ)
```

This command determines what units are used internally in BATSRUS. The units are normalized so that several physical constants become unity (e.g. the permeability of vacuum), so the equations are simpler in the code. The normalization also helps to keep the various quantities within reasonable ranges. For example density of space plasma is very small in SI units, so it is better to use some normalization, like amu/cm^3. Also note that distances and positions

(like grid size, grid resolution, plotting resolution, radius of the inner body etc) are always read in normalized units from the PARAM.in file. Other quantities are read in I/O units (see the #IOUNITS command).

The normalization of the distance, velocity and density are determined by the TypeNormalization parameter. The normalization of all other quantities are derived from these three values. It is important to note that the normalization of number density (defined as the density normalization divided by the proton mass) is usually not consistent with the inverse cube of the normalization of distance.

Possible values for TypeNormalization are NONE, PLANETARY, HELIOSPHERIC, OUTERHELIO, SOLARWIND, USER and READ.

If TypeNormalization="NONE" then the distance, velocity and density units are the SI units, i.e. meter, meter/sec, and kg/m^3. Note that the magnetic field and the temperature are still normalized differently from SI units so that the Alfven speed is $B/\sqrt{\rho}$ and the ion temperature is simply $p/(\rho/AverageIonMass)$, where the AverageIonMass is given relative to the mass of proton.

If TypeNormalization="PLANETARY" then the distance unit is the radius of the central body (moon, planet, or the Sun). If there is no central body, the length normalization is 1km. The velocity unit is rPlanet/s (so time unit is seconds), and the density unit is amu/cm3.

If TypeNormalization="HELIOSPHERIC" then the distance unit is the radius of the central body (Sun, star), the velocity unit is km/s, and the density unit is amu/cm3.

If TypeNormalization="OUTERHELIO" then the distance unit is 1 AU, the velocity unit is km/s, and the density unit is amu/cm3.

TypeNormalization="SOLARWIND" is depreciated! Don't use it. If it is used then the distance unit is the radius of the planet, and the velocity and density are normalized to the density and the sound speed of the solar wind defined by the #SOLARWIND or #SOLARWINDFILE commands. This normalization is very impractical, because it depends on the solar wind values that are variable, and may not even make sense (e.g. for a shock tube test). This normalization is only kept for sake of backwards compatibility for a few user modules (Mars, Venus, Titan, Saturn).

If TypeNormalization="USER" the normalization is set in the user module. This may be useful if the normalization depends on some input parameters.

Finally TypeNormalization="READ" reads the three basic normalization units from the PARAM.in file as shown in the example. This allows arbitrary normalization.

The restart header file saves the normalization with TypeNormalization="READ" and the actual values of the distance, velocity and density normalization factors. This avoids the problem of continuing the run with inconsistent normalization (e.g. if the SOLARWIND normalization is used and the solar wind parameters have been changed). It also allows other programs to read the data saved in the restart files and convert them to appropriate units.

The default normalization is PLANETARY for GM and SOLARWIND for all other components.

#IOUNITS command

```
#IOUNITS
PLANETARY                       TypeIoUnit
```

This command determines the physical units of various parameters read from the PARAM.in file and written out into log files and plot files (if they are dimensional. The units are determined by the TypeIoUnit string. Note that distances and positions are always read in normalized units from PARAM.in but they are written out in I/O units. In most cases the two coincides.

Also note that the I/O units are NOT necessarily physically consistent units. For example one cannot divide distance with time and compare it with the velocity because they may be in inconsistent units. One needs to convert into some consistent units before the various quantities can be combined.

If TypeIoUnits="SI" the input and output values are taken in SI units (m, s, kg, etc).

The PLANETARY units use the radius of the planet for distance, seconds for time, amu/cm^3 for mass density, cm$^-3$ for number density, km/s for speed, nPa for pressure, nT for magnetic field, micro Amper/m^2 for current density, mV/m for electric field, nT/planet radius for div B, and degrees for angles. For any other quantity SI units are used. If there is no planet (see the #PLANET command) then the distance unit is 1 km.

The HELIOSPHERIC units use the solar radius for distance, seconds for time, km/s for velocity, degrees for angle, and CGS units for mass density, number density, pressure, magnetic field, momentum, energy density, current, and div B.

When TypeIoUnit=”NONE” the input and output units are the same as the normalized units (see the #NORMALIZATION command).

Finally when TypeIoUnit=”USER”, the user can modify the I/O units (Io2Si_V) and the names of the units (NameTecUnit_V and NameIdlUnit_V) in the subroutine user_io_units of the user module. Initially the values are set to SI units.

The #IOUNITS command and the value of TypeIoUnits is saved into the restart header file so that one continues with the same I/O units after restart.

The default is ”PLANETARY” unit if BATSRUS is used as the GM component and ”HELIOSPHERIC” otherwise (EE, SC, IH or OH).

#RESTARTINDIR command

```
#RESTARTINDIR
GM/restart_n5000        NameRestartInDir
```

The NameRestartInDir variable contains the name of the directory where restart files are saved relative to the run directory. The directory should be inside the subdirectory with the name of the component.

Default value is ”GM/restartIN”.

#RESTARTINFILE command

```
#RESTARTINFILE
one series                      TypeRestartInFile
```

This command is saved in the restart header file which is included during restart, so normally the user does not have to use this command at all. The TypeRestartInFile parameter describes how the restart data was saved: into separate files for each processor (’proc’), into separate files for each grid block (’block’) or into a single direct access file (’one’). The optional ’series’ string means that a series of restart files were saved with the iteration number added to the beginning of the file names.

The default value is ’block’ for sake of backwards compatibility.

#NEWRESTART command

```
#NEWRESTART
T               DoRestartBFace
```

The RESTARTINDIR/restart.H file always contains the #NEWRESTART command. This command is really used only in the restart headerfile. Generally it is not inserted in a PARAM.in file by the user.

The #NEWRESTART command sets the following global variables: DoRestart=.true. (read restart files), DoRestartGhost=.false. (no ghost cells are saved into restart file) DoRestartReals=.true. (only real numbers are saved in blk*.rst files).

The DoRestartBFace parameter tells if the face centered magnetic field is saved into the restart files. These values are used by the Constrained Transport scheme.

#RESTARTWITHFULLB command

```
#RESTARTWITHFULLB
```

This command is written by the code into the restart header file and indicates that the full magnetic field (B=B0+B1) was saved. In the past only B1 was saved. Saving the total field allows changing B0 during restart and also allows using the restart files without knowledge of B0. The current default is saving total B, so this command is always present in the current restart header files.

#OUTERBOUNDARY command

```
#OUTERBOUNDARY
outflow                 TypeBc1
inflow                  TypeBc2
float                   TypeBc3 (only read in 2D and 3D)
float                   TypeBc4 (only read in 2D and 3D)
float                   TypeBc5 (only read in 3D)
float                   TypeBc6 (only read in 3D)
```

This command defines how the ghost cells are filled in at the cell based boundaries at the edges of the grid. TypeBc1 and TypeBc2 describe the boundaries at the minimum and maximum values of the first (generalized) coordinate. For a Cartesian grid these are at xMin and xMax, while for a spherical or cylindrical grid these are at rMin and rMax. TypeBc3 and TypeBc4 describe the boundaries at the minimum and maximum values of the second (generalized) coordinate for 2D and 3D grids. TypeBc5 and TypeBc6 describe the boundaries at the minimum and maximum values of the third (generalized) coordinate for 3D grids.

Possible values:

```
coupled       - set from coupling with another component
periodic      - periodic
float         - zero gradient for all variables except:
                Phi=0 set for the scalar in hyperbolic div B control (see #HYPERBOLICDIVB)
                radiative outflow boundaries are applied for radiation energy densities
outflow       - same as 'float' but the pressure is set to pOutflow from #OUTFLOWPRESSURE
reflect       - reflective (anti-symmetric for the normal components of V and B,
                            symmetric for all other variables)
linetied      - symmetric for density, anti-symmetric for momentum, float all others
fixed         - fixed solarwind values, total B is set
fixedB1       - fixed solarwind values, B1 is set
inflow/vary   - time dependent boundary based on solar wind input file (#SOLARWINDFILE)
shear         - sheared (intended for shock tube problem only)
ihbuffer      - values obtained from the IH component (this is set automatically)
none          - do not change ghost cells. This is useful if the outer boundary is not used.
fieldlinethreads - threaded magnetic field BC for AWSOM-R solar model
user          - user defined
```

Here are some tips for spherical grids. If the box defined in the #GRID command is completely inside the spherical grid (defined by #LIMITRADIUS) then the outer cell boundary at rMax is not used. If a "body" is used (see #BODY command) with a radius larger or equal than the minimum radius of the spherical grid (defined by #LIMITRADIUS) then the cell boundary at rMin is not used. On the other hand, if the box defined by #GRID is completely outside the spherical grid then the rMax cell boundary is used, and if there is no body defined, or if it has smaller radius than rMin, then the cell boundary at rMin is used. One can mix cell and face based boundaries. For example the xMin defined by #GRID may cut through the spherical grid, while the xMax ... zMax may be outside. This can be used to define inflow at the face based boundary at xMin using the #BOXBOUNDARY command, while the cell boundaries at the rMax boundary can be set to outflow using the #OUTERBOUNDARY command.

The default values are 'none' on all sides.

#BOXBOUNDARY command

```
#BOXBOUNDARY
outflow                 TypeBcXmin
inflow                  TypeBcXmax
float                   TypeBcYmin (only read in 2D and 3D)
float                   TypeBcYmax (only read in 2D and 3D)
float                   TypeBcZmin (only read in 3D)
float                   TypeBcZmax (only read in 3D)
```

This command defines how the face boundary values are set at the brick-shaped box cut out of a generalized coordinate grid. Normally this command should not be used for a Cartesian grid, but it is still allowed. The size of the box is defined by the xMin ... zMax parameters of the #GRID command.

General notes: face based boundary conditions are 1st order accurate in general, while cell based boundary conditions can be 2nd order accurate. Sometimes, however, it is easier to define a face value than the state of the ghost cells that are outside the computational domain. In our current implementation cell based boundaries can be used only at the outer edges of the grid.

On the other hand, face based boundaries can be applied anywhere. For a face boundary each cell center is marked as either physical or boundary cell, and the boundary conditions are applied at cell faces between a physical and a boundary cell center. The actual boundary will be ragged (along the cell faces) and this can in fact cause numerical problems. For supersonic outflow, the dot product of the face normal and the flow velocity should be positive, for inflow it should be negative.

The outer boundaries have to be face based if a brick-shaped computational domain is cut out from the sphere/cylinder (see the #LIMITRADIUS and #GRID commands) because the boundary is not aligned with the grid boundaries. If the computational domain is the full sphere/cylinder, then cell based boundaries can be used (see #OUTERBOUNDARY).

Possible values:

```
float          - zero gradient for all variables except:
                 Phi=0 set for the scalar in hyperbolic div B control (see #HYPERBOLICDIVB)
                 radiative outflow boundaries are applied for radiation energy densities
outflow        - same as 'float' but the pressure is set to pOutflow from #OUTFLOWPRESSURE
reflect        - reflect the normal component of B1, reflect the full velocity vector
reflectb       - reflect the normal component of full B, reflect the full velocity vector
reflectall     - reflect the normal component of B1 and velocity, symmetric for all other
linetied       - reflective for velocity, float for all others
fixed          - fixed values (set by #SOLARWIND or #BOUNDARYSTATE), total B is set
fixedB1        - fixed values (set by #SOLARWIND or #BOUNDARYSTATE), B1 is set
zeroB1         - B1 is reflected, all other variables float
inflow/vary    - time dependent boundary based on solar wind input file (#SOLARWINDFILE)
user           - user defined
```

There are no default values. User must set face boundary type if a box is cut out of a non-Cartesian grid.

#BOUNDARYSTATE command

```
#BOUNDARYSTATE
body1 1 2 xminbox       StringBoundary
1.0                     BoundaryStateDim_V Rho
1.0                     BoundaryStateDim_V Ux
1.0                     BoundaryStateDim_V Uy
1.0                     BoundaryStateDim_V Uz
0.0                     BoundaryStateDim_V Bx
0.0                     BoundaryStateDim_V By
```

```
0.0                     BoundaryStateDim_V Bz
0.0                     BoundaryStateDim_V Hyp
1.0                     BoundaryStateDim_V P
```

This command sets the primitive variables BoundaryState_V at one or more boundaries. The first parameter StringBoundary contains a space separated list of the names or indexes of the desired boundaries to be set. Both face and cell type boundaries can be listed.

The BoundaryStateDim_V are the nVar primitive variables used at the boundary in the order defined in the equation module ModEquation. The values are given in I/O units (see #IOUNITS command).

All boundaries can be identified with strings. Some boundaries can also be idenitified with an index between -3 and 6. Possible idenitfiers that can be listed in StringBoundary:

```
solid, -3       - solid face boundary            (#SOLIDSTATE)
body2, -2       - second body face boundary      (#INNERBOUNDARY, #SECONDBODY)
body1, -1       - first body face boundary       (#INNERBOUNDARY, #BODY)
extra,  0       - extra face boundary            (#EXTRABOUNDARY)
xminbox         - min x coordinate face boundary (#BOXBOUNDARY, #GRID)
xmaxbox         - max x coordinate face boundary
yminbox         - min y coordinate face boundary
ymaxbox         - max y coordinate face boundary
zminbox         - min z coordinate face boundary
zmaxbox         - max z coordinate face boundary
coord1min, 1    - min 1st (gen. coord.) cell boundary (#OUTERBOUNDARY)
coord1max, 2    - max 1st (gen. coord.) cell boundary (#GRIDGEOMETRY)
coord2min, 3    - min 2nd (gen. coord.) cell boundary (#LIMITRADIUS)
coord2max, 4    - max 2nd (gen. coord.) cell boundary (#GRIDGEOMETRYLIMIT)
coord3min, 5    - min 3rd (gen. coord.) cell boundary
coord3max 6     - max 3rd (gen. coord.) cell boundary
```

For each boundary name/index the commands controlling the boundary are shown on the right. Note that for Cartesian grids the outer boundaries are always cell based and the box boundary cannot be used. For non-cartesian grids the cell based outer boundaries refer to the edges of the domain given in generalized coordinates (for example rMin or rMax), while the face based box boundaries refer to a box cut out of the non-cartesian grid at the values xMin ... zMax given in the #GRID command.

There are no default values. The boundary name(s)/index(es) and primitive state values must be given.

#SOLIDSTATE command

```
#SOLIDSTATE
F                       UseSolidState (rest read if true)
user                    TypeBcSolid
sphere                  TypeSolidGeometry
1.0                     rSolid
5e-3                    SolidLimitDt
```

This command sets the solid boundary parameters. Solid boundary is one type of face boundary. Currently it works only for a sphere geometry with radius rSolid. In local time stepping mode the timestep inside the solid body is set to SolidLimitDt.

Default is UseSolidState=.false.

#OUTFLOWPRESSURE command

```
#OUTFLOWPRESSURE
```

```
T                       UseOutflowPressure
1e5                     pOutflowSi (read if UseOutflowPressure is true)
```

Set pressure for "outflow" boundary condition. This matters for subsonic outflow. Default is UseOutflowPressure=.false.

#INNERBOUNDARY command

```
#INNERBOUNDARY
ionosphere              TypeBcBody
ionosphere              TypeBcBody2  !read only if UseBody2=T
```

TypeBcBody determines the boundary conditions at the spherical surface of the inner body when these are described with face boundary conditions. For Cartesian grids this is always the case, because the spherical surface is not aligned with the grid blocks, so a ghost cell based boundary condition is not possible. For spherical grids, however, both the cell and face based boundary conditions can be used depending on the combination of commands. If face based boundary is used then the boundary condition at the body surface is determined here as TypeBcBody; if cell based boundary is used then the boundary condition at the body surface is determined by the TypeBc1 parameter of the #OUTERBOUNDARY command.

TypeBcBody2 is only read if the second body is used (see the #SECONDBODY command that has to occur BEFORE this command). The second body can be anywhere in the computational domain, so its sperical surface is never aligned with the grid block boundaries, consequently only face boundary conditions can be applied which is controlled by this command. It can have the same types as TypeBcBody, although not all those options are meaningful.

Possible values for TypeBcBody are:

```
'reflect'          - reflect all components of velocity relative to corotation,
                     reflect the normal component of B1, other variables float
'reflectb'         - same as reflect, but the normal component of full B is reflected.
'reflectall'       - reflect the normal component of B1 and all velocities.
                     This is the perfectly conducting sphere. B0 should be 0.
'float'            - float all variables
'outflow'          - same as 'float' but the pressure is set to pOutflow from #OUTFLOWPRESSU
'fixed'            - use initial solar wind values. Total B is set to solar wind B.
'fixedb1'          - use initial solar wind values. B1 is set is to solar wind B.
'inflow/vary'      - set the solar wind values. Total B is set to solar wind B.
'ionosphere'       - reflect velocity relative to corotation + ionosphere ExB drift
                     float B, fix rho, float P
'ionosphererefloat/linetied' - same as ionosphere but density floats too
'ionosphereoutflow' - same as ionosphere but an empirical outflow formula
                     is applied above 55 degrees latitude.  See #OUTFLOWCRITERIA for more in
'polarwind'        - same as ionosphere, but in the polar region use
                     the density and velocity from PW component if coupled,
                     or apply values read from the #POLARBOUNDARY command
'buffergrid'       - IH(OH) component obtains inner boundary from the SC(IH)
                     component, through a buffer grid. The buffer grid is set
                     by the #BUFFERGRID or #HELIOBUFFERGRID commands. In
                     SC/GM coipling the second body BC is implemented via the
                     buffer grid filled in from GM, for exoplanet orbiting in
                     the stellar corona.
'user'             - user defined
```

For 'ionosphere' and 'ionosphererefloat' types and a coupled GM-IE run, the velocity at the inner boundary is determined by the ionosphere model.

The 'absorb' inner BC only works with #ROTATION false.

The boundary condition on Br can be changed with the #MAGNETICINNERBOUNDARY command.

For the second body TypeBcBody2 can have the following values: 'absorb', 'reflect', 'reflectb', 'reflectall', 'float', 'ionosphere', 'ionosphererefloat/linetied', however, the corotation and ionospheric drift velocities are zero for the second body.

Default value for TypeBcBody is 'none' for the GM, EE, SC, IH and OH components, so the inner boundary must be set by this command except the cell boundary for spherical coordinates case. Default value for TypeBcBody2 is 'none'.

#ELECTRONPRESSURERATIO command

```
#ELECTRONPRESSURERATIO
1.0             ElectronTemperatureRatio
1/7.8           InnerBcPeRatio
```

The ElectronTemperatureRatio is the same as the last parameter of the #PLASMA command. It is used to set the electron pressure at the outer boundary and for the initial condition.

The InnerBcPeRatio is the electron pressure ratio assumed at or near the inner boundary. This parameter is used in two different ways. When BATSRUS solves for the electron pressure as a separate variable, it is used for inner boundary conditions for Pe. When 'ionosphere', 'polarwind' or 'ionosphereoutflow' inner boundary is used, the electron pressure is set to be float at the inner boundary by default. In order to avoid extremely low electron pressure in the inner magnetosphere, this command ensures the ratio between the electron pressure and ion pressure at the inner boundary is at least InnerBcPeRatio.

When BATSRUS does not solve for electron pressure, the InnerBcPeRatio constant is used in the coupling with RCM to set the electron pressure passed to RCM to InnerBcPeRatio times the pressure of the first ion fluid.

The default values are ElectronTemperatureRatio=0 and InnerBcPeRatio=1/7.8.

#OUTFLOWCRITERIA command

```
#OUTFLOWCRITERIA
-1              OutflowVelocity [km/s]
2.142E7         FluxAlpha
1.265           FluxBeta
```

This command configures the empirical outflow relationship that is activated via the #INNERBOUNDARY command when TypeBcBody is set to 'ionosphereoutflow'. The empirical relationship is based on the work of *Strangeway et al., 2005*:

$$F_{O^+} = \alpha S_{\parallel}^{\beta} \tag{3.1}$$

...where F_{O^+} is the local upflowing oxygen flux, $S_{\parallel}$ is the local field-aligned Poynting flux, and α and β are fitting coefficients based on observations from the FAST spacecraft. Default values for α and β, shown above, are taken directly from *Strangeway et al., 2005*. In BATS-R-US, Poynting flux is taken from coupling with the IE module.

The OutflowVelocity paramter sets the radial velocity of the outflow, which also controls how flux is converted into number density:

$$n_{O^+} = F_{O^+}/U_R \tag{3.2}$$

If OutflowVelocity is negative, the radial velocity of oxygen is set using the energy of the fluid as obtained via the local Joule heating and field-aligned- current conditions (obtained via IE coupling). This is the default behavior.

For more information on the empirical relationship for flux, see Strangeway, R., Ergun, J. R. E., Su, Y. J., Carlson, C. W., & Elphic, R. C. (2005). Factors controlling ionospheric outflows as observed at intermediate altitudes. Journal of Geophysical Research, 110(A3), A03221. http://doi.org/10.1029/2004JA010829

#MAGNETICINNERBOUNDARY command

```
#MAGNETICINNERBOUNDARY
-1.0            B1rCoef
```

The radial component of B1 on the ghost face is set as B1rGhost = B1rCoef*B1rTrue at the inner boundary. B1rCoef=-1 corresponds to a reflective boundary, while B1rCoef=1 is a floating (zero gradient) boundary. Any value between -1 and 1 is possible. Using floating condition, however, will not work well for strong storms, as there is no mechanism to restore the dipole after the storm. Reflective will recover the dipole, but it may result in some less stable behavior. The optimal value may be problem dependent.

The default value corresponding to reflection is shown above.

#BUFFERGRID command

```
#BUFFERGRID
2       nRBuff
90      nLonBuff
45      nLatBuff
19.0    rBuffMin
21      rBuffMax
0.0     LonBuffMin
360.0   LonBuffMax
-90.0   LatBuffMin
90.0    LatBuffMax
```

Define the radius, angular extent and the grid resolution of the uniform spherical buffer grid used to pass information between two coupled components running BATSRUS.

The parameters nRBuff, nPhiBuff and nThetaBuff determine the number of points in the radial, azimuthal and latitudinal directions, respectively.

The parameters rBuffMin and rBuffMax determine the inner and outer radii of the spherical shell.

PhiBuffMin, PhiBuffMax, LatBuffMin and LatBuffMax determine the limits (in degrees) of the buffer grid in the azimuthal and latitudinal directions.

When used to pass information from the SC(IH) component to the IH(OH) component, the entire shperical shell should be used (alternativly, use the #HELIOBUFFERGRID command), but in certain application only a part of the shell may be needed. The buffer should be placed in a region where the two components overlap, and the grid resolution should be similar to the grid resolution of the coarser of the two component grids. This command can only be used in the first session by the IH(OH) component. The buffer grid will only be used if 'buffergrid' is chosen for TypeBcBody in the #INNERBOUNDARY command of the target component. Default values are shown above.

#BUFFERBODY2 command

```
#BUFFERBODY2
2       nRBuff
90      nLonBuff
45      nLatBuff
19.0    rBuffMin
21      rBuffMax
0.0     LonBuffMin
360.0   LonBuffMax
-90.0   LatBuffMin
90.0    LatBuffMax
```

Define the radius, angular extent and the grid resolution of the uniform spherical buffer grid used to pass information between two coupled components running BATSRUS. In contrast with the #BUFFERGRID command, the grid is concetric with the second body, not heliocentric.

The parameters nRBuff, nPhiBuff and nThetaBuff determine the number of points in the radial, azimuthal and latitudinal directions, respectively.

The parameters rBuffMin and rBuffMax determine the inner and outer radii of the spherical shell.

PhiBuffMin, PhiBuffMax, LatBuffMin and LatBuffMax determine the limits (in degrees) of the buffer grid in the azimuthal and latitudinal directions.

When used to pass information from the GM component to the SC component, the entire shperical shell should be used. The buffer should be placed in a region where the two components overlap, and the grid resolution should be similar to the grid resolution of the coarser of the two component grids. This command can only be used in the first session by the SC component. The buffer grid will only be used if 'buffergrid' is chosen for TypeBcBody2 in the #INNERBOUNDARY command of the target component. Default values are shown above.

#EXTRABOUNDARY command

```
#EXTRABOUNDARY
T               UseExtraBoundary
user            TypeExtraBoundary
```

If UseExtraBoundary is true, the user can define an extra face boundary condition in the user files. The location of this boundary is defined in the user_set_boundary_cells routine, while the boundary condition itself is implemented into the user_set_face_boundary. The extra boundary has index ExtraBc_=0. The TypeExtraBoundary parameter can be used to select from multiple boundary conditions implemented in the user module.

#POLARBOUNDARY command

```
#POLARBOUNDARY
20.0            PolarNDim [amu/cc] for fluid 1
100000.0        PolarTDim [K]      for fluid 1
1.0             PolarUDim [km/s]   for fluid 1
2.0             PolarNDim [amu/cc] for fluid 2
-1.0            PolarTDim [K]      for fluid 2
1.5             PolarUDim [km/s]   for fluid 2
75.0            PolarLatitude [deg]
```

This command defines the boundary conditions in the polar region. The number density, temperature and velocity can be given (for all fluids in multifluid calculations). Negative temperature value sets the pressure float. This mimics polar wind like inner boundary conditions when GM is not coupled with the PW component. The PolarLatitude parameter determines the latitudinal extent of the polar boundary where the outflow is defined.

#CPCPBOUNDARY command

```
#CPCPBOUNDARY
T                       UseCpcpBc  (rest is read if true)
28.0                    Rho0Cpcp   [amu/cc] for 1st ion fluid/species
0.1                     RhoPerCpcp [amu/cc / kV]
8.0                     Rho0Cpcp   [amu/cc] for 2nd ion fluid/species
0.3                     RhoPerCpcp [amu/cc / kV]
```

NOTE: For this feature the inner boundary type has to be "ionosphere" and the GM and IE components have to be coupled together.

If UseCpcpBc is true, the ion mass densities at the inner boundary will depend on the cross polar cap potential (CPCP) in a linear fashion:

RhoBc = Rho0Cpcp[i] + RhoPerCpcp[i] * Cpcp

where i is the index of the ion fluid or ion species, RhoBc and Rho0Cpcp are in I/O units (typically amu/cc), the Cpcp is given in [kV], and the RhoPerCpcp factor is in density units per kV. The Cpcp is the average of the northern and southern CPCPs. The example shows some reasonable values for hydrogen and oxygen. For CPCP = 0 kV RhoBc[H+] = 28 amu/cc and RhoBc[O+] = 8 amu/cc, while for CPCP = 400 kV RhoBc[H+] = 68 amu/cc and RhoBc[O+] = 128 amu/cc.

By default the density at the inner boundary is determined by the body density given in the #BODY (same as #MAGNETOSPHERE) command.

#YOUNGBOUNDARY command

```
#YOUNGBOUNDARY
T                       UseYoungBc  (rest is read if true)
150.0                   F107Young
```

NOTE: For this feature the inner boundary type has to be ”ionosphere” and the GM and IE components have to be coupled together. Kp must be calculated via #GEOMAGINDICES.

This option sets the mass density via the Young et al. 1982 empirical relationship for composition. It uses Kp (calculated by GM/BATSRUS) and F10.7 flux (given as command argument) to determine the ratio of O+ to H+. The mass density of the inner boundary will be adjusted to match this ratio. The total number density is taken as constant from the #BODY command.

#OHBOUNDARY command

```
#OHBOUNDARY
T                       UseOhNeutralBc (rest of parameters are read if true)
0.05                    RhoNeuFactor
1.0                     uNeuFactor
1.E-2                   RhoNeuFactor for Ne2
0.2                     uNeuFactor for Ne2
```

Read in density and velocity factors for each neutral fluid. These factors are used to set the boundary conditions for the neutral fluids in the outer heliosphere component. If the flow points outward from the domain, the boundary condition is floating. If it points inward, the density, pressure and velocity are set as RhoNeuFactor*Rho1, RhoNeuFactor*P1 and uNeuFactor*u1, where Rho1, p1, u1 are the density, pressure and velocity of the first fluid.

Default is UseOhNeutralBc false.

#OHNEUTRALS command

```
#OHNEUTRALS
0.18                    RhoNeutralsISW [amu/cc]
6519.0                  TNeutralsISW   [K]
26.3                    UxNeutralsISW  [km/s]
0.3                     UyNeutralsISW  [km/s]
-2.3                    UzNeutralsISW  [km/s]
1.0                     mNeutral       [amu]
```

Upstream boundary conditions for the neutrals in outer heliosphere component. The density, temperature and velocity components are given by the first five parameters. The mNeutral parameter defines the mass of the neutrals in proton mass. There are no default values, so this command is required for the OH component.

3.8.6 Grid geometry

#GRIDBLOCK command

```
#GRIDBLOCK
100             MaxBlock (per processor)

#GRIDBLOCKALL
4000            MaxBlock (for the whole simulation)
```

This command can be and should be used in the first session of BATSRUS. For a restarted run, the #CHECKGRIDSIZE command in the restart header file also sets MaxBlock, but that can and should be overwritten if the grid is expected to change due to an AMR.

Set the maximum number of grid blocks either per processor (#GRIDBLOCK) or in total for the whole simulation (#GRIDBLOCKALL). Typically it is better to set the total number so the code can run on arbitrary number of CPUs. It is a good idea to set these values to be larger than but close to the actual number of blocks used during the run to minimize memory use and improve performance.

The default value is 10 blocks per processor, but it is not recommended to rely on the default setting.

#GRIDBLOCKIMPL command

```
#GRIDBLOCKIMPL
100             MaxBlockImpl per processor

#GRIDBLOCKIMPLALL
1000            MaxBlockImpl on all processors
```

This command can be used when or before the part implicit scheme is switched on.

Set the maximum number of grid blocks advanced by the part-implicit method (see #IMPLICIT) either per processor (#MAXBLOCKIMPL) or in total (#MAXBLOCKIMPLALL). Note that MaxBlockImpl cannot be more than MaxBlock, but it can be smaller to save memory.

The default is that all blocks are implicit if the part-implicit scheme is used.

#GRID command

```
#GRID
2                       nRootBlock1
1                       nRootBlock2
1                       nRootBlock3
-224.                   xMin
 32.                    xMax
-64.                    yMin
 64.                    yMax
-64.                    zMin
 64.                    zMax
```

The nRootBlock1, nRootBlock2 and nRootBlock3 parameters define the number of blocks of the base grid, i.e. the roots of the octree. By varying these parameters, one can setup a grid which is elongated in some direction. The xMin, ..., zMax parameters define a brick shaped computational domain. An inner boundary may be cut out from the domain with the #BODY and/or #LIMITRADIUS commands. It is also possible to define a spherical, cylindrical computational domain using the #GRIDGEOMETRY and the #LIMITRADIUS commands.

There are no default values, the grid size must always be given in the first session (even if the component is switched off in the first session!).

#GRIDSYMMETRY command

```
#GRIDSYMMETRY
F                       IsMirrorX
T                       IsMirrorY
T                       IsMirrorZ
```

For symmetric test problems one can model only a part of the computational domain. Providing the symmetry directions with this command allows the proper calculation of line-of-sight plots.

#COORDSYSTEM command

```
#COORDSYSTEM
GSM                     TypeCoordSystem
```

TypeCoordSystem defines the coordinate system for the component. The coordinate systems are defined in share/Library/src/CON_axes. Here we provide general suggestions.

For GM (Global Magnetopshere) the default coordinate system is "GSM" with the X axis pointing towards the Sun, and the (moving) magnetic axis contained in the X-Z plane. The inertial forces are neglected. The essentially inertial "GSE" system is also available, but it is not fully tested.

For SC (Solar Corona) one should always use the corotating HGR system to get an accurate solution even for complicated active regions. Using an inertial frame would result in huge numerical errors near the Sun.

For time accurate IH solutions (e.g. CME propagation) one should use the inertial HGI system so the grid can be refined along the Sun-Earth line. To obtain a steady state initial condition, the corotating HGC system can be used which is aligned with the HGI system for the initial time of the simuation (see #STARTTIME command). When the run is switched to time accurate mode, the coordinate system should be switched to HGI. The necessary transformation of the velocity (adding the corotating velocity) is automatically performed.

For quiet steady state IH solutions the HGR system can be used. Note however that the corotating systems may not work well if the IH domain is extended way beyond 1AU, becasue the boundary condition can become inflow type at the corners of a Cartesian domain. In this case the inertial HGI system should be used in time accurate mode even for obtaining the initial state.

For OH one should always use the inertial HGI system. A rotating frame would have extremeley fast rotational speeds.

Note that the HGR and HGI systems can be rotated with a fixed angle using the #ROTATEHGR and #ROTATEHGI commands. This can be used to align the interesting plane of the simulation with the grid.

The default is component dependent: "GSM" for GM, "HGR" for SC, and "HGI" for IH and OH.

#ROTATEHGR command

```
#ROTATEHGR
145.6                   dLongitudeHgr [deg]
```

Rotate the HGR system by dLongitudeHgr degrees around the Z axis. A negative value is interpreted as an offset angle which moves the planet into the -X, Z plane (so roughly towards the -X axis). Default value is 0, i.e. the true HGR system is used.

#ROTATEHGI command

```
#ROTATEHGI
-1.0                    dLongitudeHgi [deg]
```

Rotate the HGI and the related rotating HGC systems by dLongitudeHgi degrees around the Z axis. A negative value is interpreted as an offset angle which moves the planet into the -X, Z plane (so roughly towards the -X axis). Default value is 0, i.e. the true HGI system is used.

#GRIDGEOMETRY command

```
#GRIDGEOMETRY
spherical_genr                  TypeGeometry
Param/CORONA/grid_TR.dat        NameGridFile (read if TypeGeometry is _genr)

#GRIDGEOMETRY
roundcube                       TypeGeometry
200.0                           rRound0  ! only read for roundcube geometry
320.0                           rRound1  ! only read for roundcube geometry
```

Note: The #LIMITRADIUS command can be used to set the radial extent of the cylindrical, spherical and roundcube grids. The #GRIDGEOMETRYLIMIT command provides even more control.

This command determines the geometry of the grid. Possible values are Cartesian, rotated Cartesian, RZ geometry, cylindrical, spherical and roundcube. The cylindrical and spherical grids can have logarithmic (cylindrical_lnr and spherical_lnr) or arbitrarily stretched (spherical_genr, cylindrical_genr) radial coordinates. For the latter case the radial stretching is read from the NameGridFile file. The roundcube geometry is a radially stretched Cartesian grid. The stretching is controlled by the rRound0 and rRound1 parameters.

The "RZ" geometry is a 2D grid with axial symmetry. In our particular implementation the "X" axis is the axis of symmetry, and the "Y" axis is used for the radial direction.

The spherical coordinates are ordered as r, longitude, latitude. The longitude is between 0 and 360 degrees, the latitude is between -90 and 90 degrees. The cylindrical coordinates are r, phi, z with phi between 0 and 360 degrees.

The roundcube grid can be used to make the inner or outer boundary spherical without a singularity. It works in 2D and 3D. The rRound0 parameter indicates the distance where no stretching is applied, so the grid is Cartesian. The rRound1 parameter indicates the distance along X, Y and Z on the original grid where full stretching is applied, so the grid becomes round and the grid cells will lie on a circle in 2D, or a spherical surface in 3D. When rRound0 is less than rRound1, the grid is Cartesian up to rRound0. Outside rRound0 the grid is stretched outward so that it becomes perfectly round at a radius of rRound1*sqrt(2) in 2D and rRound1*sqrt(3) in 3D, and it remains round all the way to the outer boundary. For this case the transformation does not affect the main diagonals and the maximum stretching is applied along the main axes. To reach the perfectly round shape at the outer boundary, the xMin ... zMax parameters of the #GRID command should be equal or larger than sqrt(nDim)*rRound1. If rRound0 is larger than rRound1 then the grid is contracted inwards. At the origin there is no distortion. Moving outward the distortion is increased so that at rRound1 the grid becomes round. From rRound1 to rRound0 the grid becomes Cartesian again. This can be useful to create a sphere shaped inner boundary without any singularities. For this case the grid is not contracted along the main axes and it is maximally contracted along the diagonals.

The rotated Cartesian geometry can be used for debugging the generalized coordinate code. It allows setting up a Cartesian test on a rotated generalized coordinate grid. The rotation is around the Z axis with an angle alpha that has sin(alpha)=0.6 and cos(alpha)=0.8 for sake of getting nice rational numbers. The PostIDL code unrotates the grid and the vector variables so it can be directly compared with a Cartesian simulation. The initial conditions and the boundary conditions, however, are not rotated automatically (yet), so they require some attention. Note that only the first order schemes (see #SCHEME) will produce identical results on rotated and non-rotated grids because nonlinear limiters produce different face values for the vector components.

The default is Cartesian geometry.

#GRIDGEOMETRYLIMIT command

```
#GRIDGEOMETRYLIMIT
spherical                       TypeGeometry
1.0                             Coord1Min Radius
24.0                            Coord1Max
0.0                             Coord2Min Longitude
360.0                           Coord2Max
```

```
-90.0                           Coord3Min Latitude
90.0                            Coord3Max
```

The #GRIDGEOMETRYLIMIT command is similar to the #GRIDGEOMETRY command, but provides in addition the flexibility to change the limits of the generalized coordinates. This allows to construct grids such as a spherical or cylindrical wedge. The radial limits are given in true radius even if the radial coordinate is logarithmic or stretched. For spherical and cylindrical grids the angle limits are provided in degrees.

Default is Cartesian grid.

#LIMITRADIUS command

```
#LIMITRADIUS
 10.0                   rMin
100.0                   rMax
```

Note: the #GRIDGEOMETRYLIMIT command provides even more control.

This command allows setting the mimimum and maximum radial extent of the grid. Setting rMin to a positive value excludes the origin of a spherical grid, or the axis of the cylindrical grid.

The rMax parameter can be used to choose a spherical or cylinrical domain instead of the brick defined by the #GRID. To achieve this, rMax has to be set to a radius that fits inside the brick defined by #GRID.

By default the inner radius is set to the radius of the inner body if it is present (see the #BODY command) and the outer radius is set to the largest radial distance of the eight corners of the domain defined by the #GRID command. If there is no inner body, the default inner radius is set to 0.0 for regular spherical and cylindrical grids, and to 1.0 for logarithmic and stretched radius grids.

#UNIFORMAXIS command

```
#UNIFORMAXIS
T                       UseUniformAxis
```

This command can only be used in the first session. If UseUniformAxis is true, there can be no resolution change AROUND the axis of a spherical or cylindrical grid. This is required by the supercell algorithm that can be activated by the #FIXAXIS command. Note that there can still be resolution changes ALONG the axis.

If UseUniformAxis is false, the AMR can produce resolution changes around the axis of the grid. The super-cell algorithm cannot be used. For restarted runs the false setting has to be repeated in the PARAM.in file used for the restart.

The default is UseUniformAxis=T.

#FIXAXIS command

```
#FIXAXIS
T                       DoFixAxis
5.0                     rFixAxis
1.5                     r2FixAxis
```

The computational cells become very small near the symmetry axis of a spherical or cylindrical grid.

When DoFixAxis is true, the cells around the pole are merged into one 'supercell' for the blocks that are (partially) inside radius rFixAxis. For blocks within r2FixAxis, the radius of the supercell is 2 normal cells. Merging the small cells allows larger time steps in time accurate runs: about a factor of 2 if only rFixAxis is used, and around factor of 3 if r2FixAxis is also used.

Note that the super-cell algorithm requires that there is no resolution change around the axis in the phi direction. See the #UNIFORMAXIS command for more discussion.

Default is false for DoFixAxis.

#COARSEAXIS command

```
#COARSEAXIS
T                       UseCoarseAxis
3                       nCoarseLayer
```

The computational cells become very small near the symmetry axis of a spherical or grid.

When UseCoarseAxis is true, the cells around the pole are merged into pairs, if nCoarseLayer=1. If nCoarseLayer=2, then around the pole each 4 cells are merged and in the second (from the pole) layer each 2 cells are merged. To achieve this, nJ size parameter of the spherical gird should be a multiple of 4. If nCoarseLayer=3, then around the pole each 8 cells are merged, in the second (from the pole) layer each 4 cells are merged, and in the third (from the pole) layer each 2 cells are merged. To achieve this, nJ size parameter of the spherical gird should be a multiple of 8.

Default is false for UseCoarseAxis.

3.8.7 Initial time

#STARTTIME command

```
#STARTTIME
2000                    iYear
3                       iMonth
21                      iDay
10                      iHour
45                      iMinute
0                       iSecond
```

The #STARTTIME command sets the initial date and time for the simulation in Universal Time (UT) in stand alone mode. This time is stored in the BATSRUS restart header file. It can be overwritten with a subsequent #STARTTIME comand if necessary.

In the SWMF this command checks BATSRUS start time against the SWMF start time and warns if the difference exceeds 1 millisecond.

The default values are shown above. This is a date and time when both the rotational and the magnetic axes have approximately zero tilt towards the Sun.

#TIMESIMULATION command

```
#TIMESIMULATION
1 hour                  tSimulation [sec]
```

The tSimulation variable contains the simulation time in seconds relative to the initial time set by the #STARTTIME command. The #TIMESIMULATION command and tSimulation are saved into the restart header file, so the simulation can be continued from the same time. This value can be overwritten by a subsequent #TIMESIMULATION command if necessary.

In SWMF the BATSRUS time is checked against the global SWMF simulation time.

The default value is tSimulation=0.

#NSTEP command

```
#NSTEP
100                     nStep
```

Set nStep for the component. Typically used in the restart.H header file. Generally it is not inserted in a PARAM.in file by the user, except when the number of steps are reset for extremely long runs, such as the operational run at NOAA SWPC, to avoid integer overflow.

The default is nStep=0 as the starting time step with no restart.

#NPREVIOUS command

```
#NPREVIOUS
100                     nPrev
1.5                     DtPrev
```

This command should only occur in the restart.H header file. If it is present, it indicates that the restart file contains the state variables for the previous time step. nPrev is the time step number and DtPrev is the length of the previous time step in seconds. The previous time step is needed for a second order in time restart with the implicit scheme.

The default is that the command is not present and no previous time step is saved into the restart files.

3.8.8 Time integration

#TIMESTEPPING command

```
#TIMESTEPPING
1                       nStage
0.4                     CflExpl

#RUNGEKUTTA
2                       nStage
0.8                     CflExpl

#RK
4                       nStage
1.3                     CflExpl
```

These commands set the parameters for time integration. For explicit time integration nStage is the number of stages. Setting nStage=1 selects a temporally first order forward Euler scheme. The nStage=2 corresponds to a temporally second order scheme. The #TIMESTEPPING command uses half time step for the first stage, and full time step for the second stage. The #RUNGEKUTTA or #RK commands select a TVD Runge-Kutta scheme that employs full time step in both stages and then takes their average. The nStage=3 selects a 3rd order TVD Runge-Kutta scheme. The nStage=4 selects the classical 4th order Runge-Kutta scheme. These temporally high order options are useful in combination with spatially higher order schemes (to be implemented).

For implicit time stepping nStage=2 corresponds to the BDF2 (Backward Differene Formula 2) scheme that uses the previous time step to make the scheme 2nd order accurate in time.

For explicit time stepping the CPU time is proportional to the number of stages. In time accurate runs the 1-stage explicit time stepping scheme may work reasonably well with second order spatial discretization, especially if the time step is limited to a very small value. Using a one stage scheme can speed up the code by a factor of two with little compromise in accuracy.

For local time stepping (steady state mode) one should always use the 2-stage scheme with 2-nd order spatial accuracy to avoid oscillations (or use the 1-stage scheme with CflExpl=0.4).

For implicit scheme the second order BDF2 scheme is more accurate but not more expensive than the first order backward Euler scheme, so it is a good idea to use nStage=nOrder (or at least nStage=3 for high order schemes).

To achieve consistency between the spatial and temporal orders of accuracy, the #SCHEME command always sets nStage to be the same as nOrder except for 5th order scheme, which sets nStage=3. The #TIMESTEPPING (or #RUNGEKUTTA or #RK) command can be used AFTER the #SCHEME command to overwrite nStage if required.

If the #SCHEME command is not used, then the defaults are nStage=2 with the half-step predictor and CflExpl=0.8.

#USEFLIC command

```
#USEFLIC
T                       UseFlic
```

MHD scheme which works similarly to the hybrid one and can work together with hybrid. Requires nStage=3.

#PARTLOCALTIMESTEP command

```
#PARTLOCALTIMESTEP
1.1                     rLocalTimeStep
```

Use local time stepping inside radial distance rLocalTimeStep and time accurate mode in the rest of the domain. The global time step Dt is only limited by the cells outside rLocalTimeStep. Inside rLocalTimeStep each cell advances with the smaller of Dt and the locally stable time step. This method can speed up the calculation when near the inner boundary the solution is quasi-steady state.

Default value is rLocalTimeStep=-1, so the scheme is not used.

#TIMESTEPLIMIT command

```
#TIMESTEPLIMIT
T                       UseDtLimit
10.                     DtLimitDim [sec] (read if UseDtLimit is true)
```

If UseDtLimit is true, the local time step is limited to DtLimitDim in either steady state mode or time accurate mode. The only difference between running in steady state and time accurate is that the simulation time does not evolve in steady state mode.

Limiting the local time step in steady state mode can be useful to reach steady state with less violent transients.

For time accurate simulations this feature can be useful when in some stiff regions the local stable time step is very small, but the solution is in a quasi-steady state. If this is true, selecting a suitable value for DtLimitDim will evolve the solution in time accurate mode in the region where the stable time step is larger than DtLimitDim*Cfl, and it will iterate with the local time step in the stiff region. As long as the quasi-steady state can follow the time evolution with the local time step, the overall solution will be correct.

The limited time step approach is different from the part steady scheme (see #PARTSTEADY), which assumes a near perfect steady state in parts of the domain where the solution is not evolved at all. If the stiff region cannot keep up with the time evolution, then subcycling (see #LOCALTIMESTEP) or implicit time stepping (see #IMPLICIT) is needed.

The default is UseDtLimit false.

#FIXEDTIMESTEP command

```
#FIXEDTIMESTEP
T                       UseDtFixed
10.                     DtFixedDim [sec] (read if UseDtFixed is true)
```

The fixed time step is typically used with the implicit and partially implicit schemes in time accurate mode. The time step is set to DtFixedDim unless the time step control algorithm (see #TIMESTEPCONTROL or #UPDATECHECK) reduces the time step for the sake of robustness.

The fixed time step can also be used with explicit time stepping in time accurate mode for debugging as well as for convergence tests.

The fixed time step can not be used in steady state mode. See #TIMESTEPLIMIT if the purpose is to make transients smaller or solve part of the domain in time accurate mode.

The default is UseDtFixed false.

#PARTSTEADY command

```
#PARTSTEADY
T                       UsePartSteady
```

If UsePartSteady is true, the partially steady state algorithm is used. Only blocks which are changing or next to changing blocks are evolved. This scheme can speed up the calculation if part of the domain is in a numerical steady state. In steady state runs the code stops when a full steady state is achieved. The conditions for checking the numerical steady state are set by the #PARTSTEADYCRITERIA command.

See also the #LOCALTIMESTEP and #TIMESTEPLIMIT commands for related approaches.

Default value is UsePartSteady = .false.

#PARTSTEADYCRITERIA command

```
#PARTSTEADYCRITERIA
5               MinCheckVar
8               MaxCheckVar
0.001           RelativeEps(bx)
0.0001          AbsoluteEps(bx)
0.001           RelativeEps(by)
0.0001          AbsoluteEps(by)
0.001           RelativeEps(bz)
0.0001          AbsoluteEps(bz)
0.001           RelativeEps(p)
0.0001          AbsoluteEps(p)
```

The part steady scheme only evolves blocks which are changing, or neighbors of changing blocks. The scheme checks the neighbor blocks every time step if their state variable has changed significantly. This command allows the user to select the variables to be checked, and to set the relative and absolute limits for each variable. Only the state variables indexed from MinCheckVar to MaxCheckVar are checked. The change in the block is significant if

```
max(abs(State - StateOld)) / (RelativeEps*abs(State) + AbsoluteEps)
```

exceeds 1.0 for any of the checked variables in any cells of the block. (including body cells but excluding ghost cells). The RelativeEps variable determines the maximum ratio of the change and the norm of the old state. The AbsoluteEps variable is only needed if the old state is very close to zero. It should be set to a positive value which is much smaller than the typical significantly non-zero value of the variable.

Default values are such that all variables are checked with relative error 0.001 and absolute error 0.0001.

#POINTIMPLICIT command

```
#POINTIMPLICIT
T               UsePointImplicit
0.5             BetaPointImplicit (read if UsePointImplicit is true)
T               IsAsymmetric
T               DoNormalizeCell
```

Switches on or off the point implicit scheme. The BetaPointImplicit parameter (in the 0.5 to 1.0 range) determines the order of accuracy for a 2-stage scheme. If BetaPointImplicit=0.5 the point implicit scheme is second order accurate in time when used in a 2-stage scheme. Larger values may be more robust, but only first order accurate in time. For a 1-stage scheme or for local timestepping the BetaPointImplicit parameter is ignored and the coefficient is set to 1.

If the IsAsymmetric parameter is true, the numerical Jacobian is calculated with a one-sided (asymmetric) difference formula. Otherwise a two-sided symmetric difference is used. The latter is slower somewhat but more accurate.

If DoNormalizeCell is true, the normalization of variables (this is needed to make small perturbations for the calculation of numerical derivatives) is done cell-by-cell. The default is false, so normalization is done on a block-by-block basis.

For single-ion MHD the default is UsePointImplicit=.false. For multi-ion MHD the default values are UsePointImplicit=.true., BetaPointImplicit=1.0 and IsAsymmetric=.true.

#IMPLICIT command

```
#IMPLICIT
F               UsePointImplicit
F               UsePartImplicit
T               UseFullImplicit
100.0           CflImpl (read if UsePartImplicit or UseFullImplicit is true)
```

If UsePointImplicit=T is set, the source terms defined in the user module are evaluated with a point implicit scheme. See the #POINTIMPLICIT command for additional parameters (and another way of switching the point implicit scheme on).

If UsePartImplicit=T is set, the code uses the explicit/implicit scheme. This means that some of the grid blocks are advanced with explicit time stepping, while the rest is advance with implicit time stepping. See the #FIXEDTIMESTEP and #IMPLICITCRITERIA command on how the explicit and implicit blocks get selected.

If UseFullImplicit=T is set, the code uses a fully implicit time stepping scheme. This is usually more costly than the explicit/implicit scheme unless the whole computational domain requires implicit time stepping.

Note 1: If UseFullImplicit is true, UsePartImplicit and UsePointImplicit must be false.

Note 2: UsePartImplicit=T and UsePointImplicit=T may be used together: source terms are point implicit in the explicit blocks.

The ImplCFL parameter determines the CFL number used in the implicit blocks of the fully or partially implicit schemes. This is ignored if UseDtFixed=T is set in the #FIXEDTIMESTEP command.

Default is false for all logicals.

#SEMIIMPLICIT command

```
#SEMIIMPLICIT
T                       UseSemiImplicit
radiation               TypeSemiImplicit (read if UseSemiImplicit is true)
```

If UseSemiImplicit is true then most of the terms are evaluated explicitly, but some of them are evaluated implicitly.

The TypeSemiImplicit parameter determines which terms and corresponding variables are done semi-implicitly.

The ”radiation” option solves for the gray or multigroup diffusion energy density. For gray diffusion the internal energy and pressure is calculated in a point implicit manner. To use gray diffusion configure BATSRUS with Config.pl -nWave=1. To use the multi-group radiation set nWave larger than one.

The ”radiationsplit” option solves each radiation group separately. The energy exchange term is calculated point-implicitly. The internal energy is updated in a conservative way.

The ”radcond” option solves implicitly the radiation diffusion and electron heat conduction together with the radiation and internal energy densities being the unknowns.

The ”radcondsplit” option solves each radiation group and the electrons heat conduction separately.

The ”parcond” option solves for field aligned electron heat conduction only.

The ”cond” option solves for electron heat conduction only.

The ”resistivity” option solves for the magnetic field with dissipative and/or Hall resistivity. The ”resist” option does NOT solve Hall term with semi-implicit. The ”resisthall” option does NOT solve dissipative resistivity.

The default is UseSemiImplicit false.

3.8.9 Implicit parameters

#IMPLICITENERGY command

```
#IMPLICITENERGY
F                       UseImplicitEnergy
```

If UseImplicitEnergy is true, use the energy variable(s) as unknown(s) in the implicit scheme, otherwise use the pressure variables(s). Note that the explicit scheme that provides the right hand side of the implicit scheme may still be conservative, and thus the overall scheme can provide correct jump conditions across standing (or slowly moving) shocks. Away from shocks, using pressure as an implicit variable provides a more accurate and robust scheme than using the energy variable.

The default is UseImplicitEnergy=T for sake of backwards compatibility.

#IMPLICITCRITERIA command

```
#IMPLICITCRITERIA
r               TypeImplCrit (dt or r or test)
10.0            rImplicit    (only read for TypeImplCrit = r)
```

Both #IMPLICITCRITERIA and #STEPPINGCRITERIA are acceptable. Only effective if PartImplicit is true in a time accurate run. Default value is ImplCritType='dt'.

The options are

```
if     (TypeImplCrit =='dt'  ) then blocks with DtBLK .lt. DtFixed
elseif (TypeImplCrit =='r'   ) then blocks with rMinBLK .lt. rImplicit
elseif (TypeImplCrit =='test') then block iBlockTest on processor iProcTest
```

are handled with the implicit scheme. Here DtBlock is the time step allowed by the CFL condition for a given block, while rMinBLK is the smallest radial distance for all the cells in the block.

The iBlockTest and iProcTest can be defined in the #TESTIJK command.
DtFixed must be defined in the #FIXEDTIMESTEP command.

#PARTIMPL command

```
#PARTIMPLICIT
T               UsePartImplicit2
```

If UsePartImplicit2 is set to true, the explicit scheme is executed in all blocks before the implicit scheme is applied in the implicit blocks. This way the fluxes at the explicit/implicit interface are second order accurate, and the overall part implicit scheme will be fully second order in time. When this switch is false, the explicit/implicit interface fluxes are only first order accurate in time. A potential drawback of the second order scheme is that the explicit scheme may crash in the implicit blocks. This could be avoided with a more sophisticated implementation. There may also be a slight speed penalty, because the explicit scheme is applied in more blocks.

The default is UsePartImplicit2 = false for now, which is safe and backward compatible.

#IMPLSTEP command

```
#IMPLSTEP
0.5                     ImplCoeff
F                       UseBdf2
F                       UseSourceImpl
```

The ImplCoeff is the beta coefficient in front of the implicit terms for the two-level implicit scheme. The UseBdf2 parameter decides if the 3 level BDF2 scheme is used or a 2 level scheme. UseSourceImpl true means that the preconditioner should take point source terms into account.

For steady state run the default is the backward Euler scheme, which corresponds to ImplCoeff=1.0 and UsedBdf2=F. For second order time accurate run the default is UseBdf2=T, since BDF2 is a 3 level second order in time and stable implicit scheme. In both cases the default value for UseSourceImpl is false.

The default values can be overwritten with #IMPLSTEP, but only after the #TIMESTEPPING command! For example one could use the 2-level trapezoid scheme with ImplCoeff=0.5 and UseBDF2=F as shown in the example above. The BDF2 scheme determines the coefficient for the implicit terms itself, but ImplCoeff is still used in the first time step and after AMR-s, when the code switches back to the two-level scheme for one time step.

#SEMICOEFF command

```
#SEMICOEFF
0.5                     SemiImplCoeff
```

The SemiImplCoeff is the coefficient in front of the implicit part of the semi-implicit terms. The value should be in the range 0.5 to 1. The 0.5 value will make the semi-implicit term 2nd order accurate in time, but currently the operator splitting still renders the full scheme first order in time only. Using 1.0 is the most robust option, as it makes the semi-implicit term to be evaluated fully implicitly, but it is first order accurate in time only. The default value is 1.

#IMPLSCHEME command

```
#IMPLSCHEME
1               nOrderImpl
Rusanov         TypeFluxImpl
```

This command defines the scheme used in the implicit part ('left hand side'). The default order is first order. The default scheme is the same as the scheme selected for the explicit part.

#IMPLCHECK command

```
#IMPLCHECK
0.3             RejectStepLevel
0.5             RejectStepFactor
0.6             ReduceStepLevel
0.95            ReduceStepFactor
0.8             IncreaseStepLevel
1.05            IncreaseStepFactor
```

The update checking of the implicit scheme can be tuned with this command. Update checking is done unless it is switched off (see UPDATECHECK command). After each (partially) implicit time step, the code computes pRhoRelMin, which is the minimum of the relative pressure and density drops over the whole computational domain. The algorithm is the following:

If pRhoRelMin is less than RejectStepLevel, the step is rejected, and the time step is reduced by RejectStepFactor; else if pRhoRelMin is less than ReduceStepLevel, the step is accepted, but the next time step is reduced by ReduceStepFactor; else if pRhoRelMin is greater than IncreaseStepFactor, the step is accepted and the next time step is increased by IncreaseStepFactor, but it is never increased above the value given in the FIXEDTIMESTEP command.

Assigning ReduceStepFactor=1.0 means that the time step is not reduced unless the step is rejected. Assigning IncreaseStepFactor=1.0 means that the time step is never increased, only reduced.

Default values are shown.

#NEWTON command

```
#NEWTON
T               UseNewton (rest of parameters read if true)
F               UseConservativeImplicit
10              MaxIterNewton
```

If UseNewton is true a full non-linear Newton iteration is performed. If UseConservativeImplicit is true, the Newton iteration is finished with a conservative fix (back substitution of the solution into the non-linear implicit equations). MaxIterNewton is the maximum number of Newton iterations before giving up.

Default is UseNewton=F, i.e. we do a single "Newton" iteration, which is the linearized implicit scheme. In most cases that is the best choice.

#JACOBIAN command

```
#JACOBIAN
T               DoPrecond
1E-12           JacobianEps
```

The Jacobian matrix is always calculated with a matrix free approach, however it can be preconditioned if DoPrecond is set to true. JacobianEps contains the machine round off error for numerical derivatives. The default value is 1.E-12 for 8 byte reals and 1.E-6 for 4 byte reals.

The default values are shown by the example.

#PRECONDITIONER command

```
#PRECONDITIONER
symmetric       TypePrecondSide (left, symmetric, right)
MBILU           TypePrecond (JACOBI, BLOCKJACOBI, GS, BILU, MBILU)
0.5             GustafssonPar (0 to 1, read for MBILU preconditioner only)
```

TypePrecondSide can be left, right or symmetric. There seems to be little difference between these in terms of performance. Right preconditioning does not affect the normalization of the residual. The JACOBI and BLOCKJACOBI preconditioners are implemented to always use left preconditioning.

The TypePrecond parameter can be set to JACOBI, GAUSS-SEIDEL, BLOCKJACOBI, BILU, MBILU.

The simplest Jacobi preconditioner is mainly useful for code development purposes. It uses the inverse of the diagonal elements of the approximate Jacobian matrix. The block-Jacobi preconditioner uses the invese of the diagonal blocks of the Jacobian matrix. It coincides with the Jacobi preconditioner for a scalar equation. The Gauss-Seidel (GS) preconditioner gives better performance than Jacobi, however, the BILU and MBILU preconditioners are usually more efficient. The Modified BILU (MBILU) preconditioner allows a Gustafsson modification relative to BILU. In some cases the modification improves the preconditioner, but sometimes it makes it worse.

The GustafssonPar parameter is only read for the MBILU preconditioner. If it is 0, the standard block (BILU) preconditioning is done. This seems to be optimal for diffusion+relaxation type problems. Setting a positive GustafssonPar up to 1 results in the modified (MBILU) preconditioner. The maximum 1 corresponds to the full Gustafsson modification. The default 0.5 value seems to be optimal for matrices resulting from hyperbolic (advection) type problems.

Default parameters are shown by the first example.

#SEMIPRECONDITIONER command

```
#SEMIPRECONDITIONER
T               DoPrecond (rest of parameters are read if true)
MBILU           TypePrecond (MBILU, BILU, DILU, GS, BLOCKJACOBI, JACOBI, HYPRE)
0.5             GustafssonPar (0 to 1, read for MBILU preconditioner only)

#SEMIPRECOND
T               DoPrecond
HYPRE           TypePrecond
```

Similar to the #PRECONDITIONER command but for the semi-implicit scheme.

If DoPrecond is false, no preconditioner is used. This will result in slower convergence. It is almost always preferable to use a preconditioner. The semi-implicit scheme always uses left side preconditioning.

The TypePrecond parameter can be set to the following values: JACOBI, BLOCKJACOBI, GS, DILU, BILU, MBILU or HYPRE. Most of these options are described in some detail for the #PRECONDITIONER command.

The Diagonal Incomplete Lower-Upper (DILU) preconditioner assumes that the off-diagonal blocks are diagonal matrices, and it gives the same result but faster performance than BILU in that case. This assumption holds if the derivative of a variable in the semi-implicit terms only affects the same variable (true for heat conduction, radiative diffusion, dissipative resistivity, but not for Hall resistivity).

The HYPRE preconditioner can only be used if the HYPRE library has been checked out into the util/ directory and Config.pl -hypre has been set. The HYPRE preconditioning only works if the semi-implicit scheme solves for 1 variable at a time (split semi-implicit scheme).

Default values are shown by the first example.

#KRYLOV command

```
#KRYLOV
GMRES           TypeKrylov  (GMRES, BICGSTAB, CG)
nul             TypeInitKrylov (nul, old, explicit, scaled)
0.001           ErrorMaxKrylov
100             MaxMatvecKrylov
```

Default values are shown.

The TypeKrylov parameter selects the iterative Krylov solver. The GMRES solver is the most robust and it converges the fastest among all Krylov solvers. It uses one matrix-vector product per iteration. On the other hand it needs to store one copy of the vector of the unknowns per iteration. GMRES also has to invert an NxN matrix in the N-th iteration. This means that GMRES is the optimal choice if the number of iterations is relatively small, typically less than 100. This is almost always true when the HYPRE preconditioner is used (see the #PRECONDITIONER command).

BICGSTAB is a robust Krylov scheme that only uses 4 copies of the unknown vector, and it uses two matrix-vector products per iteration. It usually requires somewhat more matrix-vector products than GMRES to achieve the same accuracy (defined by the tolerance ErrorMaxKrylov). On the other hand all iterations have the same computational cost.

The preconditioned Conjugate Gradient (CG) scheme only works for symmetric matrices. It only uses two copies of the unknown vector. For symmetric matrices it is more efficient than BiCGSTAB. In case many iterations are needed, it is more efficient than GMRES. The CG scheme currently does not work together with the HYPRE preconditioner.

Initial guess for the Krylov type iterative scheme can be 0 ('nul'), the previous solution ('old'), the explicit solution ('explicit'), or the scaled explicit solution ('scaled'). The iterative scheme stops if the required accuracy is achieved or the maximum number of matrix-vector multiplications is exceeded.

The ErrorMaxKrylov parameter defines the relative accuracy of the solution. The iteration stops when the residual (measured in the second norm) drops below the initial residual times ErrorMaxKrylov.

The MaxMatvecKrylov parameter limits the number of Krylov iterations. It also defines the maximum number of copies of the unknown vector for the GMRES solver, although this can be overwritten with the #KRYLOVSIZE command (see the description for more detail). If the Krylov solver does not succeed in achieving the desired accuracy within the maximum number of iterations, an error message is printed.

#SEMIKRYLOV command

```
#SEMIKRYLOV
GMRES           TypeKrylov  (GMRES, BICGSTAB, CG)
0.001           ErrorMaxKrylov
```

```
100             MaxMatvecKrylov
```

Same as the #KRYLOV command, but for the semi-implicit scheme. The initial guess is always zero, so there are only 3 parameters.

Default values are shown.

#KRYLOVSIZE command

```
#KRYLOVSIZE
10              nKrylovVector
```

The number of Krylov vectors only matters for GMRES (TypeKrylov='gmres'). If GMRES does not converge within nKrylovVector iterations, it needs a restart, which usually degrades its convergence rate and robustness. So nKrylovVector should exceed the number of iterations, but it should not exceed the maximum number of iterations MaxMatvecKrylov. On the other hand the dynamically allocated memory is also proportional to nKrylovVector. The default is nKrylovVector=MaxMatvecKrylov (in #KRYLOV) which can be overwritten by #KRYLOVSIZE after the #KRYLOV command (if any).

#SEMIKRYLOVSIZE command

```
#SEMIKRYLOVSIZE
10              nKrylovVector
```

Same as #KRYLOVSIZE but for the semi-implicit scheme. This command should be used after the #SEMIKRYLOV command (if present).

3.8.10 Stopping criteria

The commands in this group only work in stand alone mode.

#STOP command

```
#STOP
100                     MaxIteration
1 week                  tSimulationMax
```

This command is only used in stand alone mode.

The MaxIteration variable contains the maximum number of iterations *since the beginning of the current run* (in case of a restart, the time steps done before the restart do not count). If nIteration reaches this value the session is finished. The tSimulationMax variable contains the maximum simulation time relative to the initial time determined by the #STARTTIME command. If tSimulation reaches this value the session is finished.

Using a negative value for either variables means that the corresponding condition is not checked.

Either the #STOP or the #ENDTIME command must be used in every session.

#ENDTIME command

```
#ENDTIME
2000                    iYear
   3                    iMonth
  22                    iDay
  10                    iHour
  45                    iMinute
   0                    iSecond
```

This command can only be used in time accurate mode and in the final session.

The #ENDTIME command sets the date and time in Greenwich Mean Time (GMT) or Universal Time (UT) when the simulation should be ended. This is an alternative to the #STOP command in the final session. This time is stored in the final restart file as the start time for the restarted run, and the tSimulation parameter of the #TIMESIMULATION and the nStep parameter of the #NSTEP commands are set to zero. This avoids accumulation of tSimulation or nStep for continuously restarted runs.

There is no default value.

#CHECKSTOPFILE command

```
#CHECKSTOPFILE
T                       DoCheckStopFile
```

This command is only used in stand alone mode.

If DoCheckStopFile is true then the code checks if the BATSRUS.STOP file exists in the run directory. This file is deleted at the beginning of the run, so the user must explicitly create the file with e.g. the "touch BATSRUS.STOP" UNIX command. If the file is found in the run directory, the execution stops in a graceful manner. Restart files and plot files are saved as required by the appropriate parameters.

The default is DoCheckStopFile=.true.

#CPUTIMEMAX command

```
#CPUTIMEMAX
7.5 hours                   CpuTimeMax [sec]
```

This command is only used in stand alone mode.

The CpuTimeMax variable contains the maximum allowed CPU time (wall clock time) for the execution of the current run. If the CPU time reaches this time, the execution stops in a graceful manner. Restart files and plot files are saved as required by the appropriate parameters. This command is very useful when the code is submitted to a batch queue with a limited wall clock time.

The default value is -1.0, which means that the CPU time is not checked. To do the check the CpuTimeMax variable has to be set to a positive value.

3.8.11 Output parameters

#RESTARTOUTDIR command

```
#RESTARTOUTDIR
GM/restart_n5000        NameRestartOutDir
```

The NameRestartOutDir variable contains the name of the directory where restart files are saved relative to the run directory. The directory should be inside the subdirectory with the name of the component.

Default value is "GM/restartOUT".

#RESTARTOUTFILE command

```
#RESTARTOUTFILE
one series                  TypeRestartOutFile
```

This command determines if the restart information is saved as an individual file for each block (block), as direct access files for each processor (proc) or into a single direct access file containing all blocks (one).

Normally saving restart files overwrites the previous files. Adding 'series' after the type results in a series of restart files with names starting as nITERATION_. This will be used by the adjoint method.

The most reliable format is 'proc'. If there is any issue with restarting with the 'one' format (some machines write empty records into the file), the 'proc' should be used. The 'block' format can fail due to too many files.

The default value is 'proc'.

#SAVERESTART command

```
#SAVERESTART
T                       DoSaveRestart Rest of parameters read if true
100                     DnSaveRestart
-1.                     DtSaveRestart [seconds]
```

Default is DoSaveRestart=.true. with DnSaveRestart=-1 and DtSaveRestart=-1. This results in the restart file being saved only at the end. A binary restart file is produced for every block and named as RESTARTOUTDIR/blkGLOBALBLKNUMBER.rst. In addition the grid is described by RESTARTOUTDIR/octree.rst and an ASCII header file is produced with timestep and time info: RESTARTOUTDIR/restart.H

The restart files are overwritten every time a new restart is done, but one can change the name of the RESTARTOUTDIR with the #RESTARTOUTDIR command from session to session. The default directory name is 'restartOUT'.

#PLOTDIR command

The NamePlotDir variable contains the name of the directory where plot files and logfiles are saved relative to the run directory. The directory should be inside the subdirectory with the name of the component.

Default value is "GM/IO2".

#SAVELOGFILE command

```
#SAVELOGFILE
T                       DoSaveLogfile, rest of parameters read if true
VAR step date GSE       StringLog
100                     DnSaveLogfile
-1.                     DtSaveLogfile [sec]
rho p ux uy uz rhoflx   NameLogVars (read if StrigLog is 'var' or 'VAR')
4.0  10.0               rLog  (radii for the flux. Read if vars include 'flx')
```

If DoSaveLogFile is set to true then an ASCII logfile containing averages, point values, fluxes and other scalar quantities is written at the frequency determined by DnSaveLogfile and DtSaveLogfile. The logfile is written into IO2/log_TIMESTAMP.log. The TIMESTAMP contains the time step or the date-time string (see the #SAVELOGNAME command). The logfile has a very simple format:

```
arbitrary header line that can define for example the units
name1 name2 name3 ... nameN
value1 value2 value3 ... valueN
value1 value2 value3 ... valueN
value1 value2 value3 ... valueN
...
```

The number variables as well as the number of data lines are arbitrary. The IDL macros getlog and plotlog can be used for visualization of one or more logfiles.

The StringLog parameter can contain the following parts in arbitrary order:

```
StringLogVar  = 'mhd', 'raw', 'flx' or 'var' - normalized units
StringLogVar  = 'MHD', 'RAW', 'FLX' or 'VAR' - I/O units
StringLogTime = 'none', 'step', 'time', and/or 'date'
NameCoord     = 'GEO', 'GSE', 'GSM', 'MAG', 'SMG', 'HGR', 'HGI' or 'HGC'
```

The StringLogVar part is required and it determines the list of variables to be saved into the logile. The capitalization of StringLogVar controls whether the variables are written in normalized units (lower case) or I/O units (UPPER CASE). (see the #IOUNITS command). The StringLogTime part is optional.

The possible values for StringLogVar and the corresponding variables together with the default values for StringLogTime are the following:

```
raw or RAW   - step time Dt AverageConsVars Pmin Pmax
mhd or MHD   - step date time AverageConsVars Pmin Pmax
flx or FLX   - step date time Rho Pmin Pmax RhoFlx Pvecflx e2dflx
var or VAR   - step time NameLogVars
```

The right side shows what will be saved into the logfile. The 'step', 'time' and 'date' columns correspond to the default value of StringLogTime that is discussed below. About the other variables: Dt is the length of the time step, the AverageConsVars contain a list of averages of the conservative variables (defined in ModEquation) over the whole domain, and Pmin and Pmax are the minimum and maximum pressures over the whole domain. The flux variables are integrals of fluxes through spherical surfaces at the radial distances defined by the rLog parameter that is read if any of the variables contain 'flx' (see below).

If StringLogVar is 'var' or 'VAR', then the NameLogVars parameter is read and it should contain a space separated list of any of the following log variable names:

```
Dt                  - time step
Cfl                 - CFL number (may vary due to #TIMESTEPCONTROL)

Pmin Pmax           - minimum and maximum pressure over the grid

VAR                 - average of variable VAR (listed in NameVar_V of ModEquation.f90)
Ux Uy Uz            - average velocity on the grid
Ekinx Ekiny Ekinz   - average kinetic energy in X, Y and Z directions
Ekin                - average kinetic energy
Erad Ew             - average radiation/wave energy (summed for all groups/waves)
Lat1 Lat2           - average of latitude of fieldline tracing (for testing)
Lon1 Lon2 Status    - average of longitude and status of fieldline (for testing)

VARpnt              - point value of VAR (listed in NameVar_V) at the test cell
Uxpnt Uypnt Uzpnt   - point value of velocity at test cell
B1xpnt B1ypnt B1zpnt- point value of B1 field at test cell
Jxpnt Jypnt Jzpnt   - point value of current density
Lat1pnt Lat2pnt     - point value of latitude of fieldline mapping
Lon1pnt Lon2pnt     - point value of longitude of fieldline mapping
Statuspnt           - point value of status of fieldline tracing from test cell:
  (0: open, 1: connected along B, 2: connected along -B, 3: closed)

Jinmax Joutmax      - maximum of inward and outward currents at rCurrents of #BODY
Jin Jout            - surface integral of inward and outward currents at rCurrents

Aflx                - surface integral of 1      at radii defined in rLog (area)
Rhoflx              - surface integral of rho*Ur at radii defined in rLog
Bflx                - surface integral of Br     at radii defined in rLog
B2flx               - surface integral of B.B Ur at radii defined in rLog
Pvecflx             - surface integral of ExB    at radii defined in rLog (Poynting flux)
Dstflx              - surface integral of B1z    at radii defined in rLog (Dst index)
```

```
DstDivb             - error of the dstflx integral due to finite value of div B
Dst                 - Biot-Savart integral for Dst index

E2dflx              - circular integral of Ex*y-Ey*x

ANYTHINGELSE        - quantity defined in user_get_log_var
```

The possible (0 or more) values for StringLogTime are the following:

```
 none  - there will be no indication of time in the logfile (not even the number of steps)
 step  - number of time steps
 date  - date-time as 7 integers:  year month day hour minute second millisecond,
 time  - simulation time
```

The rLog parameter contains a list of radius values (in normalized units) where the *flx variables are calculated. The rLog parameter is only read when there is at least one 'flx' variable. The logfile will contain the name of the flx variable combined with the radial value, for example 'dstflx_R=3.0 dstflx_R=3.5 dstflx_R=5.0'.

If the optional NameCoord part is set, the output position, velocity, magnetic field or current density vector variables will be written out in the NameCoord coordinate system instead of the coordinate system of the component. In the list of log variables the X, Y and Z components of a given vector have to be all present and following each other in this order.

The default is DoSaveLogFile false.

#SATELLITE command

```
#SATELLITE
4                       nSatellite
MHD ray                 StringSatellite
100                     DnOutput
-1.                     DtOutput [sec]
satellite1.dat          NameTrajectoryFile
MHD trajrange           StringSatellite
100                     DnOutput
-1.                     DtOutput [sec]
satellite1.dat          NameTrajectoryFile
2011-02-01T00:00:00 UT  StringStartTimeTraj ! Read if StringSatellite contains 'trajrange'
2011-02-10T00:00:00 UT  StringEndTimeTraj   ! Read if StringSatellite contains 'trajrange'
1 h                     StringDtTraj        ! Read if StringSatellite contains 'trajrange'
VAR traj                StringSatellite
100                     DnOutput
-1.                     DtOutput [sec]
satellite1.dat          NameTrajectoryFile
rho p ux uy uz
VAR step date SMG       StringSatellite
100                     DnOutput
-1.                     DtOutput [sec]
satellite2.dat          NameTrajectoryFile
rho p ux uy uz          NameSatelliteVars ! Read if StringSatellite
                                          ! contains 'var' or 'VAR'
```

The numerical solution can be extracted along one or more satellite trajectories. The number of satellites is defined by the nSatellite parameter (default is 0).

For each satellite the StringSatellite parameter determines what is saved into the satellite file(s). The StringSatellite can contain the following parts in arbitrary order

```
SatelliteVar   = mhd, ful or var (normalized units)
                 MHD, FUL or VAR (I/O units)
NameCoord      = GEO, GSE, GSM, MAG, SMG, HGR, HGI, HGC, hgr, hgi, hgc
OptionalVar    = ray, none, step, time, traj, trajrange, date
```

The SatelliteVar part is required and determines the list of variables to be saved along the satellite trajectory. The capitalization of SatelliteVar controls whether the variables are written in normalized units (lower case) or I/O units (UPPER CASE). See the #IOUNITS command.

If SatelliteVar' is set to 'mhd' or 'MHD', the primitive variables (rho, u, B, p, pPar) and the current density (Jx, Jy, Jz) will be saved, while the 'ful' or 'FUL' value also saves the B1 field values. If SatelliteVar is set to 'var' or 'VAR' then the list of variables is read from the NameSatelliteVar parameter as a space separated list. The choices for saved variables are any of the variable names listed in the NameVar_V variable in ModEquation.f90, and the following case insensitive variable names (after the name of the fluid is removed, e.g. OpPperp is Pperp for fluid Op):

```
Mx, My, Mz, Ux, Uy, Uz       - momentum and velocity components
B1x, B1y, B1z, B0x, B0y, B0z - magnetic field perturbation and background
Jx, Jy, Jz                   - current density
n                            - number density
T, Temp                      - temperature
Pperp                        - perpendicular pressure
Lon1, Lon2                   - longitude of mapped field line along B and -B
Lat1, Lat2                   - latitude of mapped field line along B and -B
Status                       - field line topology
        (0: open, 1: closed along B, 2: closed along -B 3: fully closed)
        (-1: cells inside body, -2: loop ray within block, -3: strange)
```

If the optional NameCoord part is set, the output position, velocity, magnetic field or current density vector variables will be written out in the NameCoord coordinate system instead of the coordinate system of the component. In the list of log variables the X, Y and Z components of a given vector have to be all present and following each other in this order.

If the optional OptionalVar part contains 'ray' then the ray (fieldline) tracing variables 'Lon1 Lat1 Lon2 Lat2 Status' are saved as well. The strings 'step', 'time' and 'date' define the corresponding time information. The value 'none' means that no time information is saved.

```
 none  - there will be no indication of time or steps in the logfile
 step  - number of time steps
 date  - date-time as 7 integers:  year month day hour minute second millisecond
 time  - simulation time
 ray   - fieldline tracing variables: lon1 lat1 lon2 lat2 status
 traj  - extract information along the full trajectory file
 trajrange  - extract information along a part of the trajectory file
```

More than one OptionalVar strings can be listed. They can be put together in any combination. The default value for OptionalVar is 'step date'.

The DnOutput and DtOutput parameters determine the frequency of extracting values along the satellite trajectories.

The extracted satellite information is saved into the files named

```
PLOTDIR/sat_TRAJECTORYNAME_TIMESTAMP.sat
```

where TIMESTAMP contains the time step or the date-and-time information (see #SAVELOGNAME command) and TRAJECTORYNAME is the name of the trajectory file. The satellite files have a very simple format:

```
arbitrary header line that can define for example the units
name1 name2 name3 ... nameN
value1 value2 value3 ... valueN
value1 value2 value3 ... valueN
value1 value2 value3 ... valueN
...
```

The number variables as well as the number of data lines are arbitrary. The IDL macros getlog and plotlog can be used for visualization of one or more logfiles.

Satellite input files contain the trajectory of the satellite. They should have the following format:

```
#COOR
GSM

#START
2004 6 24 0 0 58 0 2.9 -3.1 -3.7
2004 6 24 0 1 58 0 2.8 -3.2 -3.6
...
```

The #COOR command is optional. It indicates which coordinate system is used for the trajectory coordinates. Possible values (GSM, GEO, GSE, SMG, HGI, HGR...) and their meaning is described in share/Library/src/CON_axes.f90. The default coordinate system is GSM. After the #START line, the data lines contain the date-time information (year, month, day, hour, minute, second, millisecond) and the x, y and z coordinates in normalized units (typically planetary or solar radius, see the #NORMALIZATION command).

If the StringSatellite contains 'traj', then the code simply extract the information at ALL satellite locations from the satellite file.

If the StringSatellite contains 'trajrange', then StringStartTimeTraj, StringEndTimeTraj and StringDtTraj need to be provided followed by the NameTrajectoryFile. This alows writing the information along the trajectory file for a given time range, both in time accurate and steady state. StringStartTimeTraj and StringEndTimeTraj determine the time range of the output satellite file and accept the following forms of string:

```
 A time string ending with ' UT', such as 'YYYY-MM-DDTHH:MM:SS:MSC UT' or
            'YYYY MM DD HH MM SS MSC UT' (single digit need to fill 0 ahead and the
            sperator ('-', ':', 'T', ' ') between numbers can be whatever characters)
 Any number followed by ' w'/' d'/' h'/' m'/' s', in which case the time is with respect to
            #STARTTIME. For example, 1 h here means StartTime + 1 hour. THERE IS A SPACE
            BEFORE THE CHARACTER W/D/H/M/S.
 Any number (in which case it is assumed to be in seconds), in which case the time
            is with respect to #STARTTIME.
```

StringDtTraj can accept the following forms:

```
 Any number followed by ' w'/' d'/' h'/' m'/' s'
 Any number (in which case it is assumed to be in seconds)
```

The default is nSatellite=0, i.e. no satellite data is saved.

#STEADYSTATESATELLITE command

```
#STEADYSTATESATELLITE
-1 day                      SatelliteTimeStart [sec]
+1 day                      SatelliteTimeEnd   [sec]
-1 hour                     SatelliteTimeStart [sec]
+1 hour                     SatelliteTimeEnd   [sec]
```

In the non-time-accurate mode the numerical simulation result converges to a steady-state solution. In the course of this simulation mode, the progress in the iteration number is not associated with an increase in the physical time, and the ultimate solution is a ”snapshot” of the parameter distribution at the time instant set by the #STARTTIME command. Since time does not run, a satellite position cannot be determined in terms of the simulation time. Instead, the parameters along a cut of the satellite trajectory can be saved on file for a given iteration number. The trajectory points can be naturally parameterized by time, so that the cut can be specified with the choice of the start time, end time, and time interval.

The command #STEADYSTATESATELLITE is required for a steady-state simulation. For each of the satellites, the SatelliteTimeStart is a real value that sets the start of trajectory cut, while SatelliteTimeEnd sets the end of the trajectory cut. Both are in seconds with respect to the time given in #STARTTIME. A negative value means the is time prior to the #STARTTIME.

The DtOutput from the #SATELLITE command specifies the frequency of the points along the satellite trajectory for the non-time-accurate mode, while DnOutput keeps to control the iteration number at which the data at the trajectory cut are written to the satellite output file.

For more than one satellite (two satellites in the above given example), the start and end times should be set for all of them.

#PARCEL command

```
#PARCEL
T                       UseParcel ! Rest is read if true
F                       UseParcelTable
2                       nParcel ! if UseParcelTable=F
0.0508514               xParcel 1
-1.01483                yParcel 1
-0.0888                 zParcel 1
-0.178076               xParcel 2
0.442199                yParcel 2
-0.901742               zParcel 2
VAR                     StringParcelVar
1                       DnOutput
-10                     DtOutput
rho ux pe o(9)          NameParcelVars ! if StringParcelVar = var/VAR
```

The numerical solution can be extracted along one or more plasma parcel trajectories. The number of trajectories is defined by the nParcel parameter (default is 0).

For each parcel the StringParcel parameter determines what is saved into the file(s). Possible values are

```
StringParcel   = mhd, ful or var (normalized units)
                 MHD, FUL or VAR (I/O units)
```

The capitalization of StringParcel controls whether the variables are written in normalized units (lower case) or I/O units (UPPER CASE). If StringParcel is set to var or VAR, then the list of plot variables are read as the last parameter.

The DnOutput and DtOutput parameters determine the frequency of extracting values along the parcels’ trajectories.

The extracted information is saved into the files named

```
PLOTDIR/pcl_ID_TIMESTAMP.pcl
```

where TIMESTAMP contains the time step or the iteration number information and ID is the ID between 1 and nParcel of the file. The pcl files have a very simple format:

```
arbitrary header line that can define for example the units
name1 name2 name3 ... nameN
value1 value2 value3 ... valueN
value1 value2 value3 ... valueN
value1 value2 value3 ... valueN
...
```

The number variables as well as the number of data lines are arbitrary. The IDL macros getlog and plotlog can be used for visualization of one or more logfiles.

The default is UseParcel=F, i.e. no Lagrangian data is saved.

#MAGPERTURBINTEGRAL command

```
#MAGPERTURBINTEGRAL
T                       UseSurfaceIntegral
F                       UseFastFacIntegral
MAG                     TypeCoordIndex
MAG                     TypeCoordFacGrid (read if UseFastFacIntegral=F)
```

Control what method is used to do the Biot-Savart integrals to calculate the magnetic perturbations.

If UseSurfaceIntegral is true, the volume integral over the MHD grid is replaced with a surface integral using Igor Sokolov's formulas. This surface integral is analytically identical with the 3D volume integral outside the sphere plus the contributions from the external field outside the MHD grid, but computationally much less expensive. See Appendix C.5.1 in Gombosi et al 2021 JSWSC, doi:10.1051/swsc/2021020.

If UseFastFacIntegral is true, the integrals across the gap region are precalculated for each magnetometer and each line starting from the lat-lon grid and stored into an array LineContrib_DII. At any given time this array can be multiplied with the FAC values calculated at rCurrents to obtain the perturbation at a given magnetometer. The storage as well as the calculation is parallel. See Appendix C.5.2 in Gombosi et al 2021 JSWSC, doi:10.1051/swsc/2021020.

TypeCoordIndex defines whether the 48 virtual magnetometers used for Kp, Ae and other indexes are in the SMG system or the co-rotating MAG system. The MAG system allows the use of the fast FAC integration for these stations.

TypeCoordFacGrid defines the coordinate system for the spherical grid used to integrate the contributions from the FAC in the gap region. This has to be in the corotating MAG system if UseFastFacIntegral is true. For the slow FAC integral method, the SMG system is allowed too.

Default values are UseSurfaceIntegral=T, UseFastFacIntegral=T, TypeCoordIndex='MAG' and TypeCoordFacGrid='MAG', which are the optimal settings for best computational speed.

#GEOMAGINDICES command

```
#GEOMAGINDICES
180                     nSizeKpWindow [min]
60.0                    DtOutput      [sec]
```

BATS-R-US can create synthetic geomagnetic indices by first simulating ground based measurements then processing these data into indices. This allows for an apples-to-apples comparison of indices created by the simulation against indices created from observations. It is also useful in an operational setting, where quick-look activity indices are paramount. #GEOMAGINDICES activates the calculation of such indices. Results are written at a time cadence of DtOutput to the file geoindex_TIMESTAMP.log

At present, only a synthetic version of Kp is available. nSizeKpWindow, set in minutes and defaulting to 180 (3 hours), sets the size of the time-history window used in the calculation of Kp. Standard Kp uses a 3-hour window; versions of Kp used as operational products use a window as short as 15-minutes. Note that altering this window requires a re-scaling of the K-index conversion tables inside of the code. As Kp is written to file, so are the individual K-indices used in the calculation. Offical Kp averages 13 K values from 13 mid-latitude magnetometer stations around

the globe. Synthetic Kp from BATS-R-US uses 24 stations at fixed local time positions and 50 degrees magnetic latitude.

Because Kp requires a time history of geomagnetic activity, special restart files are saved when #GEOMAGINDICES is used. If nSizeKpWindow changes between restarts, however, the files will be rendered unusable because the time history will no longer be valid for the calculation.

By default no indices are calculated.

#MAGNETOMETER command

```
#MAGNETOMETER
magin.dat                NameMagInputFile
single                   TypeFileOut
-1                       DnOutput
60.0                     DtOutput
```

The #MAGNETOMETER command is used for the calculation of the ground perturbations caused by the field aligned currents in the 'gap' region and the magnetopsheric currents in the GM domain.

The NameMagInputFile parameter gives the file name that contains the locations on the Earth where the user is interested in calculating the ground magnetic perturbations. The file has the following format:

```
#COORD
MAG                      The coordinate system for the latitude/longitude below

#START
DST  360.00   360.00     Virtual DST station at the center of the Earth
YKC   68.93   299.36     The name of the station, latitude, longitude
MEA   61.57   306.20
NEW   54.85   304.68
...
```

The coordinate system can be set to GEO (geographic), MAG (geomagnetic) or SMG (solar magnetic) coordinates. The station names can have maximum 3 characters. The name, latitude, and longitude columns should be separated with spaces. If the latitude and longitude are both set to 360.0, the station is placed to the center of the Earth, and the perturbation for this "DST" station will be given in SMG coordinates.

The TypeFileOut parameter specifies the format of the output file. The value 'single' creates a single output file for all magnetometers and all output times, while 'step' creates a new file for all magnetometers for each output time. The naming convention for the files is controlled by the #SAVELOGNAME command.

The DnOutput and DtOutput parameters determine the frequency of writing out the calculated perturbations in number of time steps and time interval, respectively.

The ground-based magnetic perturbations are written into the output file

```
GM/IO2/magnetometers_*.dat,
```

in which the number of time steps, the date and time, the station index, the x, y and z coordinates of the station in SM coordinates, the 3 components (magnetic northward, eastward, and downward) of the total magnetic perturbations, as well as the contributions due to magnetospheric currents, field-aligned currents in the gap region, and Hall and Pedersen currents in the ionosphere are saved. For the "DST" station at the center of the Earth the magnetic perturbations are given in the SMG coordinates: north=x, east=y, down=z. The units of coordinates is meters, while the magnetic perturbations are given in nT.

Default is no magnetic perturbation calculation.

#MAGNETOMETERGRID command

```
#MAGNETOMETERGRID
2                       nMagGridFile
global ascii            StrGridFileOut (ascii, tec, real4, real8)
GEO                     TypeCoordGrid (GEO, MAG, SMG)
360                     nGridLon
171                     nGridLat
0.                      GridLonMin
360.                    GridLonMax
-85.                    GridLatMin
85.                     GridLatMax
-1                      DnSaveMagGrid
60.0                    DtSaveMagGrid
us real4                StrGridFileOut (ascii, tec, real4, real8)
GEO                     TypeCoordGrid (GEO, MAG, SMG)
320                     nGridLon
171                     nGridLat
200.                    GridLonMin
360.                    GridLonMax
0.                      GridLatMin
85.                     GridLatMax
-1                      DnSaveMagGrid
30.0                    DtSaveMagGrid
```

This command allows calculating and saving magnetic perturbations on multiple longitude-latitude grids. nMagGridFile specifies the number of magnetometer grid outputs.

StrGridFileOut specifies the region name (part of the file name) and format. The region name can be any arbitrary string that the user can name for the region, and the format can be 'ascii' (text file), 'tec' (Tecplot text file), 'real4' (single precision binary) or 'real8' (double precision binary). The coordinate system of the grid (not the output) can be set to GEO (geographic), MAG (geomagnetic) or SMG (solar magnetic) coordinates. The region name can be any user specified string, e.g., usa, europe, global, etc.

The number of grid points in the longitude and latitude directions is given by nGridLon and nGridLat, respectively. The longitudes span from from GridLonMin to GridLonMax, while the latitudes span from GridLatMin to GridLatMax. If the longitude spans 360 degrees, the stations will be arranged so that the equivalent longitudes of 0 and 360 are not repeated. However, if -90 or +90 degrees is used for the maximum/minimum latitude, the polar stations will be repeated nLonMagGrid times, so choose limits wisely. The 2D output files are saved every DnSaveMagGrid steps or DtSaveMagGrid seconds.

No magnetometer grid file is saved by default.

#SUPERMAGINDICES command

```
#SUPERMAGINDICES
T                       DoWriteSupermagIndices
```

This command calculates and saves synthetic SuperMAG geomagnetic indices from the magnetometer grid.

The indices SML (AL), SMU (AU), SME (AE), and SMO (AO) are computed using every grid point defined in the #MAGNETOMETERGRID command, within the magnetic latitude range +40 to +80. Output is written at the same cadence as the DnSaveMagGrid or DtSaveMagGrid and saved in the superindex*.log file.

If the #SUPERMAGINDICES command is used without the #MAGNETOMETERGRID command, or if the magnetometer grid does not cover the full latitude range from +40 to +80, then a warning message will be generated and indices are not calculated.

#SAVEPLOT command

```
#SAVEPLOT
21                      nPlotFile
cut MHD tcp             StringPlot ! 3d cell centered Tecplot with MHD data
100                     DnSavePlot
-1.                     DtSavePlot
-10.                    Coord1MinCut
10.                     Coord1MaxCut
-10.                    Coord2MinCut
10.                     Coord2MaxCut
-10.                    Coord3MinCut
10.                     Coord3MaxCut
2d  FUL hdf             StringPlot ! 2d HDF plot with a lot of data
100                     DnSavePlot
-1.                     DtSavePlot
1d  HD idl_tec          StringPlot ! 1d plot (with Tecplot header) along X axis
100                     DnSavePlot
-1.                     DtSavePlot
0.                      DxSavePlot ! with smallest grid resolution
y=0 VAR idl             StringPlot ! y=0 plane plot with listed variables
-1                      DnSavePlot
100.                    DtSavePlot
0.25                    DxSavePlot ! resolution (for IDL plots)
{MHD} impl dx           NameVars
{default} c             NamePars
cut ray idl_real8       StringPlot ! 3D cut plot with raytrace info
1                       DnSavePlot
-1.                     DtSavePlot
-10.                    Coord1MinCut
10.                     Coord1MaxCut
-10.                    Coord2MinCut
10.                     Coord2MaxCut
-10.                    Coord3MinCut
10.                     Coord3MaxCut
-1.                     DxSavePlot ! unstructured grid (for IDL plots)
los tbl idl_real4       StringPlot ! line of sight plot using table
-1                      DnSavePlot
100.                    DtSavePlot
-215.                   ObsPosX
0.                      ObsPosY
0.                      ObsPosZ
5.0                     OffsetAngle ! rotate around with 5 degree resolution
32.                     rSizeImage
0.                      xOffset
0.                      yOffset
3.                      rOccult
0.5                     MuLimbDarkening
300                     nPix
AiaXrt                  NameLosTable
lin mhd idl             StringPlot  ! field line plot
```

```
-1                      DnSavePlot
10.                     DtSavePlot
B                       NameLine ! B - magnetic field line, U - stream line
F                       IsSingleLine
2                       nLine
-2.0                    xStartLine
0.0                     yStartLine
3.5                     zStartLine
F                       IsParallel
-1.0                    xStartLine
1.0                     yStartLine
-3.5                    zStartLine
T                       IsParallel
eqr eqr idl             StringPlot ! Equatorial (magnetic) field line tracing info
1000                    DnSavePlot
-1.                     DtSavePlot
20                      nRadius    ! Starting points on the SM equatorial plane
25                      nLon
3.0                     RadiusMin
10.0                    RadiusMax
eqb eqb tec             StringPlot ! Minimum B surface plot
1000                    DnSavePlot
-1.                     DtSavePlot
20                      nRadius    ! Starting points on the SM equatorial plane
25                      nLon
3.0                     RadiusMin
10.0                    RadiusMax
60.0                    LongitudeMin
300.0                   LongitudeMax
dpl MHD tec             StringPlot  ! dipole slice Tecplot (ONLY!) plot
-1                      DnSavePlot
10.                     DtSavePlot
-10.                    xMinCut
 10.                    xMaxCut
-10.                    yMinCut
 10.                    yMaxCut
-10.                    zMinCut
 10.                    zMaxCut
slc MHD tec             StringPlot  ! general slice Tecplot (ONLY!) plot
-1                      DnSavePlot
10.                     DtSavePlot
-10.                    xMinCut
 10.                    xMaxCut
-10.                    yMinCut
 10.                    yMaxCut
-10.                    zMinCut
 10.                    zMaxCut
 0.                     xPoint
 0.                     yPoint
 0.                     zPoint
  0.                    xNormal
```

```
  0.                    yNormal
  1.                    zNormal
blk MHD tec             StringPlot  ! general block Tecplot (ONLY!) plot
-1                      DnSavePlot
10.                     DtSavePlot
 5.                     xPoint
 1.                     yPoint
 1.                     zPoint
ieb nul tec             StringPlot  !IE grid field line plots Tecplot (ONLY!)
1000                    DnSavePlot
-1.                     DtSavePlot
lcb int tec             StringPlot  !last closed field line plots with integrals
1000                    DnSavePlot  !Tecplot (ONLY!)
-1.                     DtSavePlot
6.                      Radius
36                      nLon
shl MHD idl             StringPlot
10                      DnSavePlot
-1.                     DtSavePlot
GEO                     TypeCoordPlot
5.6                     rMin
7.6                     rMax
0.5                     dRad    ! only read if rMin /= rMax
0.                      LonMin
360.                    LonMax
10.                     dLon    ! only read if LonMin /= LonMax
-90.                    LatMin
90.                     LatMax
10.                     dLat    ! only read if LatMin /= LatMax
shk HD idl              StringPlot
10                      DnSavePlot
-1.                     DtSavePlot
-10.0                   DivuDxMin [km/s]
1.2                     rMin
25.0                    rMax
0.                      LonMin
360.                    LonMax
1.                      dLon
-90.                    LatMin
90.                     LatMax
1.                      dLat
box MHD idl             StringPlot
1                       DnSavePlot
-10.0                   DtSavePlot
HGR                     TypeCoordPlot
0.0                     x0
0.0                     y0
1.5                     z0
2.0                     xSize
.01                     dX       ! only read if xSize /= 0
0.2                     ySize
```

```
0.01                    dY       ! only read if ySize /= 0
3.0                     zSize
0.01                    dZ       ! only read if zSize /= 0
0.0                     xAngle [deg]
90.0                    yAngle [deg]
0.0                     zAngle [deg]
los dem idl_ascii       StringPlot
4                       DnSavePlot
-1.                     DtSavePlot
HGR                     TypeCoord
216                     ObsPosX
0                       ObsPosY
0                       ObsPosZ
0                       x0
0                       y0
2                       xLen
0.1                     dX
2                       yLen
0.1                     dY
0.0                     TempMin
5                       LogTeMinDEM
8                       LogTeMaxDEM
0.1                     DLogTeDEM
los fux idl_ascii       StringPlot
4                       DnSavePlot
-1.                     DtSavePlot
HGR                     TypeCoord
216                     ObsPosX
0                       ObsPosY
0                       ObsPosZ
0                       x0
0                       y0
2                       xLen
0.1                     dX
2                       yLen
0.1                     dY
0.0                     TempMin
SPECTRUM_chianti_tbl.dat    NameSpmTable
F                       UseUnobserved
400                     LambdaMin
410                     LambdaMax
0.1                     DLambda
T                       UseAlfven
T                       UseDoppler
0.0                     DLambdaIns
F                       UseIonFrac
F                       UseIonTemp
los nbi idl_ascii       StringPlot
4                       DnSavePlot
-1.                     DtSavePlot
HGR                     TypeCoord
```

```
216                     ObsPosX
0                       ObsPosY
0                       ObsPosZ
0                       x0
0                       y0
2                       xLen
0.1                     dX
2                       yLen
0.1                     dY
0.0                     TempMin
SPECTRUM_chianti_tbl.dat    NameSpmTable
F                       UseIonFrac
eit195response.out      NameResponse
los phx idl_ascii               StringPlot
4                       DnSavePlot
-1.                     DtSavePlot
HGR                     TypeCoord
216                     ObsPosX
0                       ObsPosY
0                       ObsPosZ
1                       x0
1                       y0
2                       xLen
0.2                     dX
2                       yLen
0.2                     dY
0                       TempMin
PHOTOEXC_6376.2900.dat          NamePhxTable
6375                    LambdaMin
6378                    LambdaMax
0.1                     DLambda
T                       UseAlfven
T                       UseDoppler
0.02                    DLambdaInstrument
F                       UseIonFrac
F                       UseIonTemp
rfr tec rwi             StringPlot
10                      DnSavePlot
-1.0                    DtSavePlot
 -67.92                 ObsPosX
 200.40                 ObsPosY
 -26.91                 ObsPosZ
1.5 GHz 500 MHz 100 MHz    StringRadioFrequency
4.0                     xSizeImage
4.0                     ySizeImage
100                     nPixX
100                     nPixY
los ins idl_real4       StringPlot ! line of sight plot using table
-1                      DnSavePlot
100.                    DtSavePlot
soho:c3 sta:euvi stb:cor2       StringsInstrument
```

```
buf MHD idl             StringPlot
-1                      DnSavePlot
1 hour                  DtSavePlot
bx0 MHD idl_ascii       StringPlot ! bx=0(on z) isosurface plot with MHD data
100                     DnSavePlot
-1.                     DtSavePlot
-10.                    xMinCut
10.                     xMaxCut
-10.                    yMinCut
10.                     yMaxCut
-10.                    zMinCut
10.                     zMaxCut
-1                      DxSavePlot
```

The #SAVEPLOT command determines the number and type of plot files saved from BATS-R-US.

The nPlotFile parameter sets the number of plot files to be saved. For each plot file, the StringPlot parameters defines the format and the content as detailed below. The PlotString is always followed by the plotting frequencies DnSavePlot and DtSavePlot that determine the frequency of saves in temrs of time steps and simulation time, respectively. The rest of the parameters read for a given plot file depends on StringPlot.

StringPlot must contain the following 3 parts in the following order

```
PlotForm PlotArea PlotVar
```

Each of these parts can have different values. Most (but not all) combinations are valid. The PlotForm can have one of the following values:

```
tec             - Node based Tecplot format
tcp             - Cell centered Tecplot format
hdf             - HDF5 format (for VisIt)
idl/idl_real4   - Single precision binary "IDL" format
idl_real8       - Double precision binary "IDL" format
idl_ascii       - ASCII "IDL" format
idl_tec         - ASCII format with Tecplot header
```

The node based Tecplot format (for most plot areas) interpolates data to the grid cell corners (nodes). The cell centered Tecplot, HDF and IDL formats save the cell center values. The HDF output works only if the HDF library is installed, the appropriate parallel HDF module is loaded and BATSRUS/SWMF is configured with the -hdf flag. The ”IDL” format can be read with the IDL visualization macros (read_data and animate_data) in BATSRUS/Idl or with the SpacePy python package. The ASCII ”IDL” format can be easily read with any other plotting software. The ”IDL” file format is described at the beginning of the share/Library/src/ModPlotFile.f90.

The PlotArea string defines a 1, 2, or 3D volume for plotting:

```
1d    - 1D cut along the X axis (saves tree file)
2d    - 2D cut (like Z=0)       (saves tree file)
3d    - full 3D volume          (saves tree file)
x=0   - full x=0 plane: average for symmetry plane
y=0   - full y=0 plane: average for symmetry plane
z=0   - full z=0 plane: average for symmetry plane
cut   - 3D, 2D or 1D cut along (curvilinear) coordinates (IDL and TCP)
        or a 2D rectangular cut (node based Tecplot)
bx0   - bx=0 (along z direction) isosurface plot
dpl   - cut at dipole 'equator', uses PLOTRANGE to clip plot
slc   - 2D slice defined with a point and normal, uses PLOTRANGE to clip plot
```

```
shl   - spherical shell in given coordinate system (1, 2 or 3D)
sln   - spherical shell with ln(r) radial coordinate
slg   - spherical shell with log10(r) radial coordinate
shk   - shock surface extracted on a lon-lat grid. Limited in radial distance.
box   - cartesian box in given coordinate system (1, 2, or 3D)
los   - line of sight integrated plot
lin   - one dimensional plot along a field or stream or current line
blk   - 3D single block cell centered data, block specified point location
rfr   - radiotelescope pixel image plot
eqr   - field lines traced from the magnetic equatorial plane
eqb   - minimum B surface on a grid defined on the magnetic equatorial plane
ieb   - field lines traced from a subset of the IE coupled grid
lcb   - last closed field lines
buf   - coupling buffer between two components
```

The 1d, 2d and 3d cuts save the AMR tree information into a .tree file. This can be used for reconstructing the full grid and use the data with the READAMR library, for example.

For the PlotArea 'bx0' which is the bx=0 on z direction isosurface plot, an extra plot variable 'z' will be added as the first plot variable in addition to the PlotVar string. This extra plot variable 'z' records the position of the isosurface.

For IDL and cell centered Tecplot (tcp) plots the PlotArea = 'cut' can be used to create cuts.

The PlotVar string defines the plot variables and the equation parameters. It also controls whether or not the variables will be plotted in dimensional values or as non-dimensional values:

```
CAPITALIZED  - dimensional
lower case   - dimensionless

'var' - vars: READ FROM PARAMETER FILE
        pars: READ FROM PARAMETER FILE
'all' - vars: all state variables defined in the equation module
        pars: g
'hd'  - vars: Primitive_Variables
        pars: g rbody
'mhd' - vars: Primitive_Variables Jx Jy Jz
        pars: g rbody
'ful' - vars: Primitive_Variables B1x B1y B1z e Jx Jy Jz
        pars: g rbody
'raw' - vars: Conservative_Variables P b1x b1y b1z divb
        pars: g rbody
'ray' - vars: bx by bz lon1 lat1 lon2 lat2 status blk        (if DoMapEquatorRay=F)
        vars: bx by bz req1 phi1 req2 phi2 status blk        (if DoMapEquatorRay=T)
        pars: rbody
'eqr' - vars: iLine l x y z rho ux uy uz bx by bz p rCurve   (for all rays traced)
        pars: nRadius, nLon, nPoint
'eqb' - vars: z PrimVarMinB rCurve xZ0 yZ0 zZ0 PrimVarZ0 rCurveZ0  (B is in SM coordinates)
'flx' - vars: rho mr br p jr pvecr
        pars: g rbody
'bbk' - vars: dx pe blk blkall
        pars:
'pos' - vars  x y z                                          (PlotArea='lin' only)
        pars:
'sol' - vars: wl pb                                          (PlotArea='los' only)
        pars: mu
```

```
'lgq' - vars: squash.03 squash.12 squash.2 squash-15 squash-2 squash-3  (PlotArea='los' or
        pars: rbody
'euv' - vars: euv171 euv195 euv284                                (PlotArea='los' only)
        pars: mu
'sxr' - vars: sxr                                                 (PlotArea='los' only)
        pars: mu
'tbl' - vars: listed in the LOS table file                        (PlotArea='los' only)
        pars: mu
'dem' - vars: DEM, EM                                             (PlotArea='los' only)
        pars: rbody
'fux' - vars: flux
        pars: rbody
'nbi' - vars: intensity
        pars: rbody
'phx' - vars: flux
        pars: rbody
'int' - vars: 1/B n/B p/B                                         (PlotArea='lcb' only)
        pars:
'nul' - vars:                                                     (PlotArea='lcb' only)
        pars:
'ins' - vars: determined by the corresponding instrument, see line-of-sight below
      - pars: determined by the corresponding instrument, see line-of-sight below
```

Depending on StringPlot, the following parameters are also read from the parameter file in this order:

```
xMinCut...zMaxCut       if PlotArea is 'bx0', 'dpl', or 'slc'
Coord1MinCut...Coord3MaxCut     if PlotArea is 'cut'
nRadius nLon            if PlotArea is 'eqr' or 'eqb'
TypeCoordPlot           if PlotArea is 'shl', 'sln', 'slg' or 'box'
DivuDxMin               if PlotArea is 'shk'
RadiusMin RadiusMax     if PlotArea is 'eqr', 'eqb', 'shl', 'sln', 'slg', 'shk'
LonMin LonMax           if PlotArea is 'eqb', 'shl', 'sln', 'slg', 'shk'
LatMin LatMax           if PlotArea is 'shl', 'sln', 'slg', 'shk'
dRadius, dLon, dLat     if PlotArea is 'sln', 'slg' or 'shl' and range is nonzero.
dLon, dLat              if PlotArea is 'shk'
x0, y0, z0              if PlotArea is 'box'
xSize, ySize, zSize     if PlotArea is 'box'
dX, dY, dZ              if PlotArea is 'box' and associated range is nonzero.
xAngle, yAngle, zAngle  if PlotArea is 'box' (given in degrees)
xPoint yPoint zPoint    if PlotArea is 'slc', or 'blk'
xNormal yNormal zNormal if PlotArea is 'slc'
DxSavePlot              if PlotForm is 'idl' and PlotArea is not box/shl/sln/slg/shk/los/
NameVars                if PlotVar  is 'var' or 'VAR'
NamePars                if PlotVar  is 'var' or 'VAR'
```

Plotting range: the six parameters xMinCut ... zMaxCut define a 3D box in Cartesian coordinates. The Coord1MinCut ... Coord3MaxCut define a box in Cartesian or curvilinear coordinates.

For IDL plots, if the width in one or two dimensions is less than the smallest cell size within the plotarea, then the plot file will be 2 or 1 dimensional, respectively. This also works for non-Cartesian grids: the cut will be a 1D curve or a 2D surface aligned with the curvilinear coordinates. For example, from a spherical grid one can create a 1D cut along an arbitrary radial direction or along a circle, a 2D cut with fixed radius, fixed longitude or fixed latitude, or a spherical-wedge-shaped 3D cut. Note that the limits of the first coordinate are always given as true radial distance

(even for radially stretched spherical grids), while the longitude and latitude limits are given in degrees. The output file will contain 1, 2 or 3 of the radial, the longitude and latitude (in degrees) coordinates instead of the X, Y, Z coordinates. If possible, the data will be averaged to the 2D cut surface during the postprocessing.

For cell centered Tecplot files the cuts work the same way as for IDL, but 0 width cuts will produce two cells across instead of interpolating to the central plane. On the other hand, the cell centered Tecplot output retains the original AMR grid structure.

For Tecplot plots (PlotForm='tec') and PlotArea='dpl' or 'slc' the plot range clips the cut plane. For node based Tecplot files with PlotArea 'cut', the xMin .. zMax parameters are read but interpreted differently from IDL. Cuts are entire x, y, or z equal constant planes (1D or 3D cuts are not implemented). For x constant, for example, the y and z ranges do not matter as long as they are wider than the x range. The slice will be located at the average of xMinCut and xMaxCut. So, for example to save a plot in a x=-5 constant plane cut, the following can be used:

```
-5.01                   xMinCut
-4.99                   xMaxCut
-10.                    yMinCut
 10.                    yMaxCut
-10.                    zMinCut
 10.                    zMaxCut
```

The xPoint, yPoint, zPoint parameters give the coordinate of a point inside a grid block for PlotArea 'blk'. For PlotArea 'slc' they mean the coordinates of a point on the slice plane, and xNormal, yNormal, zNormal define a normal vector to the slice plane. If the normal in any given coordinate direction is less than 0.01, then no cuts are computed for cell edges parallel to that coordinate direction. For example, the following would result in only computing cuts on cell edges parallel to the Z axis.

```
 0.0                    xNormal
 0.0                    yNormal
 1.0                    zNormal
```

The DxSavePlot parameter determines the grid resolution for IDL files:

```
 positive value - uniform grid with cell size DxSavePlot in first coordinate
 0.             - uniform grid with smallest cell in the plotting area
-1.             - unstructured grid with original AMR cells
```

The line-of-sight (PlotArea 'los') plots calculate integrals along the lines of sight of some quantity and create a 2D Cartesian square shaped grid of the integrated values. Only the circle enclosed in the square is actually calculated and the corners are filled in with zeros. The image plane always contains the origin of the computational domain (usually the center of the Sun). By default the image plane is orthogonal to the observers position relative to the origin. The image plane can be rotated around the Z axis with an offset angle. If OffsetAngle is positive, a series of images are created covering the full circle with the OffsetAngle resolution. If OffsetAngle is negative, only one rotated image is created. By default the center of the image is the observer projected onto the image plane, but the center of the image can be offset. Since the central object (the Sun) contains extremely large values, an occultational disk is used to block the lines of sight going through the Sun. The variables which control the direction of the lines of sight and the grid position and resolution are the following:

```
ObsPosX,ObsPosY,ObsPosZ - the position of the observer in (rotated)
                          HGI coordinates (SC,IH and OH) or the GM coordinates
rSizeImage              - the radius of the LOS image
xOffset, yOffset        - offset relative to the observer projected onto
                          the image plane
rOccult                 - the radius of the occulting disk
MuLimbDarkening         - the limb darkening parameter for the 'wl'
                          (white light) and 'pb' (polarization brightness)
```

```
                          plot variables.
 nPix                   - the number of pixels in each direction
```

For line-of-sight Extreme Ultraviolet (EUV) and Soft X-Ray (SXR) plots, the same parametes are read as for the wl and pb plots (above) but now the integration is carried out to the surface of the sun so rOccult should be set to zero. MuLimbDarkening has no effect but needs to be included. Also, for line-of-sight (los) EUV images from STEREO-A/B and SDO/AIA using the response function table both 'ins' and 'INS' give the same dimensional output. Additionally, because EUV and SXR plots are configured to read in a response table specific to the EUV or SXR instument (e.g. SOHO EIT, STEREO EUVI, Yohkoh SXT) the tables for the response need to be read in by additional lines in the PARAM.in file. This follows the #LOOKUPTABLE command syntax e.g:

```
#LOOKUPTABLE
euv                     NameTable
load                    NameCommand
SC/Param/los_Eit.dat    NameFile
ascii                   TypeFile

#LOOKUPTABLE
sxr                     NameTable
load                    NameCommand
SC/Param/los_Sxt.dat    NameFile
ascii                   TypeFile
```

The possible values for NameVars with PlotArea 'los' are listed in subroutine set_plotvar_los in write_plot_los.f90.

The line-of-site (PlotArea 'los') plots have an option to use 'ins'/'INS', which will fill the ObsPosX, ObsPosY, ObsPosZ, rSizeImage, xOffset, yOffset, rOccult, MuLimbDarkening, nPix for SOME supported instruemts. An example is:

```
los ins idl_real4       StringPlot ! line of sight plot using table
-1                      DnSavePlot
100.                    DtSavePlot
soho:c3 sta:euvi stb:cor2       StringsInstrument
```

which is treated as ONE plot file in #SAVEPLOT, and the code would count how many instruments and expand the number of plot files after reading StringsInstrument. The StringsInstrument can contain multiple strings (maximum 200 characters). The supported combinations are:

```
Stereo A:    sta:euvi, sta:cor1, sta:cor2
Stereo B:    stb:euvi, stb:cor1, stb:cor2
SDO:         sdo:aia
Hinode:      hinode:xrt
SOHO:        SOHO:c2, SOHO:c3
```

The possible values for NameVars for other plot areas are listed in subroutine set_plotvar in write_plot_common.f90. For convenience and to avoid exceeding the line length limit of the PARAM.in file, the MHD and HD strings can be used in NameVars. These are replaced with the appropriate primitive variables (and jx jy jz for MHD), so one can add a few extra variables easily.

The possible values for NamePars are listed in subroutine set_scalar_param in write_plot_common.f90. The default string will be replaced with the default list of parameters, which include molecular masses (m1..m9) and charges (q1..q9) for each fluid, the length and time units (xSI and tSI) if different from 1, the adiabatic index (g) or indexes (g1..g9 if they are not equal, and the radius of the inner boundary (r) if present. The electron mass (me) is saved if there is an electron fluid, and the adiabatic index of electrons (ge) if it is different from gamma.

The refracting rays based plots (PlotArea 'rfr') plots calculate integrals along the curved rays (distorted by refraction) of the (radio) emissivity in the solar (stellar) corona and create a 2D Cartesian square shaped grid of the integrated intensity. Only the circle enclosed in the square is actually calculated and the corners are filled in with zeros. The image plane always contains the origin of the computational domain (usually the center of the Sun). The image plane is orthogonal to the line coonecting the observers position to the center of the Sun. The variables which control the direction of the lines of sight and the grid position and resolution are the following:

```
ObsPosX,ObsPosY,ObsPosZ - the position of the observer in the
                          coordinate system of the component
StringRadioFrequency    - the frequency or list of frequencies
xSizeImage, ySizeImage  - the size of the radio image
nPixX, nPixY            - the number of pixels in each direction
```

Most plot files are written in parallel: each processor writes out part of the data. These intermediate files are typically ASCII for the 'tec' and 'tcp' formats (see the #SAVETECBINARY command, which is useful for conversion to vtk format) and can be either binary or ASCII in 'idl' format as chosen with the #SAVEBINARY command (default is binary). The name of the files are

```
IO2/PlotArea_PlotVar_PlotNumber_TIMESTAMP_PEnumber.extension
```

where extension is 'tec' for the TEC/TCP and 'idl' for the IDL file formats. The PlotNumber goes from 1 to nPlotFilr in the order of the files in PARAM.in. The TIMESTAMP contains time step, simulation time or date-time information depending on the settings in the #SAVEPLOTNAME command.

After all processors wrote their plot files, processor 0 writes a small ASCII header file named as

```
IO2/PlotArea_PlotVar_PlotNumber_TIMESTAMP.headextension
```

where headextension is:

```
'T' for TEC/TCP file format
'h' for IDL file format
```

The line of sight integration produces TecPlot and IDL files directly:

```
IO2/los_PlotVar_PlotNumber_TIMESTAMP.extension
```

where extension is 'dat' for TecPlot and 'out' for IDL file formats.

The shell plot area ('shl', 'sln', 'slg') can be used to extract a spherical shell defined by radius, longitude and latitude ranges in the coordinate system given by TypeCoordPlot. If the range has extent zero in one or two coordinates, the shell becomes a 2D or 1D slice (for example 2D Lon-Lat, r-Lon, r-Lat surfaces, or 1D circle at fixed latitude, or a radial line with fixed longitude and latitude). The 'sln' and 'slg' are uniform in the logarithm of the radius, the former saves ln(r) into the file, the latter the 10-based lg(r). The meaning of the radial resolution dR becomes the size of the first cell at the minimum radius. The minimum and maximum radii are read as normal radii (not logarithm). The output is a single file in IDL or Tecplot format.

The shock surface ('shk') is extracted along radial lines started from a longitude-latitude grid. The surface is defined by the smallest value of div u*dx along each radial line. If the minimum div u*dx is larger than DivuDxMin (a negative value in velocity units), there is no shock surface. The output always contains DivuDx and the radial distance of the surface as additional plot variables.

The box plot area ('box') can be used to extract a Cartesian box defined by the center position and the size (length of the edges) and angles by which it is rotated around the axes of the coordinate system given by TypeCoordPlot. If the range has extent zero in one or two coordinates, the box becomes a 2D or 1D slice (for example 2D X-Y, X-Z, Y-Z surfaces, or 1D line along the X, Y or Z axis). The output is a single file in IDL or Tecplot format.

Default is nPlotFile=0, so no plot files are saved.

#RADIOEMISSION command

```
#RADIOEMISSION
simplistic                TypeRadioEmission
```

This command is used for 'rfr' plots (see #SAVEPLOT). It allows the selection of mechanisms for radio emission ('bremsstrahlung' or 'simplistic' mechanism, which interpolates between Bremsstrahlung and contributions from non-thermal emission at critical and quarter-of-critical densities, the different contributions being weighted quite arbitrarily. Default is 'simplistic'.

#NOREFRACTION command

```
#NOREFRACTION
T               UseNoRefraction
```

This command allows switching off the radio wave referaction to evaluate how the refraction affects the images obtained with 'rfr' plots (see #SAVEPLOT). Default is switched on (UseNoRefraction=F).

#SAVETECPLOT command

```
#SAVETECPLOT
T                         DoSaveOneTecFile
```

This command only works with 3D tecplot file (see #SAVEPLOT). It allows saving a single direct access formatted tecplot data/connectivity file. Post processing is still needed because the tecplot file is separated into 3 diffrent files: the header file, the data file and the connectivity file.

The default is false, which saves the tecplot data/connectivity for each processor. In some systems, saving the data/connectivity in a single file might not work.

The default value is F.

#SAVEPLOTNAME command

```
#SAVEPLOTNAME
T                         UsePlotNameStep
T                         UsePlotNameTime
F                         UsePlotNameDateTime
```

The TIMESTAMP of plot files (see #SAVEPLOT) can contain the time step in the _nSTEP format, the simulation time in the _tSIMTIME format and the date and time in the _eYYYYMMDD-HHMMSS-MSC format. Any combination of these logicals are allowed

The default values are UsePlotNameStep and UsePlotNameTime true and UsePlotNameDateTime false.

#SAVELOGNAME command

```
#SAVELOGNAME
T                         UseLogNameStep
F                         UseLogNameDateTime
```

The TIMESTAMP part of the names of logfiles (see #LOGFILE), satellite files (see #SATELLITE) and magnetometer files (see #MAGNETOMETER) can be controlled with the logicals UseLogNameStep and UseLogNameDateTime. If UseLogNameStep is true, the TIMESTAMP will contain the time step in the form _nTIMESTEP (see #NSTEP command). If UseLogNameDateTime is true, the TIMESTAMP will contain the date and time in the form _eYYYYMMDD-HHMMSS. If both logicals are true, both the step and the date-time will be in the TIMESTAMP. If both are false, the TIMESTAMP will be empty.

The default is UseLogNameStep true and UseLogNameDateTime false.

#SAVEBINARY command

```
#SAVEBINARY
T                       DoSaveBinary   used only for 'idl' plot file
```

Default is .true. Saves unformatted IO2/*.idl files if true. This is the recommended method, because it is fast and accurate. The only advantage of saving IO2/*.idl in formatted text files is that it can be processed on another machine or with a different (lower) precision. For example PostIDL.exe may be compiled with single precision to make IO2/*.out files smaller, while BATSRUS.exe is compiled in double precision to make results more accurate.

#SAVETECBINARY command

```
#SAVETECBINARY
F                       DoSaveTecBinary
```

If true, save Tecplot data and connectivity information in binary format. Currently this only works for 3D tec and tcp files. Note that the resulting files are not directly readable by Tecplot, but can be converted to other formats.

Default is false.

#PLOTFILENAME command

```
#PLOTFILENAME
hour                    NameMaxTimeUnit
```

For time accurate runs the plot filenames contain an 8-charcter timestamp string. The NameMaxTimeUnit string determines the content of this string.

If the longest time unit is hours or shorter, the string contains the simulation time. If the time unit is days or longer the string contains the physical date (set by the #STARTTIME command) and time information.

For NameMaxTimeUnit='hour' the string contains the simulation time described by a 4-character string for hours, and two 2-character strings for minutes and seconds, respectively. For NameMaxTimeUnit='hr' the string contains the simulation time described by a 2-character strings for hours, minutes, and seconds with a decimal point and one decimal digit. For NameMaxTimeUnit='minute' the first 2 characters describe the minutes, and the rest is seconds including 3 decimal digits. NameMaxTimeUnit='second' gives the simulation time up to 100 seconds with 5 decimal digits. NameMaxTimeUnit='millisecond' ('microsecond', 'nanosecond') give the simulation time up to 1000 milliseconds (microseconds, nanoseconds) with 4 decimal digits.

For time unit 'date' the full 14-character date-time string (YYYYMMDDHHMMSS) is used. For time units 'day', 'month', 'yr' and 'year' an 8-character-long substring of the date-time string is used. For NameMaxTimeUnit='year' the time stamp will contain the four digit year, and the two-digit month and day. For NameMaxTimeUnit='yr' the last two digits of the year, and the month, day and hour are used. For NameMaxTimeUnit='month' the month, day, hour, and minute are used. For NameMaxTimeUnit='day' the day, hour, minute and seconds are used. For NameMaxTimeUnit='timestep' only the timestep is used.

The #PLOTFILENAME command and the NameMaxTimeUnit parameter are saved into the restart header file so that the #PLOTFILENAME command does not have to be repeated in restarted runs (unless the unit is changed).

The default value is NameMaxTimeUnit='hour'.

#SAVEINITIAL command

```
#SAVEINITIAL
T                       DoSaveIntial
```

Save plots and log/satellite files at the beggining of the session. Default is DoSaveInitial=.false. except for the first time accurate session (when simulation time is zero) when the initial state is always saved.

#SAVEPLOTSAMR command

```
#SAVEPLOTSAMR
F                       DoSavePlotsAmr
```

Save plots before each AMR. Default is DoSavePlotsAMR=.false.

#FLUSH command

```
#FLUSH
F                       DoFlush
```

If the DoFlush variable is true, the output is flushed when subroutine ModUtility::flush_unit is called. This is used in the log and satellite files. The flush is useful to see the output immediately, and to avoid truncated files when the code crashes, but on some systems the flush may be very slow.

The default is to flush the output, i.e. DoFlush=T.

3.8.12 Eruptive event generator

#CME command

```
#CME
T                       UseCme ! Rest is read if UseCme is true
T                       DoAddFluxRope
2 h                     tDecayCme
0                       LongitudeCme   [deg]
0                       LatitudeCme    [deg]
0                       OrientationCme [deg]
GL                      TypeCme    GL/TD/TD14/TD22/SPHEROMAK
5.0                     BStrength      [Gs]
1                       iHelicity
1.03                    Radius         [Rs]
0.3                     Stretch        [Rs]
1.8                     ApexHeight     [Rs]

#CME
T                       UseCme ! Rest is read if UseCme is true
T                       DoAddFluxRope
-1.0                    tDecayCme
0                       LongitudeCme   [deg]
0                       LatitudeCme    [deg]
0                       OrientationCme [deg]
SPHEROMAK               TypeCme
5.0                     BStrength      [Gs]
-1                      iHelicity
1.03                    Radius         [Rs]
0.3                     Stretch        [Rs]
1.8                     ApexHeight     [Rs]
600                     uCme           [km/s]
```

If UseCme is false, no CME gnerator is applied. If UseCme is true, the CME generator is applied via the boundary condition. If, in addition to this, DoAddFluxRope is true, then the flux rope is added as a ”user perturbation” at the beginning of the session. The tDecayCme determines how long it takes for the CME related boundary conditions

to decay toward zero. If tDecayCme is -1, there boundary conditions do not decay. If tDecayCme is positive, the boundary conditions linearly decay from the original values to zero in tDecayCme time.

The LongitudeCme and LatitudeCme parameters characterize location of the superimposed configuration. They provide the longitude and latitude in degrees, of the configuration center. Third parameter, OrientationCme, chaaracterizes the CME orientation. The recommended value of OrientationCme is the counterclockwise angle, in degrees, between the local parallel (the horizontal axis of solar magnetogram), and the "major direction of the horizontal solar magnetic field" in the active region from which the eruption ooccurs. For different types of the eruptive event generator this major field direction may be quantified in somewhat different way. For example, for GL soluution this is the direction from the center of positive magnetic spot to that of the negative magnetic spot, which can be determined from the observed magnetogram for a simple bi-polar region. For the modified TD generator (TD14) this is the direction of the magnetic field acting on the super-imposed current filament, which may be found based on 3D reconstruction of the solar magnetic field. Depending on this input parameter, the simulated CME configuration will be properly orriented to better fit the observed solar magnetic field.

The latest implementation for the Gibson-Low eruptive event generator (TypeCme=GL) follows the paper Borovikov, D., I. V. Sokolov, W. B. Manchester, M. Jin, and T. I. Gombosi (2017), Eruptive event generator based on the Gibson-Low magnetic configuration, J. Geophys. Res. Space Physics, 122, 7979, doi:10.1002/2017JA024304.

BStrength (denoted as B0 in the cited paper) is the characteristic magnetic field, in Gauss, such that the magnetic field at the center of configuration (prior to stretching) equals about 0.7 B0.

The integer iHelicity defines positive (+1) or negative (-1) helicity by setting the sign of the poloidal field. The sign of the toroidal field is fixed, as it points from the positive to the negative spot of the active region.

The Radius parameter sets of the radius of the last magnetic (spherical) surface confining all currents (before stretching).

The Stretch parameter ("a" in the paper) is the scale of stretching transformation (see details in the paper). ApexHeight is the altitude of top of configuration from the solar surface. It is expected that Radius $<$ ApexHeight $<$ 2*Radius.

NOTE1: Before 08.15.2022 the sign of BStrength was used to set helicity. The negative sign corresponded to the positive helicity (which was essentially a bug).

NOTE2: If you have an outdated parameter file you can convert it to the new format as follows: 1. Move line for BStrength to have it just below "GL TypeCme" line and add the line setting iHelicity. 2. Move line for reading Radius just below the line for BStrength. 3. RESCALE the OLD data for BStrength as follows: BStrengthNew = 13.1687517342067082*BStrengthOld*Radius**2 4. The old line for reading Distance should be converted to the line for reading ApexHeight = Distance + Radius - Stretch - 1 5. Line for reading pBackground should be removed

In addition to the above parameters uCme is read, controlling the speed of the CME self-similar expansion

```
#CME
T                       UseCme
T                       DoAddFluxRope
180.0                   LongitudeCme  [deg]
15.0                    LatitudeCme   [deg]
30.0                    OrientationCme[deg]
TD                      TypeCme    TD/TITOV-DEMOULIN
+1                      iHelicty
0.5                     RadiusMajor  [R_s]
0.2                     RadiusMinor  [R_s]
0.2                     Depth        [R_s]
F                       UsePlasmaBeta
1.000E+16               Mass         [g]
readbstrap              TypeBStrap readbstrap/getbstrap/none
5                       bStrapping  [Gs]
steady                  TypeCharge  none/steady/moving/cancelflux
1.0                     BqFraction  [ ]
```

```
0.4                     qDistance   [R_s]
```

Version with UsePlasmaBeta=.true.: ... T UsePlasmaBeta PlasmaBeta 0.1 [ ] 5.0e4 EjectaTemperature [K] ... steady TypeCharge none/steady/moving/cancelflux 5 BStrapping [Gs] 0.4 qDistance [R_s]

version with the flux cancelation ... cancelflux TypeCharge none/steady/moving/cancelflux 5 BqStrapping [Gs] 0.4 qDistance [R_s] 5.0 ChargeUx [km/s]

```
#CME
T                       UseCme
T                       DoAddFluxRope
180.0                   LongitudeCme  [deg]
15.0                    LatitudeCme   [deg]
30.0                    OrientationCme[deg]
TD                      TypeCme    TD/TITOV-DEMOULIN
10.0                    BcTubeDim    [Gs]
0.5                     RadiusMajor  [R_s]
0.2                     RadiusMinor  [R_s]
0.2                     Depth        [R_s]
F                       UsePlasmaBeta
1.000E+16               Mass         [g]
readbstrap              TypeBStrap readbstrap/getbstrap/none
5                       bStrapping  [Gs]
steady                  TypeCharge  none/steady/moving/cancelflux
1.0                     BqFraction  [ ]
0.4                     qDistance   [R_s]
```

Version with UsePlasmaBeta=.true.: ... T UsePlasmaBeta PlasmaBeta 0.1 [ ] 5.0e4 EjectaTemperature [K] ... steady TypeCharge none/steady/moving/cancelflux 5 BStrapping [Gs] 0.4 qDistance [R_s]

BcTubeDim, in Gauss, is a magnetic field at the center of totoid, which field is created by the current inside the toroidal filament. RadiusMajor and RadiusMinor are characteristics of the toroid shape. Depth characterizes the toroid center location below the photosphere level. Mass in kilograms is an approximate integral of density over the volume of current filament.

To convert an outdated parameter file, with the Current in [A] and spatial scales in [m], one can convert it to the standard format as follows:

1. RESCALE the data for Current[A] as follows: BcTubeDim[Gs] = 2.0E-03*cPi*Current[A]/RadiusMajor[m]

2. Convert all spatial scales (RadiusMajor, RadiusMinor, Depth) in meters into those is R_S: Scale[R_s] = Scale[m]/6.96E+08 Or, if the scales are in megameters, the formulae are: Scale[R_s] = Scale[Mm]/6.96E+02

For TD configuration, magnetic field at the center of configuration is always parallel, while the starpping field is anti-parallel, to the x-axis. Phi-conponent of the toroidal current is positive. However, the sign of the field toroidal component may be both positive and negative, corresponding to the positive and negative helicity. To set the negative helicity, the input parameter, BcTube, should be negative. This choice affects only the sign of helicity, but not the direction of the poloidal magnetic field components, including the nagnetic field at the center of configuration.

```
#CME
T                       UseCme
T                       DoAddFluxRope
180.0                   LongitudeCme  [deg]
15.0                    LatitudeCme   [deg]
30.0                    OrientationCme[deg]
TD                      TypeCme    TD/TITOV-DEMOULIN
10.0                    BcTubeDim    [Gs]
0.5                     RadiusMajor  [R_s]
```

```
0.2                     RadiusMinor  [R_s]
0.2                     Depth        [R_s]
PlasmaBeta              0.1           [ ]
5.0e4                   EjectaTemperature [K]
readbstrap              TypeBStrap readbstrap/getbstrap/none
5                       bStrapping  [Gs]
steady                  TypeCharge  none/steady/moving/cancelflux
1.0                     BqFraction  [ ]
0.4                     qDistance   [R_s]
```

TypeCME=BREAKOUT should be described by Bart van der Holst. Please ask him for a description.
There is no CME by default.

#CMETIME command

```
#CMETIME
0.0                     tStartCme [sec]
```

The tStartCme variable contains the time in seconds when the flux-rope is added via the #CME command and is relative to the initial start time. It is saved into the restart header file, so if the CME run is restarted with DoAddFluxRope set to F and the tDecayCme is positive the CME related boundary conditions continue to decay to zero in tDecayCme time.

3.8.13 Amr parameters

#AMRINITPHYSICS command

```
#AMRINITPHYSICS
3                       nRefineLevelIC
```

Defines number of physics (initial condition) based AMR-s AFTER the geometry based grid refinement was finished. Only useful if the initial condition has a non-trivial analytic form.

#REGION command

required=”F” required=”F” required=”F”

```
#REGION
region1                 NameRegion
box                     StringShape
-64.0                   xMinBox
-16.0                   yMinBox
-16.0                   zMinBox
-32.0                   xMaxBox
 16.0                   yMaxBox
  0.0                   zMaxBox

#REGION
region2                 NameRegion
brick                   StringShape
-48.0                   xPosition
  0.0                   yPosition
 -8.0                   zPosition
```

```
 32.0                   xSizeBrick
 32.0                   ySizeBrick
 16.0                   zSizeBrick

#REGION
ellipsoid               NameRegion
sphere stretched        StringShape
-10.0                   xPosition
 10.0                   yPosition
  0.0                   zPosition
 20.0                   Radius
 30.0                   RadiusY  (only read if stretched)
 20.0                   RadiusZ  (only read if stretched)

#REGION
region3                 NameRegion
shell0                  StringShape
3.5                     Radius1
4.5                     Radius2

#REGION
region5                 NameRegion
cylinderx stretched tapered
-30.0                   xPosition
  0.0                   yPosition
  0.0                   zPosition
 60.0                   Length
 20.0                   Radius
 25.0                   RadiusPerp (only read if stretched)
  5.0                   Taper      (only read if tapered)

#REGION
region6                 NameRegion
ringz0 rotated          StringShape
  5.0                   Height
 20.0                   Radius1
 25.0                   Radius2
 10.0                   xRotate   (only read if rotated)
 10.0                   yRotate   (only read if rotated in 3D)
  0.0                   zRotate   (only read if rotated in 3D)

#REGION
region7                 NameRegion
conex stretched         StringShape
-30.0                   xPosition
  0.0                   yPosition
  0.0                   zPosition
 -5.0                   Height (base is at xPosition-5)
 20.0                   Radius
 30.0                   RadiusPerp (only read if stretched)

#REGION
```

```
region8                 NameRegion
funnelx stretched       StringShape
 10.0                   xPosition
 20.0                   yPosition
 30.0                   zPosition
 45.0                   Height
 10.0                   RadiusStart
 20.0                   Radius
 25.0                   RadiusPerp (only read if stretched)

#REGION
region9                 NameRegion
doubleconez0 tapered    StringShape
100.0                   Height
 20.0                   Radius
  2.0                   Taper

#REGION
region10                NameRegion
box weight tapered      StringShape
-30.0                   xMinBox
-30.0                   yMinBox
-2.0                    xMaxBox
30.0                    yMaxBox
2.0                     Taper
2.0                     Weight

#REGION
magnetosphere           NameRegion
paraboloidx stretched   StringShape
 10.0                   xPosition
  0.0                   yPosition
  0.0                   zPosition
-100.0                  Height
  30.0                  Radius
  20.0                  RadiusPerp

#REGION
myregion                NameRegion
user                    StringShape
```

The #REGION comand allows making a library of areas in the simulation domain identified by a unique NameRegion to be used in other commands (e.g. #AMRCRITERIALEVEL, #AMRCRITERIARESOLUTION, #HALLREGION) to define where some action (e.g. grid refinement) should be performed. In those commands the regions can be combined with + and - signs to form more complex shapes. For example

+dayside +tail -nearearth -nearaxis

The + sign means that a region is ”added” (more formally, take the union of the regions). The - sign means that the region is excluded. The best way to list multiple regions is to start with the + signs and finish with the - signs. If all regions have - signs, for example

-nearearth -nearaxis

then the interpretation is that they are excluded from the whole domain.

The StringShape parameter defines the shape of the region with the possible options. The basic shape names are the following: ’box’, ’box_gen’, ’brick’, ’brick_gen’, ’conex’, ’cylinderx’, ’doubleconex’, ’funnelx’, ’paraboloidx’,

'ringx', 'shell', 'sphere', and 'user'. The names that end with an 'x' indicate the orientation of the symmetry axis. These have alternative versions ending with 'y' and 'z', for example 'coney' and 'conez'. Most of these names can be followed by the optional '0', 'stretched', 'tapered' and 'rotated' strings as discussed below.

The area 'box' is a box aligned with the X, Y and Z axes, and it is given with the coordinates of two diagonally opposite corners. The area 'brick' has the same shape as 'box', but it is defined with the center of the brick and the size of the brick. The 'box_gen' and 'brick_gen' areas can be used for non-Cartesian grids to define a box in the generalized coordinates. For example a sphere around the origin can be described as a box in generalized coordinates with radius going from 0 to R, phi going from 0 to 360 degrees and latitude going from -90 to +90 degrees. Note that angles are given in degrees, and radius is given even if the generalized coordinates use its logarithm.

The area 'sphere' is a sphere around an arbitrary point, which is defined with the center point and the radius of the sphere. The area 'shell' consists of the volume between two concentric spherical surfaces, which is given with the center point and the two radii. The area 'cylinderx' is a cylinder with an axis parallel with the X axis, and it is given with the center, the length of the axis and the radius, The areas 'cylindery' and 'cylinderz' are cylinders parallel with the Y and Z axes, respectively, and are defined analogously as 'cylinderx'. The area 'ringx', 'ringy' and 'ringz' are the volumes between two cylindrical surfaces parallel with the X, Y and Z axes, respectively. The ring area is given with the center, the height and the two radii. The 'conex', 'doubleconex' and 'paraboloidx' are all aligned with the X axis and are described by the position of the tip, the height and the radius. The 'funnelx' is a cone with its tip chopped off. It is described by the position of the center of the starting circle, its height, the starting radius and the ending radius. The sign of the height specifies the orientation of the shape along its symmetry axis. Note that all these round shapes can be made elliptical with the "stretched" option (see below).

If the area name contains the number '0', the center/tip is taken to be at the origin and the Position coordinates are not read. Note that the areas 'box' and 'box_gen' are defined with the corners so the '0' cannot be used for these.

If the word 'stretched' is added after the area name, the shape can be stretched in all directions. This allows making an ellipsoid from a sphere, or an elliptical slab from a cylinder.

It the word 'tapered' is used in StringShape, the Taper parameter is read and the shape is surrounded by a tapering region of this width. This is useful when the region is used as a switch with a continuous transition between the inside and outside, see for example the #HALLREGION command.

If the word 'rotated' is added after the area name, the area can be rotated around the Z axis in 2D simulation, and by 3 angles around the X, Y and Z axes (in this order) in 3D simulations. These are the Tait-Bryan angles (yaw, pitch and roll) corresponding the X-Y-Z extrinsic rotations in a fixed coordinate system or Z-Y-X intrinsic rotations in a rotating coordinate system.

If the word 'weight' is used in StringShape, the parameter Weight is read. The weight is the Hall factor when this region is used by the #HALLREGION command.

If NameShape starts with 'user', the shape is defined by the subroutine user_specify_region. Any parameters for the user region should be read in the user section of the PARAM.in file.

By default there are no "regions" defined.

#GRIDRESOLUTION command

```
#GRIDRESOLUTION
2.0                     Resolution
initial                 StringShape

#GRIDLEVEL
3                       nLevel
all                     StringShape

#GRIDLEVEL
4                       nLevel
box                     StringShape
-64.0                   xMinBox
-16.0                   yMinBox
```

```
-16.0                   zMinBox
-32.0                   xMaxBox
 16.0                   yMaxBox
  0.0                   zMaxBox

#GRIDLEVEL
4                       nLevel
brick                   StringShape
-48.0                   xPosition
  0.0                   yPosition
 -8.0                   zPosition
 32.0                   xSizeBrick
 32.0                   ySizeBrick
 16.0                   zSizeBrick

#GRIDRESOLUTION
1/8                     Resolution
shell0                  StringShape
3.5                     RadiusInner
4.5                     Radius

#GRIDRESOLUTION
0.5                     Resolution
sphere                  StringShape
-10.0                   xPosition
 10.0                   yPosition
  0.0                   zPosition
 20.0                   Radius

#GRIDRESOLUTION
1/8                     Resolution
cylinderx               StringShape
-30.0                   xPosition
  0.0                   yPosition
  0.0                   zPosition
 60.0                   Height
 20.0                   Radius

#GRIDRESOLUTION
1/8                     Resolution
ringz0 rotated          StringShape
  5.0                   Height
 20.0                   RadiusInner
 25.0                   Radius
 10.0                   xRotate
 10.0                   yRotate
  0.0                   zRotate

#GRIDRESOLUTION
1/4                     Resolution
paraboloidx0 stretched  StringShape
30.0                    Height
10.0                    Radius
12.0                    RadiusPerp
```

```
#GRIDRESOLUTION
1/8                     Resolution
user                    StringShape
```

The #GRIDRESOLUTION and #GRIDLEVEL commands allow to set the desired (!) grid resolution or refinement level, respectively, in a given area. This grid resolution is only realized if either the StringShape='initial' resolution is set to an equal or finer refinement than the desired resolution, for example

```
#GRIDRESOLUTION
1/8                     Resolution
initial                 StringShape
```

or by applying adaptive mesh refinement during the run (see the #DOAMR command).

The Resolution parameter of the #GRIRDESOLUTION command usually refers to the size of the cell in the first direction (Dx or Dr), but for logarithmic/stretched radial coordinate (see #GRIDGEOMETRY), it refers to the resolution in the Phi coordinate in degrees. Note that this definition of resolution is different from that used in the #AMRCRITERIARESOLUTION command for non-Cartesian grids.

Normally it is best to use the #GRIDRESOLUTION command to define the desired grid as it describes the resolution in physical units, so it is independent of the number of the size of the domain and the number of root blocks. The alternative #GRIDLEVEL command can be useful for simple numerical tests, where an algorithm is tested on anon-uniform grid. For this command the nLevel parameter is an integer with level 0 meaning no refinement relative to the root block, while level N is a refinement by 2 to the power N.

Note that the #REGION commands in combination with the #AMRCRITERIALEVEL and #AMRCRITERIARESOLUTION commands allow even more flexibility in controlling the grid adaptation, but the initial resolution/level still has to be set by this command. However, if #AMRCRITERIALEVEL/AMRCRITERIARESOLUTION is specified, the user should not use #GRIDLEVEL/#GRIDRESOLUTION to specify any StringShape except 'initial', otherwise the code will just crash.

If StringShape is set to 'initial', it determines the number of grid adaptations used to initialize the grid. The grid adaptations are done according to the other #GRIDLEVEL, #GRIDESOLUTION commands. **The default is no refinement initially, which means that the grid is uniform at the beginning, and it is refined during the run according to the #AMR or #DOAMR commands. This means that one has to set the initial refinement level to get a non-uniform grid from the beginning.**

The StringShape 'all' refers to the whole computational domain, and it can be used to set the overall minimum resolution.

For other values of StringShape, the command specifies the shape of the area. where the blocks are to be refined. See the #REGION command for a description of these parameters. Note that "tapering" can only be used with the #REGION command.

If the desired grid resolution is finer than the initial resolution, then initially the grid will be refined to the initial resolution only, but the area will be further refined in subsequent pre-specified adaptive mesh refinements (AMRs) during the run (see the #AMR command). Once the resolution reaches the desired level, the AMR-s will not do further refinement. If a grid block is covered by more than one areas, the area with the finest resolution determines the desired grid resolution.

All computational blocks that intersect the area and have a coarser resolution than the resolution set for the area are selected for refinement.

The default is a uniform grid.

#AMRLEVELS command

```
#AMRLEVELS
0                       MinBlockLevel
99                      MaxBlockLevel
```

Set the minimum/maximum levels that can be affected by AMR. The usage is as follows:

```
MinBlockLevel .ge.0 Cells can be coarsened up to the listed level but not
                      further.
MinBlockLevel .lt.0 The current grid is ``frozen'' for coarsening such that
                      blocks are not allowed to be coarsened to a size
                      larger than their current one.
MaxBlockLevel .ge.0 Any cell at a level greater than or equal to
                      MaxBlockLevel is unaffected by AMR (cannot be coarsened
                      or refined).
MaxBlockLevel .lt.0 The current grid is ``frozen'' for refinement such that
                      blocks are not allowed to be refined to a size
                      smaller than their current one.
```

This command has no effect when DoAutoRefine is .false. in the #AMR command.

Note that the user can set either #AMRLEVELS or #AMRRESOLUTION but not both. If both are set, the final one in the session will set the values for AMR.

#AMRRESOLUTION command

```
#AMRRESOLUTION
0.                      DxCellMin
99999.                  DxCellMax
```

Serves the same function as AMRLEVELS. The DxCellMin and DxCellMmax parameters are converted into MinBlockLevel and MaxBlockLevel when they are read. Note that MinBlockLevel corresponds to DxCellMax and MaxBlockLevel corresponds to DxCellMin. See details above.

This command has no effect when DoAutoRefine is .false. in the #AMR command.

Note that the user can set either #AMRLEVELS or #AMRRESOLUTION but not both. If both are set, the final one in the session will set the values for AMR.

#DOAMR command

```
#DOAMR
T                       DoAmr (the rest is only read if true)
1                       DnAmr
-1.0                    DtAmr
T                       IsStrictAmr
```

DoAmr is telling if you do adaptive mesh refinement (AMR) during the simulation every DnAmr step or DtAmr intervals. For both DtAmr and DnAmr negative values mean that no AMR is performed based on that condition. If both values are positive then DnAmr is used in steady state mode and DtAmr is used in time accurate mode. If DtAmr is negative then DnAmr (has to be positive) is used always. If IsStrictAmr is true, we demand that the AMR is fully performed. If the AMR would require too many grid blocks, the code stops with an error message. If IsStrictAmr is false, the code will do a partial AMR allowed by the maximum of available blocks and continue running. For pure geometry based AMR the IsStrictAmr=F will cause the code to skip the complete AMR if there are not enough blocks.

Defaults are DoAmr false and IsStrictAmr true.

#AMRLIMIT command

```
#AMRLIMIT
40.                     PercentCoarsen
30.                     PercentRefine
999999                  MaxBlockAll
1e-8                    DiffCriteriaLevel
```

This is the obsolete way of doing AMR. All users are advised to use AMR where the grid depends on geometry and solution only, but not on the number of blocks. The #AMRCRITERIALEVEL and #AMRCRITERIARESOLUTION and #REGION commands provide all the needed functionality.

This command sets a desired percentage of blocks to be coarsened (PercentCoarsen) and refined (PercentRefine). In addition, the total number of grid blocks can be limited with MaxBlockAll. The criteria will be given by #AMRCRITERIA or #AMRCRITERIALEVEL. To maintain symmetry of the solution, it is useful to treat blocks with similar criteria value to be coarsened and refined together. The DiffCriteriaLevel gives the tolerance so that blocks with criteria values closer than DiffCriteriaLevel will be refined or coarsened together.

The default is to refine and coarsen blocks based on the criteria without any percentage limits.

#AMR command

```
#AMR
2001                    DnRefine
T                       DoAutoRefine   ! read if DnRefine is positive
0.                      PercentCoarsen ! read if DoAutoRefine is true
0.                      PercentRefine  ! read if DoAutoRefine is true
99999                   MaxTotalBlocks ! read if DoAutoRefine is true
```

This command is kept for backwards compatibility. The #DOAMR and #AMRLIMIT commands offer more control.

The DnRefine parameter determines the frequency of adaptive mesh refinements in terms of total steps nStep.

When DoAutoRefine is false, the grid is refined by one more level based on the areas and resolutions defined by the #GRIDLEVEL and #GRIDRESOLUTION commands. If the number of blocks is not sufficient for this pre-specified refinement, the code stops with an error.

When DoAutoRefine is true, the grid is refined or coarsened based on the criteria given in the #AMRCRITERIA command. The number of blocks to be refined or coarsened are determined by the PercentRefine and PercentCoarsen parameters. These percentages are approximate only, because the constraints of the block adaptive grid may result in more or fewer blocks than prescribed. The total number of blocks will not exceed the smaller of the MaxTotalBlocks parameter and the total number of blocks available on all the PE-s (which is determined by the number of PE-s and the MaxBlocks parameter in ModSize.f90).

Default for DnRefine is -1, i.e. no run time refinement.

#AMRCRITERIA command

```
#AMRCRITERIA
3                       nRefineCrit (1 to3)
gradP                   TypeRefine
0.2                     CoarsenLimit
0.8                     RefineLimit
user                    TypeRefine
0.5                     CoarsenLimit
0.5                     RefineLimit
Transient               TypeRefine
Rho_dot                 TypeTransient ! Only if 'Transient' or 'transient'
```

Note: "#AMRCRITERIALEVEL" gives even more control.

This command defines the criteria to select blocks for refinement or coarsening when the #AMR command is used with DoAutoRefine=T parameter. Up to 3 criteria can be used. Refinement is done if ANY of the criteria demand it, and the block can be refined (a block cannot be refined if the refinement level would exceed the maximum level or too many blocks would be created.

Coarsening is done if ALL the criteria allow it and the block can be coarsened (a block cannot be coarsened if the block level is already at the minimum level, or a neighboring block is finer).

The CoarsenLimit and RefineLimit parameters set the coarsening and refinement thresholds for the criteria that are given in I/O units.

If nRefineCrit is set to zero, the blocks are not ordered. This can be used to refine or coarsen all the blocks limited by the minimum and maximum levels only (see commands #AMRLEVELS and #AMRRESOLUTION). If nRefineCrit is 1, 2, or 3 then the criteria can be chosen from the following list (all criteria are based on the maximum over the cells in the grid block):

```
'gradP'               - gradient of pressure
'gradlogP'            - gradient of log10(P)
'pjumpratio'          - pmax/pmin in neighboring cells
'gradlogrho'          - gradient of log10(rho)
'J'                   - magnitude of current
'J2'                  - current squared
'currentsheet'        - current sheet (radial B changes sign)
'-divU'               - divergence of velocity times -1
'-divUdX'             - divergence of velocity times cell size times -1
'user'                - criteria defined in the user module
'transient'           - criteria is defined by the TypeTransient parameter.
```

The possible choices for TypeTransient:

```
'P_dot'       - relative change of pressure       (dP/dt)/P
'T_dot'       - relative change of temperature    (dT/dt)/T
'Rho_dot'     - relative change of density        (drho/dt)/rho
'RhoU_dot'    - relative change of momentum       (d|rhoU|/dt)/|rhoU|
'B_dot'       - relative change of magnetic field (d|B|/dt)/|B|
'meanUB'      - max[(d|rhoU|/dt)/|rhoU|] * max[(d|B|/dt)/|B|]
'Rho_2nd_1'   - (|d2Rho/dx2| + |d2Rho/dy2| + |d2Rho/dz2|)/rho
'Rho_2nd_2'   - (|d2Rho/dx2  +  d2Rho/dy2  +  d2Rho/dz2|)/rho
```

By default there are no criteria, so all blocks are refined or coarsened together.

#AMRCRITERIALEVEL command

```
#AMRCRITERIALEVEL
5                       nCriteria
J2 +tail -nearbody      TypeCriteria
0.1                     CoarsenLimit
0.75                    RefineLimit
1                       MaxLevel
J2 +tail -nearbody      TypeCriteria
1.0                     CoarsenLimit
2.0                     RefineLimit
2                       MaxLevel
level                   TypeCriteria
2                       RefineTo
3                       CoarsenFrom
dx +nearbody            TypeCriteria
0.5                     RefineTo
0.25                    CoarsenFrom
transient P_dot         TypeCriteria
1.0                     CoarsenLimit
2.0                     RefineLimit
```

```
1                       MaxLevel
T                       UseSunEarth   ! Only if there are any 'transient' crit
0.00E+00                xEarth        ! Only if UseSunEarth is true
2.56E+02                yEarth        ! Only if UseSunEarth is true
0.00E+00                zEarth        ! Only if UseSunEarth is true
5.00E-01                InvD2Ray      ! Only if UseSunEarth is true

#AMRCRITERIARESOLUTION
3                       nCriteria
dphi                    TypeCriteria
3.0                     RefineTo
1.5                     CoarsenFrom
dphi Innershell         TypeCriteria
1.5                     RefineTo
0.75                    CoarsenFrom
currentsheet            TypeCriteria
0.5                     CoarsenLimit
0.5                     RefineLimit
1.5                     MaxResolution

#AMRCRITERIACELLSIZE
3                       nCriteria
J2 +tail -nearbody      TypeCriteria
0.1                     CoarsenLimit
0.75                    RefineLimit
0.25                    MaxResolution
J2 +tail -nearbody      TypeCriteria
1.0                     CoarsenLimit
2.0                     RefineLimit
0.125                   MaxResolution
error Bx -nearbody      TypeCriteria
0.025                   CoarsenLimit
0.1                     RefineLimit
0.5                     MaxResolution
1.0e-2                  SmallError    ! Only if there are any 'error' crit
```

The #AMRCRITERIALEVEL, #AMRCRITERIARESOLUTION or #AMRCRITERIACELLSIZE command defines the criteria to select blocks for refinement or coarsening when the #DOAMR command is used with DoAmr=T parameter. In one session you can only have one #AMRCRITERIALEVEL, #AMRCRITERIARESOLUTION or #AMRCRITERIACELLSIZE command. #AMRCRITERIARESOLUTION or #AMRCRITERIACELLSIZE is equvilent with #AMRCRITERIALEVEL but works with cell size (MaxResolution) instead of grid level (MaxLevel).

The #AMRCRITERIARESOLUTION command defines the criteria to refine / coarsen based on the physical cell size in the first dimension (dx/dr) except for logarithmic or generalized radial coordinate when the size in the phi direction is used in degrees. Typecriteria =”dx/dr/dphi” can be used.

The #AMRCRITERIACELLSIZE command defines the criteria to refine / coarsen based on the maximum length of the cell edges inside a block. For non-cartesian grid this results in varying AMR level but roughly uniform cell sizes for a given resolution. Note that this is different from the definition used in the #GRIDRESOLUTION command.

The number of criteria is given by the nCriteria parameter. For each criteria the first parameter TypeCriteria determines its type. TypeCriteria=”level” and ”dx/dr/dphi” are geometric criteria, while any other values (that depend on the solution) are non-geometric. Up to 3 different types of non-geometric criteria can be used, but there can be multiple criteria (with different criteria levels and/or geometric restrictions) for the same non-geometric TypeCriteria.

For the geometric criteria there are two additional parameters read: RefineTo and CoarsenFrom. These are given either as grid level (for TypeCriteria="level") or as grid resolution (for TypeCriteria="dx/dr/dphi"). In the above example the "level" criteria tries to refine the grid to level 2 (and coarsen from level 3 down to level 2) everywhere in the computational domain. The "dx" criteria above tries to refine to a grid resolution 0.5 (and coarsen from 0.25 to 0.5) inside the region named "nearbody" that has to be defined by the #REGION command.

For non-geometric criteria there are three additional parameters read: CoarsenLimit, RefineLimit and MaxLevel (for #AMRCRITERIALEVEL) or MaxResolution (for #AMRCRITERIARESOLUTION, #AMRCRITERIACELLSIZE). The TypeCriteria determines how the criterion value (a positive real number in "I/O" units) is calculated. In the above example TypeCriteria="J2" is the largest value of the current density squared inside the grid block. The CoarsenLimit and RefineLimit are positive real numbers that are compared to the criterion value, for every grid block. If the criterion value is above the RefineLimit then the block will be refined if it has not yet reached the grid level or grid resolution defined by the MaxLevel (for #AMRCRITERIALEVEL) or MaxResolution (for #AMRCRITERIARESOLUTION, #AMRCRITERIACELLSIZE) parameter. If the criterion value is below the CoarsenLimit then the block is allowed to get coarsened according to this criterion.

Refinement is done if ANY of the criteria demand it, and the block can be refined. A block cannot be refined if the refinement level would exceed the maximum grid level or too many blocks would be created.

Coarsening is done if ALL the criteria allow it and the block can be coarsened. A block cannot be coarsened if the block level is already at the minimum level or it has a neighbor block that is and remains finer.

By default the AMR criteria are applied in the whole simulation domain. This can be limited to a certain area by adding +REGIONNAME and -REGIONNAME modifiers to the end of the TypeCriteria string. The unique REGIONNAME names have to be defined in the #REGION command(s), where the definition of the region is given (for example a sphere, or a box). See the #REGION command for a description of how to combine multiple regions.

TypeCriteria can be chosen from the following list:

```
'dx'                  - refinment based on (max) cell size in a block
'level'               - refinment based on the grid level of a block
'gradT'               - gradient of temperature
'gradP'               - gradient of pressure
'gradlogrho'          - gradient of log(rho)
'gradlogP'            - gradient of log(P)
'gradE'               - gradient of electric field magnitude
'curlV','curlU'       - magnitude of curl of velocity
'curlB'               - magnitude of current density
'J2'                  - square of current density
'currentsheet'        - current sheet (radial B changes sign)
'divU', 'divV'        - divergence of velocity
'user'                - criteria defined in the user module
```

For TypeRefine="transient TypeTransient" there are the following possibiities:

```
'transient P_dot'     - relative change of pressure       (dP/dt)/P
'transient T_dot'     - relative change of temperature    (dT/dt)/T
'transient Rho_dot'   - relative change of density        (drho/dt)/rho
'transient RhoU_dot'  - relative change of momentum       (d|rhoU|/dt)/|rhoU|
'transient B_dot'     - relative change of magnetic field (d|B|/dt)/|B|
'transient meanUB'    - max[(d|rhoU|/dt)/|rhoU|] * max[(d|B|/dt)/|B|]
'transient Rho_2nd_1' - (|d2Rho/dx2| + |d2Rho/dy2| + |d2Rho/dz2|)/rho
'transient Rho_2nd_2' - (|d2Rho/dx2  +  d2Rho/dy2  +  d2Rho/dz2|)/rho
```

For TypeRefine="error StateVarName" the criteria is a numerical error estimate for the state variable StateVarName. The error estimation is based on the second and first derivatives:

```
                          d^2 U
```

$$E = \frac{\frac{1}{DX}\frac{dU}{dx} + \frac{SmallError}{DX^2} * U + Epsilon}{\frac{dx^2}{7}}$$

The SmallError parameter gives a relative error with respect to the mean value of the state variable. This parameter is read as the last parameter of the command if there are any "error" type criteria.

A useful tool to see the values of the various criteria is to plot the 'crit1'..'crit9' plot variables with the #SAVEPLOT command just before the AMR(s).

The default setting is nCriteria = 0. This can be used to refine or coarsen all the blocks limited by the minimum and maximum levels only (see commands #AMRLEVELS and #AMRRESOLUTION).

3.8.14 Scheme parameters

#UPDATE command

```
#UPDATE
slow                    TypeUpdate (orig/slow/fast)
```

The TypeUpdate parameter determines which implementation of the update is used. The value "orig" is the original implementation that works on CPU only. The value "slow" is the original implementation but parameters are forced to match the fast implementation for sake of debugging. Works on CPU only. The value "fast" is the implementation for the GPU, but it can also be run on the CPU.

The default value is "orig" if the code is compiled for CPU, and "fast" when compiled for the GPU.

#UPDATEVAR command

```
#UPDATEVAR
rho mx my               StringVarUpdate

#UPDATEVAR
all                     StringVarUpdate
```

Update only a subset of the state variables. The variables should be listed with a single space separator in the StringVarUpdate. If StringVarUpdate is set to 'all' then all the variables are updated.

If density is updated but some components of the momentum are not, then the velocity is preserved (not the momentum). In the first example above, the Z component of the velocity is fixed. In the current implementation this only works with classical momentum (not with semi-relativistic momentum, see #BORIS command) and only for the first fluid.

The default is to update all the variables.

#SCHEME command

```
#SCHEME
5                       nOrder (1, 2 or 5)
Rusanov                 TypeFlux
1.2                     LimiterBeta ! Only read if TypeLimiter is NOT 'minmod'
```

The nOrder parameter determines the spatial and temporal accuracy of the scheme. The spatially first order scheme uses a one-stage time integration. The spatially second order MUSCL scheme either uses an explicit two-stage Runge-Kutta or an implicit three-level BDF2 time discretization. The spatially 5th order schemes uses the 3rd order Runge-Kutta scheme.

NOTE 1: The 5th order scheme requires 3 ghost cells and at least 6x6x6 grid blocks (to be set with Config.pl -g=... -ng=...). The 1st and 2nd order schemes work with 2 ghost cells and can have 4x4x4 blocks.

NOTE 2: the time discretization scheme can be modified with the #TIMESTEPPING, #RUNGEKUTTA or #RK commands after the #SCHEME command.

Possible values for TypeFlux:

```
'Rusanov'     - Rusanov or Lax-Friedrichs flux
'Linde'       - Linde's HLLEL flux
'Sokolov'     - Sokolov's Local Artificial Wind flux
'LFDW'        - Lax-Friedrichs + Dominant-Wave (Andrea Mignone)
'HLLDW'       - HLLE + Dominant-Wave           (Andrea Mignone)
'HLLD'        - Miyoshi and Kusano's HLLD flux
'Roe'         - Roe's approximate Riemann flux (new)
'RoeOld'      - Roe's approximate Riemann flux (old)
'Godunov'     - Godunov flux with exact Riemann solver
'Simple'      - Physical fluxes are applied without any Riemann solver.
```

The Rusanov, Linde, Sokolov, LFDW and HLLDW schemes are general for any equation set. The Rusanov scheme is the most diffusive (and robust), the HLLDW scheme is the least diffusive. The Godunov flux is only implemented for (multi-material) hydrodynamics. The Roe and HLLD schemes are implemented for ideal MHD only (single fluid, non-relativistic, no Hall term). The new and old Roe schemes differ in some details of the algorithm, the new Roe scheme is somewhat more robust in magnetospheric applications. The Simple solver is for testing purposes only at this point.

No limiter is used by the 1st order scheme. The second order TVD scheme uses a TVD limiter everywhere. The 5th order schemes has its own 5th order accurate limiter (see #SCHEME5 command). The TypeLimiter is still used inside the region specified by the #LOWORDERREGION command and where the stencil is not large enough for the high order scheme but sufficient for the second order scheme, which happens near face boundaries (see the #BOXBOUNDARY and #INNERBOUNDARY command). Possible values for TypeLimiter:

```
'minmod'      - minmod limiter is the most robust and diffusive limiter
'mc'          - monotonized central limiter with a beta parameter
'mc3'         - Koren's third order limiter with a beta parameter
'beta'        - beta limiter is less robust than the mc limiter for
                the same beta value
```

Possible values for LimiterBeta (for limiters othen than minmod) are between 1.0 and 2.0:

```
LimiterBeta = 1.0 is the same as the minmod limiter
LimiterBeta = 1.5 is a typical value for the mc/mc3 limiters
LimiterBeta = 1.2 is the recommended value for the beta limiter
LimiterBeta = 2.0 for the beta limiter is the same as the superbee limiter
```

The default is the second order Rusanov scheme with the minmod limiter.

#LOWORDERREGION command

```
#LOWORDERREGION
+nearbody +faraway              StringLowOrderRegion
```

This command is only useful if the nOrder is larger than 2 in the #SCHEME command. In this case the StringLowOrder-Region string can specify a region where the low (second) order scheme is used. The regions must be described with the #REGION command. A linear combination of a low order and high order face value is used in the tapering region.

The default is to apply the high order scheme everywhere.

#ADAPTIVELOWORDER command

```
#ADAPTIVELOWORDER
T                       UseAdaptiveLowOrder
2                       nLowOrder
2.0                     PCritLow
1.5                     PCritHigh
2.0                     VelCrit
```

The role of this command is similar to #LOWORDERREGION. #LOWORDERREGION selects the faces use 1st/2nd order face values based on the geometry, while this comamd selectes low order faces based on local physical conditions, which are total pressure jump and normal velocity difference in the current implementation. The low order value could be 1st or 2nd, which is set by nLowOrder.

For each face, its 6 neighbor cells (3 cells on each side) are used as criteria. Among these 6 cells, let's denote the ratio between maximum and minimum pressure as pRatio, and the difference between the lagest and smallest march number as dVel. When dVel is smaller than VelCrit, then the high order schemes are used. When dVel is larger than VelCrit, further check the value of pRatio. If pRatio is larger/smaller that pCritLow/pCritHigh, then a low/high order face value will be used, otherwise, use a linear combination of the low and high order value.

#SCHEME5 command

```
#SCHEME5
T                        UseFDFaceFlux
MP5                      TyperLimiter5
T                        UseHighResChange
T                        UseHighOrderAMR
T                        DoCorrectFace
```

UseFDFaceFlux is meaningful only when nOrder is 5 (see #SCHEME). If it is true, a finite difference space discretization, which is 5th order accurate for nonlinear equations, is used. Otherwise, the finite volume based discretization, which is 2nd order accurate except for 1D linear equations, is applied.

TypeLimiter5 can be MP5 or CWENO, which is used to limit 5th order space interpolation. The MP5 scheme is recommended.

If UseHighResChange is true the ghost cells are filled in with 5th order accurate values at the grid resolution changes so the scheme becomes 5th order accurate even at the resolution changes. If it is set to false, we switch to the second order prolongation algorithm (see #PROLONGATION) and also switch on the DoConserveFlux parameter of the #CONSERVEFLUX command.

If UseHighOrderAMR is true, 5th order interpolation is used for grid refinement and coarsening, so the scheme is 5th order accurate even with dynamic AMR. If false, the second order refinement and coarsening algorithms are used.

DoCorrectFace is true by default when 5th order FD scheme is used. The face values are corrected so that the 1st order derivatives df/dx are 5th order accurate when DoCorrectFace is true.

NOTE 1: This command has no effect unless nOrder is set to 5 in the #SCHEME command.

NOTE 2: this command has to be used after the #SCHEME command, because the #SCHEME command sets the default values.

NOTE 3: The DoConserveFlux parameter of the #CONSERVEFLUX can be overwritten with the #CONSERVEFLUX command AFTER the #SCHEME5 command.

The default values are shown above (assuming nOrder=5 is set in #SCHEME).

#CONSERVEFLUX command

```
#CONSERVEFLUX
T               DoConserveFlux
```

Correct face flux near resolution change to keep conservation.

The default is true in general. The only exception is when the 5th order finite difference scheme is used with UseFDFaceFlux set to true (see #SCHEME5). The default may be overwritten with this command after the #SCHEME and #SCHEME5 commands.

#NONCONSERVATIVE command

```
#NONCONSERVATIVE
T               UseNonConservative
```

If UseNonConservative is false, the total energy density equation is solved everywhere, and the pressure is derived from the total energy density. If UseNonConservative is true, then the pressure equation is solved, and the total energy density is calculated from the pressure and the kinetic and magnetic energy densities either everywhere (if nConservCrit=0), or in the regions defined in the #CONSERVATIVECRITERIA command. For further control of neutral fluids see the #NEUTRALFLUID command.

The default is using the conservative equations.

#CONSERVATIVECRITERIA command

```
#CONSERVATIVECRITERIA
3               nConservCrit
r               TypeConservCrit
6.              rConserv             ! read if TypeConservCrit is 'r'
parabola        TypeConservCrit
6.              xParabolaConserv     ! read if TypeConservCrit is 'parabola'
36.             yParabolaConserv     ! read if TypeConservCrit is 'parabola'
p               TypeConservCrit
0.05            pCoeffConserv        ! read if TypeConservCrit is 'p'
GradP           TypeConservCrit
0.1             GradPCoeffConserv    ! read if TypeConservCrit is 'GradP'
```

Select the parts of the grid where the conservative vs. non-conservative schemes are applied. The number of criteria is arbitrary, although there is no point applying the same criterion more than once.

If no criteria is used, the whole domain will use conservative energy density or non-conservative pressure equations depending on UseNonConservative set in command #NONCONSERVATIVE.

The physics based conservative criteria ('p' and 'GradP') select cells which use the non-conservative scheme if ALL of them are true:

```
'p'      - the pressure is smaller than fraction pCoeffConserv of the energy
'GradP'  - the relative gradient of pressure is less than GradPCoeffConserv
```

The geometry based criteria are applied after the physics based criteria (if any) and they select the non-conservative scheme if ANY of them is true:

```
'r'        - radial distance of the cell is less than rConserv
'parabola' - x less than xParabolaConserv - (y**2+z**2)/yParabolaConserv
```

The default is to have no conservative criteria: nConservCrit = 0.

#UPDATECHECK command

```
#UPDATECHECK
T                       UseUpdateCheck
40.                     RhoMinPercent
```

```
400.                    RhoMaxPercent
40.                     pMinPercent
400.                    pMaxPercent
```

Note that the "update-check" algorithm controlled by this command does not work together with high order Runge-Kutta schemes (see the #RK command) because the RK method combines the intermediate stages for the final update. Use the time step control method (see #TIMESTEPCONTROL and related commands) in combination with RK time stepping. In general, for time accurate simulations the time step control method has more flexibility and it is likely to be more effective and efficient than this update-check method.

If UseUpdateCheck is true, the local or global time step will be adjusted so that the density and pressure does not decrease or increase by more than the given percentages in a single timestep. For example with the default settings, if density is 1.0 initially and it would change below 0.6 or above 5.0, the (local) time step will be reduced so that the final density remains inside the prescribed bounds.

Default is UseUpdateCheck false. For Runge-Kutta schemes UseUpdateCheck is forced to be false.

#CHECKTIMESTEP command

```
#CHECKTIMESTEP
T               DoCheckTimeStep
2               DnCheckTimeStep
1e-6            TimeStepMin
```

This command is only effective in time accurate mode.

If DoCheckTimeStep is true, then check if average time step is smaller than TimeStepMin every nCheckTimeStep time steps. If it is, save the output files (but not restart) and stop the code.

Default is DoCheckTimeStep false.

#CONTROLTIMESTEP command

```
#CONTROLTIMESTEP
T                       UseTimeStepControl

#TIMESTEPCONTROL
T                       UseTimeStepControl
```

Setting UseTimeStepControl=T switches on the new time step control scheme that controls the time step based on the relative change in selected set of variables. The variables can be selected with the #CONTROLVAR command. The various thresholds in the relative increase and decrease of these variables can be set by the #CONTROLINCREASE and #CONTROLDECREASE commands. The #CONTROLFACTOR command determines how much the time step changes when the various thresholds are reached.

Currently this scheme only works in time accurate mode.

The default is UseTimeStepControl false.

#CONTROLINIT command

```
#CONTROLINIT
0.01                    TimeStepControlInit
```

Set the initial reduction factor applied to the time step or Cfl number. The factor should be positive and it should typically not more than 1.

The default value is 1, i.e. there is no initial reduction applied.

#CONTROLVAR command

```
#CONTROLVAR
rho p                   NameVarControl
```

The NameVarControl string contains the list of variables that are monitored to control the time step. The variable names, separated by spaces, should be chosen from the NameVar_V(1:nVar) array in the equation module. The names are not case sensitive. Typically only the positive variables, like density and pressure, should be monitored.

Note that this command is only effective if the time step control is switched on by th #CONTROLTIMESTEP command.

The default is the control density and pressure as shown by the example.

#CONTROLDECREASE command

```
#CONTROLDECREASE
0.3                     RejectStepLevel
0.6                     ReduceStepLevel
0.8                     IncreaseStepLevel
```

This command sets thresholds for the relative decrease in the control variables in the time step control scheme. The relative decrease is defined as D = min(VarNew/VarOld) where the minimum is taken over all cells in the computational domain and all the control variables.

If D is below the RejectStepLevel threshold, the time step is rejected, and it will be redone with a smaller time step/CFL number.

If D is above RejectStepLevel but below the ReduceStepLevel then the time step is accepted, but the next time step/CFL number will be reduced.

If D is above RejectStepLevel but below IncreaseStepLevel, the time step is accepted and there is no change in the time step/CFL number.

If is above the IncreaseStepLevel threshold, then the time step/CFL number is increased, but it will never exceed the original value.

This command is only effective if the time step control is switched on with the #CONTROLTIMESTEP command. The control variables are selected by the #CONTROLVAR command, the factors that change the time step or the CFL number are set by the #CONTROLFACTOR command.

Default values are shown.

#CONTROLINCREASE command

```
#CONTROLINCREASE
3.0                     RejectStepLevel
1.5                     ReduceStepLevel
1.2                     IncreaseStepLevel
```

This command sets thresholds for the relative increase in the control variables in the time step control scheme. The relative increase is defined as I = max(VarNew/VarOld) where the maximum is taken over all cells in the computational domain and all the control variables.

If I is above the RejectStepLevel threshold, the time step is rejected, and it will be redone with a smaller time step/CFL number.

If I is below RejectStepLevel but above the ReduceStepLevel then the time step is accepted, but the next time step/CFL number will be reduced.

If I is below ReduceStepLevel but above IncreaseStepLevel, the time step is accepted and there is no change in the time step/CFL number.

If I is below the IncreaseStepLevel threshold, then the time step/CFL number is increased, but it will never exceed the original value.

This command is only effective if the time step control is switched on with the #CONTROLTIMESTEP command. The control variables are selected by the #CONTROLVAR command, and the factors that change the time step or the CFL number are set by the #CONTROLFACTOR command.

Default values are shown.

#CONTROLFACTOR command

```
#CONTROLFACTOR
0.5                     RejectStepFactor
0.95                    ReduceStepFactor
1.05                    IncreaseStepFactor
```

This command sets how much the time step/CFL number is changed by the time step control scheme.

If the update is rejected then the next time step/CFL factor is multiplied by RejectStepFactor.

If the update is accepted but the time step needs to be reduced, then the next time step/CFL factor is multiplied by ReduceStepFactor.

If the update is accepted and the relative changes in the control variables are within the range determined by the IncreaseStepLevel parameters of the #CONTROLDECREASE and #CONTROLINCREASE commands, then the time step/CFL number is multiplied by IncreaseStepFactor, but the original values cannot be exceeded.

This command is only effective if the time step control is switched on with the #CONTROLTIMESTEP command. The control variables are selected by the #CONTROLVAR command.

Default values are shown.

#MULTISPECIES command

```
#MULTISPECIES
T                       DoReplaceDensity
1.0                     SpeciesPercentCheck
```

This command is only useful for multispecies equations. If the DoReplaceDensity is true, the total density is replaced with the sum of the species densities. The SpeciesPercentCheck parameter determines if a certain species density should or should not be checked for large changes. If SpeciesPercentCheck is 0, all species are checked, if it is 1, then only species with densities reaching or exceeding 1 per cent are checked for large changes (see the #UPDATECHECK command).

Default values are shown.

#NEUTRALFLUID command

```
#NEUTRALFLUID
F               DoConserveNeutrals
Linde           TypeFluxNeutral (default, Rusanov or Linde)
```

If DoConserveNeutrals is false, the pressure equation is used for neutrals even when the energy equation is used for the ions. If DoConserveNeutrals is true, the neutrals do the same as ions. The neutral fluid uses the flux function set by TypeFluxNeutral. The default is to use the same as the ion fluid if possible. Currently only the Rusanov and Linde (HLLE) schemes are available for the neutrals. If the ion fluid uses any other flux function, the neutrals will use the Linde scheme.

Default values are DoConserveNeutrals=T and TypeFluxNeutral=default.

#MULTIION command

```
#MULTIION
0.0001                  LowDensityRatio
1e-10                   LowPressureRatio
T                       DoRestrictMultiIon
3.0                     MachNumberMultiIon (read if DoRestrictMultiIon)
30.0                    ParabolaWidthMultiIon (read if DoRestrictMultiIon)
```

This command is useful for multiion simulations. Since the numerical schemes cannot handle zero densities or temperatures, it is necessary to have all the ions present in the whole computational domain. The parameters of this command determine how the code behaves in regions where one of the ions is dominant.

The LowDensityRatio parameter determines the relative density of the minor ion fluids in regions where essentially only one ion fluid is present.

The LowPressureRatio parameter is used to keep the pressures of the minor fluids above a fraction of the total pressure.

If DoRestrictMultiIon is true, the first ion fluid is set to be dominant in the region determined by the MachNumberMultiIon and ParabolaWidthMultiIon parameters. The current parametrization tries to find the region occupied by the solar wind outside the bow shock. The region is identified as the velocity being negative and the hydrodynamic Mach number in the X direction is being larger than MachNumberMultiIon and the point being outside the paraboloid determined by the $y^2 + z^2 + x * ParabolaWidthMultiIon = 0$ equation.

The defaults are LowDensityRatio=0.0001, LowPressureRatio=1e-10, and DoRestrictMultiIon=false.

#MULTIIONSTATE command

```
#MULTIIONSTATE
T                       UseSingleIonVelocity
F                       UseSingleIonTemperature
```

This command allows to enforce uniform ion velocities and/or temperatures in multi-ion simulations. When both logicals are true, the multi-ion simulation should become equivalient with a singlefluid multi-species simulation. This is useful for testing.

When UseSingleIonVelocity is true, the ion velocities are set to the average fluid velocity $u = \sum\_s(\rho\_su\_s)/\sum\_s\rho\_s$ as if there was an infinitely strong friction force between the ion fluids.

When UseSingleIonTemperature is true, the ion temperatures are set to the average temperature $k\_BT = \sum\_sp\_s/\sum\_s(\rho\_s/M\_s)$ as if there was an infinitely fast energy exchange between the ion fluids.

Default values are false for both parameters.

#COLLISION command

```
#COLLISION
-1.0                            CollisionCoefDim
1.0e3                           TauCutOffDim [s]
100.0                           uCutOffDim [km/s] read if TauCutOffDim positive
2                               nPowerCutOff read if TauCutOffDim positive
```

This command is only useful for multiion simulations. It determines the parameters for physical collisions and artificial friction.

If the CollisionCoefDim parameter is negative the ion-ion collisions are neglected. This is typically a very good approximation in the low density plasma of space physics. The collisions may be important in the ionosphere of unmagnetized planets. For positive value the collision rate is taken to be CollisionCoefDim$*n/T^{1.5}$ where T is the temperature measured in Kelvin, n is the number density measured in $/cm^{-3}$ and the resulting rate is in units of $1/s$. Note that this feature is implemented but it has not been tested yet.

The TauCutOffDim parameter determines if the relative velocity between ion fluids should be limited and at what rate. If TauCutOffDim is positive, it gives the time rate of the friction. If the TauCutOffDim parameter is negative the relative velocity of the ion fluids (especially parallel to the magnetic field) can become very large. In reality the streaming instability limits the relative speed. Instead of trying to model the streaming instability directly, in the current implementation we apply a simple friction term.

The uCutOffDim determines the speed difference (in input units, typically km/s), at which the friction term becomes large. Setting uCutoffDim = -1.0 switches to a physics based cut-off velocity which is defined as B/sqrt[rho1*rho2/(rho1+rho2)] where B is the magnetic field magnitude, and rho1 and rho2 are the densities of the two ion fluids in normalized units. Setting uCutoffDim = -2.0 applies a cut-off velocity based on the total Alfven speed B/sqrt(rho1+rho2).

The nPowerCutOff is the exponent applied to the square of the velocity difference. The friction force is applied between all pairs of ion fluids, and it is

$(1/TauCutOffDim)\min(\rho\_i, \rho\_j)(u\_j - u\_i)[(u\_i - u\_j)^2/uCutOffDim^2]^{nPowerCutOff}$.

where i and j are the indexes of two different ion fluids and u is the velocity vector. Note that the friction force is proportional to the smaller of the two densities so that the acceleration of the minor ion fluid is independent of the density of the major ion fluid.

The default values are CollisionCoefDim=-1 and TauCutOffDim=-1, ie. neither collision, nor friction are applied.

#MESSAGEPASS command

```
#MESSAGEPASS
all                     TypeMessagePass
```

Possible values for TypeMessagePass:

```
'all'       - fill in all ghost cells (corners, edges and faces)
'opt'       - fill in face ghost cells only
```

The default value is 'all', because there are many schemes that require the ghost cells at the edges and corners (viscosity, resistivity, Hall MHD, radiative diffusion, accurate resolution change algorithm, etc.). These will automatically change to the 'all' option even if the user sets "opt", which is only recommended for advanced users.

#OPTIMIZEMPI command

```
#OPTIMIZEMPI
F                       UseOptimizeMpi
```

If UseOptimizeMpi is true, then the two explicit MPI barriers are switched off, which may help with MPI performance. This feature is not fully tested. The default is false.

#RESOLUTIONCHANGE command

```
#RESOLUTIONCHANGE
F               UseAccurateResChange
T               UseTvdResChange
2.0             BetaLimiterResChange
2               nFaceLimiterResChange
```

If UseAccurateResChange is true, then a second order accurate, upwind and oscillation free scheme is used at the resolution changes. It requires message passing edge ghost cells (this is switched on automatically) which may effect the performance slightly.

If UseTvdResChange is true, then an almost second order and partially downwinded TVD limited scheme is used at the resolution changes. This scheme does not require message passing of the edge ghost cells. Only one of UseAccurateResChange and UseTvdResChange can be true.

If BetaLimiterResChange is set to a value smaller than the BetaLimiter parameter in the #SCHEME command, then the limiter will use this BetaLimiterResChange parameter at and near grid resolution changes. The smallest value is 1.0 that corresponds to the minmod limiter, the maximum value is 2.0 that means that the same limiter is applied at the resolution change as anywhere else. Recommended values are 1.0 to 1.2 combined with BetaLimiter=1.5 in the #SCHEME command.

The nFaceLimiterResChange determines how many faces around the resolution change itself are affected. If nFaceLimiterResChange is 0, the limiter using BetaLimiterResChange is applied at the face at the resolution change itself. If nFaceLimiterResChange is 1 or 2, the limiter is applied at 3 or 5 faces altogether. The recommended value is 2.

Default values are shown, ie. the TVD reschange algorithm is used, and the limiter applied at the resolution changes is the same as everywhere else, because BetaLimiterResChange is set to 2.

#RESCHANGE command

```
#RESCHANGE
T               UseAccurateResChange
```

This command is kept for backwards compatibility. See description at the #RESOLUTIONCHANGE command.

#TVDRESCHANGE command

```
#TVDRESCHANGE
T               UseTvdResChange
```

This command is kept for backwards compatibility. See description at the #RESOLUTIONCHANGE command.

#PROLONGATION command

```
#PROLONGATION
2               nOrderProlong  (1 or 2)
```

The nOrderProlong parameter determines if the fine ghost cells are filled in with first order or second order accurate values at the resolution change. The first order value is simply a copy of the coarse cell that covers the fine ghost cell. The second order value is obtained from linear interpolation of coarse cells covering and surrounding the fine ghost cell. This command sets the ”UseAccurateResChange” and ”UseTvdResChange” parameters to false (see #RESOLUTIONCHANGE command). Since the subcylcing algorithm (see #SUBCYCLING) is not quite compatible with the ”accurate res. change” and ”TVD res. change” algorithms, this command provides an alternative approach that is compatible.

Note that the 5th order scheme (see #SCHEME command) uses a 5th order accurate prolongation procedure unless the ”UseHighResChange” parameter is set to false in the #SCHEME5 command.

The default is using 1st order prolongation for the first order scheme (nOrder=1 in the #SCHEME command) the ”TVD reschange” algorithm for the 2nd order scheme, and the 5th order prolongation for the 5th order scheme.

#LIMITER command

```
#LIMITER
F               UseLogRhoLimiter
F               UseLogPLimiter
T               UseRhoRatioLimiter
Xe Be Pl        NameVarLimitRatio (read if UseRhoRatioLimiter)
```

The spatially second order scheme uses a limited reconstruction to obtain face values from the cell center values. The order of the scheme and the type of the limiter can be set in the #SCHEME command. This command provides additional options to the limiting procedure.

If UseLogRhoLimiter is true, the logarithm of the density is limited instead of the density itself. This can reduce numerical diffusion in regions where the density changes exponentially with distance (e.g. in the solar corona).

If UseLogPLimiter is true, the logarithm of pressure is limited instead of the pressure itself.

If UseRhoRatioLimiter is true, then parameter NameVarLimitRatio is read and the variables listed in NameVarLimitRatio (the variable names are defined in ModEquation) are divided by the total density before the limiter is applied and then multiplied back by the density at the face after the limiting is completed. This modification is useful for the high energy density simulations of the CRASH project for the level set functions or for the internal energy associated with ionization.

Default values are false for all variables, which results in the limited reconstruction procedure directly applied to the original primitive variables (velocity and pressure).

#CLIMIT command

```
#CLIMIT
T                       UseClimit (rest of parameters are read if true)
3000.0                  ClimitDim [km/s]
6.0                     rClimit
```

If UseClimit is true, the wave speeds used in the numerical diffusive fluxes are limited by the value of ClimitDim (in I/O units, typically km/s) within the sphere of radius rClimit (typically in units of planetery radii). This scheme cannot be used with a fully explicit time integration, because it will not be stable! One should use the fully or part implicit scheme (see the #IMPLICIT command). In contrast with the Boris correction (see the #BORIS command), this scheme is fully consistent with the governing equations in time accurate mode as well. It can be combined with the Roe scheme too unlike the Boris correction. The limiting scheme cannot be combined with the HLLD scheme (neither can be the Boris correction at this point).

A reasonable set of values are shown above. Much smaller velocity limit will result in slow convergence for the implicit solver. The radial limit is not very crucial, but it should be set large enough to cover the whole region where the wave speed may exceed it and the reduced diffusion is important.

Default is UseClimit false.

#BORIS command

```
#BORIS
T                       UseBorisCorrection
1.0                     BorisClightFactor !Only if UseBorisCorrection is true
```

If UseBorisCorrection is set to true and there is only a single ion fluid, then the semi-relativistic MHD equations are solved. If there are multiple ion fluids, the code automatically switches to the ”simple Boris correction” described at the #SIMPLEBORIS command.

The semi-relativistic MHD equations limit the Alfven speed to the speed of light. The speed of light can be artificially reduced by the BorisClightFactor. Set BorisClightFactor=1.0 for true semi-relativistic MHD. BorisClightFactor less than 1 can be used to allow larger explicit time steps and to reduce the numerical diffusion. Typical values are 0.01 to 0.02, which set the speed of light to 3,000km/s and 6,000km/s, respectively. Note that semi-relativistic MHD gives the same steady state solution as normal MHD analytically, but there can be differences due to discretization errors, in particular the Boris correction reduces the numerical diffusion. See Toth et al. 2011 (Journal of Geophysical Research, 116, A07211, doi:10.1029/2010JA016370) for an in-depth discussion.

See also the #BORISSIMPLE command as an alternative. Note that you cannot set both UseBorisCorrection and UseBorisSimple to true.

Default is UseBorisCorrection=.false.

#BORISSIMPLE command

```
#BORISSIMPLE
T                       UseBorisSimple
0.05                    BorisClightFactor !Only if UseBorisSimple is true
```

Use simplified semi-relativistic MHD. For single fluid MHD this means that the time derivative of the momentum density is multiplied with a factor $(1 + vA^2/c^2)$, which reduces the change of velocity. For the multi-ion MHD the JxB - grad(Pe) forces acting on the fluids are reduced by the same factor (which has a similar effect).

The speed of light can be reduced by the BorisClightFactor. This scheme is only useful with BorisClightFactor less than 1. The single fluid case should give the same steady state as normal MHD, but there can be a difference due to discretization errors. The multi-ion MHD case will not even give the same steady state analytically as the unmodified multi-ion MHD. You can use either Boris or BorisSimple but not both. For multi-ion MHD only the simple Boris scheme is available.

Default is UseBorisSimple=.false.

#BORISREGION command

```
#BORISREGION
+nearbody       NameBorisRegion
```

This command can be used to limit the effect of the (simple) Boris correction (see #BORIS and #SIMPLEBORIS) to a region defined by one or more #REGION commands. Outside this region the semi-relativistic equations are solved with the true speed of light, while inside the Boris region the speed of light is reduced. It is probably a good idea to use tapering (see the #REGION command) so that the speed of light changes gradually at the edges of the region.

The default is to use the reduced speed of light everywhere.

#B0 command

```
#B0
F               UseB0
```

If UseB0 is true, the magnetic field is split into an analytic B0 and a numerical B1 field. The B0 field may be a (rotating) dipole of a planet, or the potentialf field solution for the corona. B1 is not small relative to B0 in general. The default value depends on the application.

#CURLB0 command

```
#CURLB0
T               UseCurlB0
2.5             rCurrentFreeB0    (read if UseCurlB0 is true)
T               UseB0MomentumFlux (read if UseCurlB0 is true)
```

If UseCurlB0 is true, then the B0 field has non-zero curl. The B0 field of planets has zero curl, but the potential field source surface model (PFSS) for the corona has a finite curl beyond the source surface, where the field is forced to become radial.

The rCurrentFreeB0 parameter is set to the radius within which the B0 field has no curl (i.e. it is current free).

If UseB0MomentumFlux is true, the contribution from B0 field to momentum source is calculated as $div(B0B0) - gradB0^2/2 - B0divB0$ otherwise as $curlB0 \times B0$. Although mathematically identical, these expressions are numerically different.

The default is UseCurlB0 false in general, but it is set to true if the radius of the B0 grid generated by #HARMONICSGRID command or by FDIPS is less than the radius of the solar corona domain. In this case rCurrentFreeB0 is set to the radius of the B0 grid, while the default for UseB0MomentumFlux is false.

#LIGHTSPEED command

```
#LIGHTSPEED
10.0                    cLightDim
```

Set speed of light used in the Maxwell equations. Reducing the speed of light artificially will allow larger explicit time steps. The speed of light should be larger than the typical wave speeds present in the problem.

Default is the true speed of light.

#FORCEFREEB0 command

```
#FORCEFREEB0
T                       UseForceFreeB0
```

Define the B0 field to be force-free but with non-zero J0=curl B0. The force-free property means that J0 x B0 = 0 in the momentum and energy equations. UseForceFreeB0=T switches on curl B0 in the whole domain, so there is no need for the #CURLB0 command.

By default the B0 field is curl free.

#HYPERBOLICDIVE command

```
#HYPERBOLICDIVE
0.1                     HypEDecay
```

This command sets the decay rate for the hyperbolic/parabolic constraint for the div E = charge density condition when we solve for the electric field. The hyperbolic cleaning is always applied with the speed of light. In addition the scalar HypE decays as HypE=HypE*(1-HypEDecay) if HypEDecay is set to a positive value. If HypEDecay is less than zero, no parabolic decay is applied.

Default is HypEDecay=0.1.

#DIVB command

```
#DIVB
T                       UseDivbSource
F                       UseDivbDiffusion
F                       UseProjection
F                       UseConstrainB
```

Default values are shown above. If UseProjection is true, all others should be false. If UseConstrainB is true, all others should be false. At least one of the options should be true unless the hyperbolic cleaning is used. The hyperbolic cleaning can be combined with UseDivbSource only.

#B0SOURCE command

```
#B0SOURCE
T                       UseB0Source
F                       UseDivFullBSource (read if UseB0Source is true)
```

If UseB0Source is true, add extra source terms related to the non-zero divergence and curl of B0 in the momentum equation to cancel out numerical errors in the divergence of the Maxwell tensor.

If UseDivFullBSource is also true, use div(B1+B0) in the induction and energy equations instead of div(B1) in the 8-wave scheme.

Default values are shown.

#DBTRICK command

```
#DBTRICK
F                       UseDbTrick
```

This "trick" tries to maintain positivity when the conservative MHD energy equation is used by adding $dB^2/2$ to the energy density where dB is the change of the magnetic field relative to the previous time step. This trick is done either at the half step of the time accurate 2-stage scheme or in the (1 or 2-stage) local time stepping scheme. In steady state the correction is zero becasuse dB=0. For the time-accurate 2-stage scheme, the correction done at the half step is $O(dt^2)$ because dB is proportional to the time step, so the trick is consistent and still second order accurate.

Default is true if the trick is compatible with other settings. In particular, the trick is switched off for Runge-Kutta schemes.

#LORENTZFORCE command

```
#LORENTZFORCE
F                     UseJCrossBForce
```

By default the single ion MHD equations use the divergence of the Maxwell tensor to calculate the momentum fluxes (UseJCrossBForce is false), while in the multiion equations the force acting on each fluid is calculated as (qi/qe)*(J x B), where qi and qe are the charge densities of the ion fluid and the electrons, respectively. An alternative approach is to use (qi/qe)*Ec, where Ec is the electric field in the comoving (moving with the ions) frame, and Ec is obtained from the divergence of the Maxwell tensor. The advantage of the latter approach is that the equation for the total momentum will be in conservation form, since sum(qi) = qe.

In the future, we may switch the default value of UseJCrossBForce=F for multiion MHD too. For now, it can be set with this command.

#HYPERBOLICDIVB command

```
#HYPERBOLICDIVB
T                       UseHyperbolicDivb
400.0                   SpeedHypDim
0.1                     HypDecay
```

This command sets the parameters for hyperbolic/parabolic cleaning. The command (and the hyperbolic cleaning method) can only be used if there is a hyperbolic scalar named Hyp in the equation module. The SpeedHypDim parameter sets the propagation speed for div B errors in dimensional units. Do not use a speed that limits the time step (ie. exceeds the fastest wave speed). The HypDecay parameter is for the parabolic cleaning. If HypDecay is less than zero, no parabolic cleaning is applied. If it is positive, the scalar field is modified as Hyp=Hyp*(1-HypDecay) after every update. This corresponds to a point implicit evaluation of a parabolic diffusion of the Hyp scalar.

Default is UseHyperbolicDivb false.

#PROJECTION command

```
#PROJECTION
cg                      TypeProjectIter:'cg' or 'bicgstab' for iterative scheme
rel                     TypeProjectStop:'rel' or 'max' error for stop condition
0.1                     RelativeLimit
0.0                     AbsoluteLimit
50                      MaxMatvec (upper limit on matrix.vector multipl.)
```

Default values are shown above.

For symmetric Laplacian matrix TypeProjectIter='cg' (Conjugate Gradients) should be used, as it is faster than BiCGSTAB. In current applications the Laplacian matrix is always symmetric.

The iterative scheme stops when the stopping condition is fulfilled:

```
TypeProjectStop = 'rel':
     stop if ||div B||    < RelativeLimit*||div B0||
TypeProjectStop = 'max' and RelativeLimit is positive:
     stop if max(|div B|) < RelativeLimit*max(|div B0|)
TypeProjectStop = 'max' and RelativeLimit is negative:
     stop if max(|div B|) < AbsoluteLimit
```

where ||.|| is the second norm, and B0 is the magnetic field before projection. In words 'rel' means that the norm of the error should be decreased by a factor of RelativeLimit, while 'max' means that the maximum error should be less than either a fraction of the maximum error in div B0, or less than the constant AbsoluteLimit.

Finally the iterations stop if the number of matrix vector multiplications exceed MaxMatvec. For the CG iterative scheme there is 1 matvec per iteration, while for BiCGSTAB there are 2/iteration.

In practice reducing the norm of the error by a factor of 10 to 100 in every iteration works well.

Projection is also used when the scheme switches to constrained transport. It is probably a good idea to allow many iterations and require an accurate projection, because it is only done once, and the constrained transport will carry along the remaining errors in div B. An example is

```
#PROJECTION
cg                      TypeProjIter
rel                     TypeProjStop
0.0001                  RelativeLimit
0.0                     AbsoluteLimit
500                     MaxMatvec
```

3.8.15 Coupling paramaters

#TRACE command

```
#TRACE
T                       UseTrace (rest is read if true)
T                       UseAccurateTrace
0.1                     DtExchangeTrace [sec]
1                       DnTrace
```

Tracing (field-line tracing) is needed to couple the GM with the IM or RB components. It can also be used to create plot files with open-closed field line information. There are two algorithms implemented for integrating field lines and for tracing field lines.

By default UseTrace parameter is true if there is magnetic field in the equation module. The parameter can be set to false to save memory allocation.

If UseAccurateTrace is false (default), the block-wise algorithm is used, which interpolates at block faces. This algorithm is fast, but less accurate than the other algorithm. If UseAccurateTrace is true, the field lines are followed all the way. It is more accurate but potentially slower than the other algorithm.

In the accurate tracing algorithms, when the line exits the domain that belongs to the PE, its information is sent to the other PE where the line continues. The information is buffered for sake of efficiency and to synchronize communication. The frequency of the information exchanges (in terms of CPU seconds) is given by the DtExchangeTrace parameter. This is an optimization parameter for speed. Very small values of DtExchangeTrace result in many exchanges with few lines, while very large values result in infrequent exchanges thus some PE-s may become idle (no more work to do). The optimal value is problem dependent. A typically acceptable value is DtExchangeTrace = 0.1 seconds (default).

The DnTrace parameter contains the minimum number of iterations between two tracings. The default value 1 means that every new step requires a new trace (since the magnetic field is changing). A larger value implies that the field does not change significantly in that many time steps. The tracing is always redone if the grid changes due to an AMR.

Default values are UseAccurateIntegral = .true. (if there is magnetic field), UseAccurateTrace = .false., DtExchangeTrace = 0.1 and DnTrace=1.

#TRACERADIUS command

```
#TRACERADIUS
2.5             rTrace
-1.0            rIonosphere
```

This command sets the inner boundary of field tracing. If rTrace is negative, there is no inner boundary for the tracing. If rTrace is positive then the field/stream lines are traced to rTrace. If rIonosphere is also positive, then the magnetic field lines are further traced along the dipole field down to rIonosphere. If rIonosphere is negative then this is not done.

For the GM component the default is rIonosphere=1 if there is a central dipole field. For other cases the default is rIonosphere=-1. For rTrace the default is the larger of the ionosphere and body radii. If there is no ionosphere and no body then the default is rTrace=-1.

#TRACELIMIT command

```
#TRACELIMIT
50                      TraceLengthMax
```

TraceLengthMax provides the maximum length for tracing a field/stream line. Setting a small limit can avoid tracing extremely long field lines that are not used later. The default is 200 units.

#TRACEACCURACY command

```
#TRACEACCURACY
5.0                      AccuracyFactor
```

Set the accuracy of the tracing algorithm. The default accuracy is optimized for speed. Setting AccuracyFactor to a larger value reduces the step size proportionally. Factor 5 may avoid failed tracing. Factor 20 is close to fully converged accuracy. The higher accuracy means that more time is spent on the field line tracing.

Default is AccuracyFactor=1.

#SQUASHFACTOR command

```
#SQUASHFACTOR
360                     nLonSquash
180                     nLatSquash
20.0                    AccuracyFactorSquash
```

Set the resolution of the spherical grid at the inner boundary on which the squash factor is calculated. Also set the accuracy of the tracing used for the squash factor calculation.

Default values are shown above.

#TRACETEST command

```
#TRACETEST
35                      iLonTest
10                      iLatTest
```

Set the longitude and latitude indexes of the tested trace. This is useful to test an individual trace line starting from a spherical or cylindrical grid. The subroutine to be tested still needs to be set with the #TEST command.

Default values are 1 for both indexes.

#TRACEIE command

```
#TRACEIE
T                       DoTraceIE
```

DoTraceIE will activate accurate ray tracing on closed field lines for coupling with the IE module. If not set, then only Jr is sent. If set, then Jr as well as 1/B, average rho, and average p on closed field lines are passed. This command is required (!) for the MAGNIT conductance model in IE/RIM.

Default is DoTraceIE false.

#IECOUPLING command

```
#IECOUPLING
T                       UseIonoVelocity (rest of parameters read if true)
4.0                     rCoupleUiono
10.0                    TauCoupleUiono
```

This command sets parameters for a new experimental coupling of the velocity from IE to GM.

The rCoupleUiono paramter determines the radius within which the GM velocity is effected. The TauCoupleUiono parameter determine how fast the GM velocity should be nudged towards the E x B drift plus corotation.

coupling occurs, but the nudging towards the velocity is done in every GM time step. When GM is not run in time accurate mode, the orthogonal (to B) velocity is set as

uOrth' = uOrth + (uIonoOrth - uOrth)/(TauCoupleUiono+1)

Therefore the larger TauCoupleUiono is the slower the adjustment will be. It takes approximately 2*TauCoupleUiono time steps to get the orthogonal velocity close to what the ionosphere would prescribe. In time accurate mode, the nudging is based on physical time:

uOrth' = uOrth + min(1.0, dt/TauCoupleUiono)*(uIonoOrth - uOrth)

where dt is the time step. It takes about 2*TauCoupleUiono seconds to get uOrth close to uIonoOrth. If the time step dt exceeds TauCoupleIm, uOrth is set in a single step.

By default the coupling is switched off.

#IM command

```
#IM
20.0                    TauCoupleIm
F                       DoImSatTracing
```

Same as command IMCOUPLING, except it only reads the first and second parameters of #IMCOUPLING.

The default value is TauCoupleIm=20.0, which corresponds to typical nudging and DoImSatTrace false.

#IMCOUPLING command

```
#IMCOUPLING
20.0                    TauCoupleIm
F                       DoImSatTrace
T                       DoCoupleImPressure
F                       DoCoupleImDensity
0.01                    DensityCoupleFloor (read if DoCoupleImDensity is true)
T                       DoFixPolarRegion (rest read if true)
5.0                     rFixPolarRegion
20.0                    PolarNDim [amu/cc] for fluid 1
100000.0                PolarTDim [K]      for fluid 1
2.0                     PolarNDim [amu/cc] for fluid 2
20000.0                 PolarTDim [K]      for fluid 2
```

This command sets various parameters for the GM-IM coupling.

The TauCoupleIm parameter determines how fast the GM pressure p (and possibly density rho) should be relaxed towards the IM pressure pIm (and density dIM). but the relaxation towards these values is done in every GM time step. When GM is not run in time accurate mode, the pressure is set as

```
p' = (p*TauCoupleIm + pIm)/(TauCoupleIm+1)
```

Therefore the larger TauCoupleIm is the slower the adjustment will be. It takes approximately 2*TauCoupleIm time steps to get p close to pIm. In time accurate mode, the relaxation is based on physical time:

```
p' = p + min(1.0, dt/TauCoupleIm)*(pIm - p)
```

where dt is the time step. It takes about 2*TauCoupleIm seconds to get p close to pIm. If the time step dt exceeds TauCoupleIm, p' = pIm is set in a single step. The default value is TauCoupleIm=20.0, which corresponds to typical relaxation rate.

The DoImSatTrace logical sets whether the IM component receives the locations of the satellites in GM mapped down along the magnetic field lines. The IM component then can produce satellite output files with IM data.

The DoCoupleImPressure logical sets whether GM pressure is driven by IM pressure. Default is true, and it should always be true (except for testing), because pressure is the dominant variable in the IM to GM coupling.

The DoCoupleImDensity logical sets whether the GM density is relaxed towards the IM density.

The DensityCoupleFloor parameter is read if DoCoupleImDensity is true. If DensityCoupleFloor is positive, it sets a minimum density floor for every fluid coupled between GM and IM. This avoids situations where very low densities in the ring current model would push the BATS-R-US densities to very low values, which can cause numerical problems. If a floor value is necessary, the recommended value is 0.01 amu/cc.

The DoFixPolarRegion logical decides if we try to fix the pressure (and density) values in the open field line region. The pressure/density tends to diffuse numerically from the closed field line region (controlled by IM) into the polar region that should not be affected by IM. This can cause unphysically fast outflow from the polar region. If DoFixPolarRegion is set to true, the pressure (and density) are relaxed toward the values given in the #POLARBOUNDARY command in the open field line region within radius defined by rFixPolarRegion and where the flow points outward.

If DoFixPolarRegion is true then the following parameters are also read:

The rFixPolarRegion radius (given in planetary radii) sets the outer limit for relaxing the pressure (density) in the open field line region towards the PolarNDim and PolarTDim values. For multi-fluid MHD, the PolarNDim and PolarTDim parameters are read for each fluid.

The default is to couple the IM pressure only and no fix is applied in the polar region.

#IMCOUPLINGSMOOTH command

```
#IMCOUPLINGSMOOTH
10.0                    dLatSmoothIm [deg]
```

Smooth out the pressure and density nudging at the edge of the IM boundary. The nudging is ramped up linearly within dLatSmootIm degrees along the magnetic latitude direction. Default is -1.0, which means no smoothing.

#MULTIFLUIDIM command

```
#MULTIFLUIDIM
F                       DoMultiFluidIMCoupling
```

If DoMultiFluidIMCoupling is true, the information exchanged between GM and IM is in multi-fluid mode: GM gives IM four more variables (density_Hp, density_Op, pressure_Hp, pressure_Op) in addition to one-fluid MHD paramters, and IM passes GM the same four more variables.

The default value is DoMultiFluidIMCoupling = false, MHD variables are exchanged between GM and IM.

#ANISOPRESSUREIM command

```
#ANISOPRESSUREIM
F                       DoAnisoPressureIMCoupling
```

If DoAnisoPressureIMCoupling is true, the information exchanged between GM and IM allows for pressure anisotropy. This only makes sense if BATSRUS is configured with anisotropic pressure equations and the IM model allows for non-isotropic pressure (which is all models except RCM).

The default value is DoAnisoPressureIMCoupling = false, which means that isotropy is assumed in the coupling (even if both GM and IM allow for anisotropy).

#PSCOUPLING command

```
#PSCOUPLING
20.0                    TauCouplePs
T                       DoCouplePsPressure
T                       DoCouplePsDensity
.1                      DensityCoupleFloor
```

This command controls density and pressure coupling from the plasmasphere (PS) component into BATS-R-US. TauCouplePs sets the rate at which MHD fluids are ”nudged” towards the PS solution in the exact fashion as #IMCOUPLING. DoCouplePsPressure and DoCouplePsDensity select which values are nudged. DensityCoupleFloor controls the minimum density that results from this coupling. Setting this to a reasonable value helps prevent near-zero time steps.

The default action is to not couple density or pressure from PS.

#PWCOUPLING command

```
#PWCOUPLING
F                       DoLimitRhoPw
```

If DoLimitRhoPw is true, limit the PW supplied densities by the body densities from below.

Default is false.

3.8.16 Pic coupling

#PICUNIT command

```
#PICUNIT
1.0             xUnitPicSi [m]
3000e3          uUnitPicSi [m/s] Speed of light for PIC
```

Define the length and velocity units for the PIC model. The length unit is arbitrary (can be defined to be the same as for the MHD model, or any other convenient length). The velocity unit, however, determines the speed of light for the PIC model, since c=1 is defined. Using the true speed of light makes the convergence slow in the implicit solver of PIC. Therefore uUnitPicSi should be set to a velocity that is larger than the typical velocities (including the electron thermal velocity), but not orders of magnitude larger. For typical magnetosphere applications a few 1000 km/s can work.

Default is 1 for both parameters, which is only meaningful if the velocities are much smaller than 1 (e.g. in shock tube test problems).

#PICGRIDUNIT command

```
#PICGRIDUNIT
2               nPicGrid
1.0             xUnitPicSi [m]
3000e3          uUnitPicSi [m/s] Speed of light for the first PIC region
8               ScalingFactor
6400e3          xUnitPicSi [m]
1000e3          uUnitPicSi [m/s] Speed of light for the second PIC region
16              ScalingFactor
```

Similar to command #PICUNIT, but this command allows setting different normalization units for different PIC grids. ScalingFactor is used for changing the kinetic scales. See Toth et al. 2017 for more details.

If Hall MHD is used, the scaling factors in this command will not be used, and PIC boxes use the surrounding Hall factors as the scaling factor.

The defaults are the same as described in #PICUNIT. Do not use #PICUNIT and #PICRGRIDUNIT at the same time.

#PICGRID command

```
#PICGRID
2                       nPicGrid
  6.                    xMinPic
 10.                    xMaxPic
 -5.                    yMinPic (read for 2D or 3D only)
  5.                    yMaxPic (read for 2D or 3D only)
 -5.                    zMinPic (read for 3D only)
  5.                    zMaxPic (read for 3D only)
1/32                    DxPic
1/32                    DyPic   (read for 2D or 3D only)
1/32                    DzPic   (read for 3D only)
 10.                    xMinPic
 40.                    xMaxPic
-10.                    yMinPic (read for 2D or 3D only)
 10.                    yMaxPic (read for 2D or 3D only)
 -6.0                   zMinPic (read for 3D only)
  6.0                   zMaxPic (read for 3D only)
1/8                     DxPic
1/8                     DyPic   (read for 2D or 3D only)
1/8                     DzPic   (read for 3D only)
```

This command defines the number of PIC grids, their sizes and resolutions. All distances are given in the BATSRUS distance units. The grid resolution of the PIC grid can be different from the grid resolution of BATSRUS. When

coupling with FLEKS, the number of PIC grid cells in each direction should be a multiple of the patch sized defined by the #PICPATCH command.

The default is to have no PIC regions at all, so this command is required for the MHD-EPIC algorithm.

#PICADAPT command

```
#PICADAPT
T                       DoAdaptPic (rest is read if true)
100                     DnAdaptPic
-1.                     DtAdaptPic
```

This command controls the PIC adaptation functionality. This results in FLEKS covers a fixed region using PIC by the parameters set up in #PICGRID. If DoAdaptPic=.true., the PIC region will be recalculated based on the frequency (DnAdaptPic and DtAdaptPic) provided.

Default is DoAdaptPic false.

#PICPATCH command

```
#PICPATCH
4                        PatchSize
```

PatchSize is the minimum patch size of the adaptive PIC grid given as the number of cells in each direction, so a patch is a square in 2D and a cube in 3D. The PIC cells in a patch can be switched on and off together. The number of cells in the PIC grid, which is defined by #PICGRID, should be divisible by the PatchSize.

The smallest patch size is 2, and 4 is the default value. The smaller the patch size is, the smoother the PIC boundary will be. But the PIC code may become slower with smaller patch size. 4 or 8 are two typical values. 2 may slow down the code significantly.

The default value is 4.

#PICPATCHEXTEND command

```
#PICPATCHEXTEND
5                        NxExtend
5                        NyExtend
10                       NzExtend
```

This command contains the number of patches extended from the PIC region defined by #PICCRITERIA on different directions.

Default is no extension.

#PICBALANCE command

```
#PICBALANCE
T                      DoBalancePicBlock
F                      DoBalanceActivePicBlock
```

If DoBalancePicBlock is switched on, the BATSRUS blocks that are overlapped with the PIC domains (defined by #PICGRID) will be load balanced separatelly.

If DoBalanceActivePicBlock is true, only the BATSRUS blocks that are overlapped with the active PIC regions, which are determined by #PICREGIONMIN and/or #PICREGIONMAX, are load balanced separatelly. If the GM-PC coupling is slow, this option is likely speed up the coupling significantly. However, it has two known minor side effects, which may change nightly test results but are acceptable for a production run: 1. After calling load balance function, the PIC adaptation criteria will be re-calculated. For a simulation with physics based PIC region criteria,

running with 1 MPI or a few MPIs may produce different results, because the simulation with 1 MPI does not need to do load balance and the pic criteria will not be re-calculated. Load balancing BATSRUS blocks frequently may also slow down BATSRUS. 2. Inside BATS_advance(), set_global_timestep(TimeSimulationLimit) is called, and the timestep is limited by TimeSimulationLimit. However, if load_balance_blocks() is called later, the global time step will be overwritten inside calc_other_vars().

The default is DoBalancePicBlock true, and DoBalanceActivePicBlock is false.

#PICREGIONMIN command

```
#PICREGIONMIN
+daysidefixed -nearbody
```

The #PICREGIONMIN command sets the regions in the PIC grids that are always active (on). The region strings are defined by the #REGION command.

By default there is no minimal PIC region, so any part of the PIC domain can be switched off.

#PICREGIONMAX command

```
#PICREGIONMAX
+tailsidelarge
```

This command defines the maximum region the PIC can cover cover. PIC patches outside this region cannot become active. The region strings are defined by the #REGION commands.

By default the whole domain covered by PIC grids can become active.

#PICCRITERIA command

```
#PICCRITERIA
4                       nPicCriteria
j/b                     StringPicCriteria
0.1                     MinCriteriaValue
999.0                   MaxCriteriaValue
10.0                    CriteriaB1
j/bperp                 StringPicCriteria
0.8                     MinCriteriaValue
999.0                   MaxCriteriaValue
divcurv                 StringPicCriteria
-0.1                    MinCriteriaValue
999.0                   MaxCriteriaValue
entropy                 StringPicCriteria
0.02                    MinCriteriaValue
999.0                   MaxCriteriaValue
```

This command defines the physical criteria for selecting the PIC region. The first input is the number of criteria. For each criterion, there are three inputs: the name, minimum value and maximum value. The cell which satisfies all criteria will be active. This physics based selection can be limited geometrically by the #PICREGIONMIN and #PICREGIONMAX commands.

The available criteria are ”rho” (for testing), ”beta” (the ratio between plasma pressure and magnetic paressure), ”j/b”, ”j/bperp” (current divided by magnetic field for finding current sheets), ”divcurv” (the divergence of the curvature of the magnetic field lines to distinguish X lines from flux ropes), ”speed” (bulk flow speed to exclude magnetosheath), and ”jy” (distinguish main current sheet in magnetotail). Criteria ”j/b” and ”j/bperp” require an extra input parameter for B1 in the denominator to avoid dividing by 0. Default value of B1 is 1 nT.

By default the whole PIC region is active (possibly limited by #PICREGIONMAX).

3.8.17 Physics parameters

#GAMMA command

```
#GAMMA
4/3                     Gamma for fluid 1
1.4                     Gamma for fluid 2
5/3                     GammaElectron (if UseElectronPressure)
```

The adiabatic index gamma = c_p/c_v (ratio of the specific heats for fixed pressure and fixed volume). The gamma values have to be given for each fluid and also for the electrons if there is a Pe_ variable in the equation module.

Default is 5/3 for all the gamma-s.

#PLASMA command

```
#PLASMA
1.0             FluidMass [amu] H+
1.0             IonCharge [e]   H+
0.5             ElectronTemperatureRatio
```

For single fluid, single species MHD the FluidMass parameter determines the average mass of ions (and strongly coupled neutrals) in atomic mass units (amu). The number density is n=rho/FluidMass. For a pure hydrogen plasma FluidMass=1.0, while for a mix of 90 per cent hydrogen and 10 per cent helium FluidMass=1.4.

The IonCharge parameter gives the average ion charge in units of the proton charge. For a fully ionized hydrogen plasma AverageIonCharge=1.0, for a fully ionized helium plasma IonCharge=2.0, while for a 10 per cent ionized hydrogene plasma IonCharge=0.1.

For multifluid/multispcies MHD/HD the command reads the mass of all fluids/species (ions and neutrals), and the charges of all ion fluids/species. For example for proton and double ionized helium and neutral oxygen molecule fluids:

```
#PLASMA
1.0             FluidMass H+ [amu]
4.0             FluidMass He++ [amu]
32.0            FluidMass O2 [amu]
1.0             IonCharge H+ [e]
2.0             IonCharge He++ [e]
0.2             ElectronTemperatureRatio
```

The ElectronTemperatureRatio determines the ratio of electron and ion temperatures. The ion temperature Te = T * ElectronTemperatureRatio where T is the ion temperature. The total pressure p = n*k*T + ne*k*Te, so T = p/(n*k+ne*k*ElectronTemperatureRatio). If the electrons and ions are in temperature equilibrium, ElectronTemperatureRatio=1.0. For multi-fluid MHD the ElectronTemperatureRatio is interpreted as electron pressure ratio. The electron pressure is taken as pE = ElectronTemperatureRatio*sum(pIon_I). Note that one can also solve the electron pressure equation if 'Pe' is present in the equation module.

Multispecies MHD reads the mass and charge for all species in the same manner as multifluid. But the ion charge is still assumed to be 1 in the code and the values read in will not be used so far. ElectronTemperatureRatio is interpreted as the single fluid case.

In a real plasma all these values can vary in space and time, but in a single fluid/species MHD description using these constants is the best one can do. In multispecies MHD the number density can be determined accurately as n = sum(RhoSpecies_V/(ProtonMass*MassSpecies_V)).

The default ion/molecular masses are given in the equation module. The default ion charges are always 1. The default electron temperature ratio is zero, i.e. the electron pressure is assumed to be negligible relative to the (total) ion pressure, however in the solar wind boundary the electron pressure is set to equal to the first ion pressure.

This default is backwards compatible with previous versions of the code.

#LOOKUPTABLE command

```
#LOOKUPTABLE
p(rho,e)                          NameTable
use param                         NameCommand (use, load, save, make, param)
table1.out                        NameFile (read this if use/load/save)
real4                             TypeFile (read this unless "param")
zXe zBe nPl                       NameTableParam (if NameCommand has "param")
54.0                              TableParam number of protons in Xenon
4.0                               TableParam number of protons in Beryllium
4.0                               TableParam number of elements in plastic
p(rho,e) for ionized plasma       Description (read this and rest unless "load")
logrho logp pXe pBe pPl           NameVar
2                                 nIndex
100                               nIndex1
1e-6                              Index1Min
1e+6                              Index1Max
50                                nIndex2
0.001                             Index2Min
100.0                             Index2Max
```

Lookup tables allow interpolating one or more variables from a discrete table. For sake of efficiency, lookup tables should have uniform indexes, but non-uniform tables are also supported. Tables with up to 5 indexes are supported. Lookup tables are in the same format as structured ”IDL” plotfiles. The file format is described at the beginning of the source code share/Library/src/ModPlotFile.f90.

Tables are identified by the NameTable string that should be unique for the table and must agree with the name used in the ModUser module. The NameCommand tells if we should ”load” the table from a file, ”make” the table using some algorithm defined in the ModUser module, or make the table and then ”save” it into a file. The ”use” option is the same as ”load” if the table file already exists, otherwise it is the same as ”save”.

If NameCommand contains ”param” and the table is not loaded from a file, then the NameTableParam variable and the TableParam values of table parameters are read from the input file and stored into the table.

The file name and file type (”real4”, ”real8”, ”ascii”, ”log” or ”sat”) of the table are read when NameCommand contains ”load”, ”save”, or ”use”. The TypeFile is also read for ”make”, because setting it to ”real4” implies the use of single precision storage internally as well. This saves a factor of two in storage (both disk and memory), which may be very significant for large lookup tables. In fact, TypeFile=real4 is the recommended setting.

The ”log” or ”sat” file type corresponds to a one dimensional lookup table where the coordinate is usually time. The time can be given by up to 7 integer columns (year, month, day, hour, sec, min, msec). The integer time description is converted into a double precision time (number of seconds since 01/01/1965) which is the standard representation of time in the SWMF. These columns are identified by the space separated variable names that are just before the ”#START” string. Standard variable names indicating the date-time information are ”year” or ”yr”, ”month” or ”mo”, ”day” or ”dy”, ”min” or ”mn”, ”sec” or ”sc” and ”msec” or ”msc” in this order. Alternatively the variable name can be ”dateN” where N = 2...7 is the number of integers describing the date and time. The actual data follows the line containing the ”#START” string.

The rest of the parameters are read for commands ”make”, ”save” or ”use”. The NameVar string contains the space separated list of the names of the indexes and the one or more returned value(s). If the index name starts with a ”log”, a logarithmic index is assumed (ie. the table will be uniform in the logarithm of the index value). The nIndex parameter defines the number of indexes (dimensionality) of the table. The nIndex1 parameter defines the number of discrete values the first lookup index, and Index1Min and Index1Max are the smallest and largest values for the first index, respectively. For nIndex larger than 1, the nIndex2, Index2Min, Index2Max parameters define the number and range of the second index, etc.

This command can occur multiple times. By default no lookup tables are used.

#ADVECTION command

```
#ADVECTION
T                       UseAdvectionSource
Rho                     NameVarAdvectFirst
Rho                     NameVarAdvectLast
```

If UseAdvectionSource is true, then add a source term Var*div(u) for all variables from NameVarAdvectFirst to NameVarAdvectLast. This could be improved to a string of variables.

No advection source is added by default.

#FRICTION command

```
#FRICTION
0.2                     FrictionSi [1/s] (rest read if larger than 0)
0.5                     FrictionUxDim
0.0                     FrictionUyDim
0.0                     FrictionUzDim
```

Define a friction force against a background fluid moving at velocity FrictionU = (FrictionUxDim, FrictionUyDim, FrictionUzDim). The force is F = Friction*Rho*(FrictionU - U), where Friction is in normalized units (1/time), Rho and U are the density and velocity vector of the first fluid, respectively. The current implementation is for the first fluid with an explicit source term.

By default there is no friction.

#GRAVITY command

```
#GRAVITY
T                        UseGravity (rest of parameters read if true)
3                        iDirGravity(0 - central, 1 - X, 2 - Y, 3 - Z direction)
10.0                     GravitySi [m/s^2] (read if iDirGravity is not 0)
```

If UseGravity is false, the gravitational force of the central body is neglected. If UseGravity is true and iDirGravity is 0, the gravity points towards the origin and the gravitational force is determined by the mass of the central body. If iDirGravity is 1, 2 or 3, the gravitational force is parallel with the X, Y or Z axes, respectively, and the gravitational acceleration is given by the GravitySi parameter.

Default values depend on problem_type.

When a second body is used the gravity direction for the second body is independent of the GravityDir value. Gravity due to the second body is radially inward toward the second body.

#ARTIFICIALVISCOSITY command

```
#ARTIFICIALVISCOSITY
T               UseArtificialViscosity
0.3             AlphaVisco
0.3             BetaVisco
```

This command adds artificial viscosity (diffusion) to the density, moments and pressure equations based on the section 2.5.2 of the paper by P. McCorquodale and P. Colella (2010). The larger/smaller AlphaVisco/BetaVisco is the larger the artificial viscosity will be. AlphaVisco should be non-negative and BetaVisco should be positive. The recommended values are shown above.

Default is no artificial viscosity.

#VISCOSITY command

```
#VISCOSITY
T               UseViscosity
0.01            ViscosityCoeffSi [m2/s] (read if UseViscosity is true)
```

If UseViscosity is true, apply Navier-Stokes type viscosity using the viscosity coefficient ViscoCoeffSi.

Default is no viscosity.

#VISCOSITYREGION command

```
#VISCOSITYREGION
+magnetotail -nearbody          StringViscoRegion
```

This command is only useful if viscosity is switched on with the #VISCOSITY command.

The StringViscoRegion string can specify the region(s) where viscosity is used. The regions must be described with the #REGION commands. Note the 'tapered' option in the shape desciption that can be used to make the transition smoother.

The default is to apply viscosity everywhere if the it is switched on.

#RESISTIVITY command

```
#RESISTIVITY
T               UseResistivity (rest of parameters read only if set to true)
anomalous       TypeResistivity
1.0E+9          Eta0Si       [m2/s] (read except for Spitzer resistivity)
2.0E+9          Eta0AnomSi   [m2/s] (read for anomalous resistivity only)
2.0E+10         EtaMaxAnomSi [m2/s] (read for anomalous resistivity only)
1.0E-9          jCritAnomSi  [A/m2] (read for anomalous resistivity only)
```

The true SI units of resistivity are Ohm m = $Nm^2/(A^2 s)$. In BATSRUS, however, we use "normalized" units, so that the magnetic permeability $[N/A^2]$ disappers from the equations. So what is described here as "resistivity", is really eta/mu_0 which has units of $[m^2/s]$, same as (magnetic) diffusion. Since the normalized current is defined as curl B (instead of curl B/mu0), the electric field is E = -u x B + eta * J in the normalized units.

If UseResistitivy is false, no resistivity is included. If UseResistivity is true, then one can select a constant resistivity, the classical Spitzer resistivity, anomalous resistivity with a critical current, or a user defined resistivity.

For TypeResistivity='Spitzer' the resistivity is very low in space plasma. The only parameter read is the CoulombLogarithm parameter with typical values in the range of 10 to 30. Note that this can also be set with the #COULOMBLOG command.

For TypeResistivity='constant' the resistivity is uniformly set to the parameter Eta0Si.

For TypeResistivity='anomalous' the anomalous resistivity is Eta0Si + Eta0AnomSi*(j/jCritAnomSi-1) limited by 0 and EtaMaxAnomSi. Here j is the absolute value of the current density in SI units. See the example for the order of the parameters.

For TypeResistivity='user' only the Eta0Si parameter is read and it can be used to scale the resistivity set in subroutine user_set_resistivity in the ModUser module. Other parameters should be read with subroutine user_read_inputs of the ModUser file.

The default is UseResistivity=.false.

#COULOMBLOG command

```
#COULOMBLOG
20.0                    CoulombLog
```

Set the Coulomb logarithm for Spitzer resistivity and heat conduction. Default value is shown.

#RESISTIVITYOPTIONS command

```
#RESISTIVITYOPTIONS
T                       UseResistiveFlux
T                       UseJouleHeating
F                       UseHeatExchange
```

Switch off negligible resistivity effects for sake of computational speed. If UseResistiveFlux is false, the resistive terms in the induction equation are neglected. If UseJouleHeating is false and non-conservative equations are used then the Joule heating is neglected in the electron/ion pressure equation. If UseHeatExchange is false, the heat exchange between electron and ion pressures is neglected.

The defaults are true for all three logicals.

#RESISTIVEREGION command

```
#RESISTIVEREGION
+magnetotail -nearbody          StringResistRegion
```

This command is only useful is the resistivity is switched on with the #RESISTIVITY command.

The StringResistRegion string can specify the region(s) where resistivity is used. The regions must be described with the #REGION commands. Note the 'tapered' option in the shape desciption that can be used to make the transition smoother.

The default is to apply the resistive MHD scheme everywhere if it is switched on.

#HALLRESISTIVITY command

```
#HALLRESISTIVITY
T               UseHallResist
1.0             HallFactorMax
0.1             HallCmaxFactor
```

If UseHallResist is true the Hall resistivity is used. All parameters are read even if it is false to allow setting the kinetic scaling equal to HallFactorMax for MHD-EPIC, although it is better to use the #PICGRIDUNIT command for this purpose.

The off-diagonal Hall elements of the resistivity tensor are multiplied by HallFactorMax. If HallFactorMax is 1 then the physical Hall resistivity is used (but also see the #HALLREGION command). Note that a physically consistent way of changing the strength of the Hall effect is changing the ion mass and/or charge with the #PLASMA command.

If HallCmaxFactor is 1.0, the maximum propagation speed takes into account the full whistler wave speed. If it is 0, the wave speed is not modified. For values betwen 1 and 0 a fraction of the whistler wave speed is added. The full speed is needed for the stability of the one or two-stage explicit scheme (unless the whistler speed is very small and/or the diagonal part of the resistivity tensor is dominant). For 3 and 4-stage explicit schemes (see the #RK command) and also for the semi-implicit and implicit time stepping the HallCmaxFactor can be reduced, possibly all the way to zero to minimize the discretization errors. If the (semi-)implicit scheme does not converge well, using HallCmaxFactor larger than zero (for example 0.1) can help.

Default is UseHallResist false.

#HALLREGION command

```
#HALLREGION
+magnetotail -nearbody          StringHallRegion
```

This command is only useful if the Hall MHD scheme is switched on with the #HALLRESISTIVITY command.

The StringHallRegion string can specify the region(s) where the Hall resistivity is used. The regions must be described with the #REGION commands. Note the 'tapered' option in the shape desciption that can be used to make the transition smoother.

Each region has its own Hall factor, which is the 'Weight' associated with the #REGION command. If 'Weight' is not read in, then HallFactorMax, which is set by #HALLRESISTIVITY, is used as the default.

The default is to apply the Hall MHD scheme everywhere if it is switched on.

#BIERMANNBATTERY command

```
#BIERMANNBATTERY
T               UseBiermannBattery
```

If UseBiermannBattery is true then the Biermann battery term in the generalized Ohm's law is used, otherwise it is switched off.

If the Hall term is used in combination with the electron pressure equation then the Biermann battery term is switched on by default. In that case the BIERMANNBATTERY command is not needed.

Default is UseBiermannBattery false.

#MINIMUMDENSITY command

```
#MINIMUMDENSITY
0.001           RhoMinDim for fluid 1
-1.0            RhoMinDim for fluid 2
```

Provide minimum density(s) for the ion/neutral fluid(s). If the minimum density is positive, the density is kept above this limit for that fluid. The minimum density is given in the input/output units for density, which varies from application to application. A negative value indicates that no minimum density is applied for that fluid.

By default no minimum density limit is applied.

#MINIMUMPRESSURE command

```
#MINIMUMPRESSURE
0.001           pMinDim for fluid 1
-1.0            pMinDim for fluid 2
0.002           PeMinDim for electron pressure (if used)
```

Provide minimum pressure(s) for the ion/neutral fluid(s) and electrons. If the pMinDim is positive, the pressure is kept above this limit for that fluid. The minimum pressure is given in the input/output units for pressure, which varies from application to application. A negative value indicates that no minimum density is applied for that fluid.

By default no minimum pressure limit is applied.

#MINIMUMTEMPERATURE command

```
#MINIMUMTEMPERATURE
5e4             TminDim for fluid 1
-1.0            TminDim for fluid 2
2e4             TeMinDim for electron pressure (if used)
```

Provide minimum temperature(s) for the ion/neutral fluid(s) and electrons. If the minimum temperature (TMinDim) is positive, the temperature is kept above this limit. The minimum temperature is given in Kelvin. A negative value indicates that no minimum temperature is applied for that fluid.

By default no minimum temperature limit is applied.

#MINIMUMRADIALSPEED command

```
#MINIMUMRADIALSPEED
T                UseSpeedMin
10               rSpeedMin
250              SpeedMinDim
10 h             TauSpeedMinDim
```

If UseSpeedMin is true, the minimum speed is enforced. If the radial speed falls below SpeedMin beyond the radial distance rSpeedMin, then a force is applied (via a source term) to push the solar wind speed above SpeedMin with a time rate TauSpeedMinDim.

By default no minimum speed limit is applied.

#ELECTRONPRESSURE command

```
#ELECTRONPRESSURE
1.1e5                    PeMinSi
```

Provide the minimum electron pressure threshold in SI units. Currently the minimum electron pressure is only used in ModRadDiffusion. The default value is -1, i.e. no threshold is applied.

#ELECTRONENTROPY command

```
#ELECTRONENTROPY
T                        UseElectronEntropy
F                        UseElectronEnergy
```

If UseElectronEntropy is true, solve for the electron entropy Se defined as Se = Pe**(1/GammaE). The electron entropy, unlike electron pressure, satisfies a pure conservation law, so it is well behaved across shocks.

If UseElectronEnergy is true, include electron energy into the total energy equation for the conservative scheme. This is optional, when UseElectronEntropy is false, but required for UseElectronEntropy true if total energy is to be conserved. This feature is still under testing.

Explicit electron heatconduction is not implemented for the electron entropy, but the semi-implicit heat conduction should work fine.

The default values are shown above.

#ENTROPY command

```
#ENTROPY
T                        UseEntropy
T                        UseTotalIonEnergy (only read for multi-ion equations)
```

If UseEntropy is true and the ion pressure is isotropic, then solve for the ion entropy density s defined as s = P**(1/Gamma). The ion entropy, unlike ion pressure, satisfies a pure conservation law, so it is well behaved across shocks. If the ion pressure is anisotropic, then solve for Sperp = Pperp/B and Spar = Ppar*(B/rho)**2.

For multiion case the UseTotalIonEnergy parameter is read. If it is true, the total ion energy equation is solved, which is in conservation form. This is not yet implemented.

The default value of UseEntropy and UseTotalIonEnergy are false, except when the #SHOCKHEATING command is used, which requires and sets UseEntropy=T and UseTotalIonEnergy=T in the multiion case.

#SHOCKHEATING command

```
#SHOCKHEATING
0.5       PeShockHeatingFraction (read if electron pressure is used)
1.0       PparShockHeatingFraction (read if ion pressure is anisotropic)
-0.4      PiShockHeatingFraction for ion fluid 2 (if exists)
-0.1      PiShockHeatingFraction for ion fluid 3 (if exists)
```

If electron pressure is solved for and PeShockHeatingFraction is set to a positive value, this fraction of the non-adiabatic heating is deposited into the electron thermal energy Pe/(GammaE-1) between the electrons and the first ion fluid. For a negative value the weights are multiplied by the number densities to the (2-gamma) power.

If anisotropic ion pressure is solved for and PparShockHeatingFraction is set to a positive value, then a fraction of the non-adiabatic heating is deposited into the parallel ion energy, and the same amount of energy is removed from the perpendicular ion energy. If PparShockHeatingFraction is negative, then the fraction of heating going into the parallel pressure is abs(b.u), where b and u are the unit vectors for the magnetic field and velocity, respectively.

If multiple ion fluids are solved for and PiShockHeatingFraction is set to a positive value for ion fluid iFluid (2...nIonFluid), then this fraction of the non-adiabatic heating is deposited into fluid iFluid between the first and iFluid-th ion fluids. If PiShockHeatingFraction is set to a negative value, then the fluid weights are multiplied with the number densities to the (2-gamma) power, which results in correct heating of two fluids with equal ion masses, velocities and temperatures but different number densities.

If this command is used, UseEntropy, UseTotalIonEnergy and UseElectronEntropy are set to true (see #ENTROPY and #ELECTRONENTROPY).

Note that the fractions describe ratio of entropies, not energies. The scheme conserves the total energy and the linear combination of entropies Weight2*Entropy1 - Weight1*Entropy2, where Weight1 and Weight2=1-Weight1 are obtained from the various fractions defined by the command, while Entropy1 and Entropy2 are the volumetric entropy densities, for example p/rho^{γ} for the isotropic case. For isotropic pressure and the same gamma for ions and electrons the entropies and thermal energy densities $p/(\gamma - 1)$ are closely related. For anisotropic ion pressure the entropy densities depend on the magnetic field too, which makes things complicated.

Default values are 0, which means that all shock heating goes to the (perpendicular) ion pressure. For the multiion case, total energy is only conserved if UseTotalIonEnergy is set to true (either here or in the #ENTROPY command), otherwise the magnetic energy is ignored. Solving for individual ion energies will transfer some shock heating into both ion fluids, but it may not be the correct distribution. This can be improved by using this command and prescribing the energy distribution.

#ANISOTROPICPRESSURE command

```
#ANISOTROPICPRESSURE
T                       UseConstantTau fluid 1
10                      TauInstabilitySi
100                     TauGlobalSi
T                       UseConstantTau fluid 2
10                      TauInstabilitySi
100                     TauGlobalSi
```

Set parameters for the pressure relaxation term for each fluid. Note that in the previous version, TauInstabilitySi will only be read if UseConstantTau is true. However, this version TauInstabilitySi will be read even UseConstantTau is false.

If UseConstantTau is set to false, use the growth-rate based relaxation time. This is the default for single ion fluid and also recommended.

If UseConstantTau is set to true (default for multiple ion fluids), then TauInstabilitySi provides the exponential relaxation time in seconds to restrict the pressure anisotropy in unstable regions. Within the time, the parallel pressure

is pushed towards plasma instability limits. The default value is -1, i.e, do not apply the pressure relaxation due to instabilities. If applied, a typical value for magnetospheric simulations is 10 seconds.

TauGlobalSi provides the global pressure exponential relaxation time in seconds applied in the whole domain. Within the time, the parallel pressure is pushed towards the total scalar pressure. In the presence of both the instability and global relaxation, the one that changes pressure more will be used for the pressure relaxation term. The default value for TauGlobalSi is -1, i.e. do not apply the global relaxation. The example shows a recommended value for magnetospheric simulations.

When UseConstantTau = T and TauInstabilitySi = -1, the pressure relaxation term is not applied, thus TauGlobalSi is meaningless in this case.

#EXTRAINTERNALENERGY command

```
#EXTRAINTERNALENERGY
-1e3                    ExtraEintMinSi
```

Provide the minimum extra internal energy density threshold in SI units. The extra internal energy density is the difference between true internal energy density and the p/(gamma-1) of the ideal gas. Using a large enough gamma (e.g. 5/3) can guarantee that the difference is always non-negative. The default value is zero.

#RADIATION command

```
#RADIATION
T               UseRadDiffusion    (rest of parameters read only if true)
T               UseRadFluxLimiter
larsen          TypeRadFluxLimiter (read only if UseRadFluxLimiter is true)
300.0           TradMinSi
```

If UseRadDiffusion is true the radiation hydrodynamics with radiation nonequilibrium diffusion approximation is used.

If the UseRadDiffusion is set to true, then optionally a non-linear flux limiter can be invoked via UseRadFluxLimiter set to true. This limits the radiation diffusion flux so that it does not exceed the optically thin streaming limit, the speed of light. The type of flux limiter can be selected by setting TypeRadFluxLimiter.

If TypeRadFluxLimiter=”sum”, then Wilson’s sum flux limiter is used. If TypeRadFluxLimiter=”max”, then Wilson’s max flux limiter is used. For TypeRadFluxLimiter=”larsen” the square-root flux limiter of Larsen is used.

The TradMinSi parameter sets a minimum temperature in Kelvins for the radiation. This helps avoiding negative radiation temperature due to numerical errors. A recommended value is 300K.

The default for UseRadFluxLimiter is false.

#HEATFLUXLIMITER command

```
#HEATFLUXLIMITER
T                       UseHeatFluxLimiter
0.06                    HeatFluxLimiter
```

If UseHeatFluxLimiter is set to false, the original Spitzer-Harm formulation for the collisional isotropic electron thermal heat conduction is used as set by the #SEMIIMPLICIT command.

If UseHeatFluxLimiter is set to true, this isotropic heat conduction is modified to correct the heat conduction coefficient if the electron temperature length scale is only a few collisonal mean free paths of the electrons or smaller. The flux limited heat conduction that is used in this case is the threshold model.

If we define the free streaming flux as F_fs = n_e*k_B*T_e*v_th, where v_th = sqrt(k_B*T_e/m_e) is a characteristic thermal velocity, then the threshold model limits the heat conduction flux F = -kappa*grad(Te), with heat conduction coefficient kappa, by F = -min(kappa, f*F_fs / —grad(Te)—) * grad(Te) Here, f is the heat flux limiter.

A possible application of interest for the heat flux limiter is laser-irradiated plasmas. For this limiter to work properly, the thermodynamic quanties in the user_material_properties subroutine in the ModUser module need to be defined (see ModUserCrash for an example).

The default for UseHeatFluxLimiter is false.

#LASERPULSE command

```
#LASERPULSE
T               UseLaserHeating (rest of parameters are read if true)
3.8e10          IrradianceSI [J/s]
1.0e-10         tPulse [s]
1.0e-11         tRaise [s]
1.0e-11         tDecay [s]
```

This command is used for CRASH applications and it requires a CRASH related user file.

Read parameters for the laser pulse. The irradiance determines the energy per second. The length, rise, and decay times are given by the other three parameters. The laser heating is switched off by default.

#LASERBEAMS command

```
#LASERBEAMS
rz              TypeBeam
30              nRayPerBeam
438.0           rBeam
-290.0          xBeam
```

This command is used for CRASH applications and it requires a CRASH related user file. This command should be used together with the #LASERPULSE command.

The TypeBeam determines the geometry of the beams. Currently all beam definition are only available for rz-geoemrty.

For TypeBeam=rz, each beam consists of 2*nRayPerBeam+1 rays. The rays are parallel and are up to 1.5 rBeam away from the central ray. The xBeam determines the starting X position of the rays.

For TypeBeam=3d in rz-geometry there is the option for a beam definition on a polar or cartesian grid (The grid is defined orthogonal to the initial ray propagation direction). On a polar grid the rays locations are defined on a uniform grid with nRayR rays in the radial direction from 0 to 1.5*rBeam and nRayPhi+1 rays in the angle direction from 0 to pi. Due to symmetry properties in the laser beams the angle from pi to 2*pi are not needed. On a cartesian grid the ray locations are defined on a 2*nRayY+1 by nRayZ+1 uniform grid. The y-direction ranges from -1.5*rBeam to 1.5*rBeam. Due to symmetry in each beam the z-direction is limited between 0 and 1.5*rBeam.

#LASERBEAM command

```
#LASERBEAM
10.0            SlopeDeg
0.0             yBeam
1.0             AmplitudeRel
```

This command is used for CRASH applications and it requires a CRASH related user file. This command should be used together with the #LASERPULSE command.

The SlopeDeg parameter determines the direction of the beam relative to the X axis. The yBeam has to do with the Y coordinate of the initial positions. The AmplitudeRel gives the relative intensity of the beam.

#LASERBEAMPROFILE command

```
#LASERBEAMPROFILE
4.2             SuperGaussianOrder
```

This command is used for CRASH applications and it requires a CRASH related user file. This command should be used together with the #LASERPULSE command.

The SuperGaussianOrder parameter determines the profile of each laser beam. The irradiance profile of the beam is of the form exp[- (r / rBeam)**SuperGaussianOrder], where r is the distance to the tilted central ray of the beam and rBeam is defined by the #LASERBEAMS command. The default value for SuperGaussianOrder is 4.2

#MASSLOADING command

```
#MASSLOADING
F                       UseMassLoading
F                       DoAccelerateMassLoading
```

#HEATCONDUCTION command

```
#HEATCONDUCTION
T                       UseHeatConduction
spitzer                 TypeHeatConduction
```

If UseHeatConduction is false, no heat conduction is included. If UseHeatConduction is true, then one can select the collisional heat conduction of Spitzer or a user defined heat conduction. Both heat conduction formulations are field-aligned and are only applied to the electrons.

For TypeHeatConduction='spitzer' a spatially uniform Coulomb logarithm of 20 is assumed by default, resulting in a heat conduction coefficient of

```
9.2e-12 W m^-1 K^-7/2.
```

Fully ionized hydrogen plasma is assumed. The Coulomb logarithm can be modified with the #COULOMBLOG command.

For TypeHeatConduction='user' the heat conduction coefficient of the field-aligned heat conduction is read from the user_material_properties subroutine in the ModUser module. Optional parameters should be read with subroutine user_read_inputs of the ModUser file.

The default is UseHeatConduction=.false.

#IONHEATCONDUCTION command

```
#IONHEATCONDUCTION
T                       UseIonHeatConduction
spitzer                 TypeIonHeatConduction
```

If UseIonHeatConduction is false, no proton heat conduction is included. If UseIonHeatConduction is true, then one can select the classical Coulomb-mediated ion heat conduction or a user defined heat conduction. Both heat conduction formulations are field-aligned and are only applied to the protons.

For TypeIonHeatConduction='spitzer' a spatially uniform Coulomb logarithm of 20 is assumed by default, resulting in a heat conduction coefficient of

```
2.6e-13 W m^-1 K^-7/2
```

for protons. A non-default value can be set with the #COULOMBLOG command.

For TypeIonHeatConduction='user' the heat conduction coefficient of the field-aligned heat conduction is read from the user_material_properties subroutine in the ModUser module. Optional parameters should be read with subroutine user_read_inputs of the ModUser file.

The default is UseIonHeatConduction=.false.

#HEATFLUXREGION command

```
#HEATFLUXREGION
T                       UseHeatFluxRegion
5.0                     rCollisional
8.0                     rCollisionless
```

If UseHeatFluxRegion is false, the electron heat conduction (as set by the #HEATCONDUCTION command), is applied everywhere.

If UseHeatFluxRegion is true, the electron heat conduction is multiplied with a geometrical function depending on the sign of rCollisionless. If rCollisionless is smaller than zero, then the electron heat is multiplied by

$$f_S = \frac{1}{1 + (r/rCollisional)^2}.$$

If rCollisionless is positive, then the electron heat conduction coefficient is multiplied by

$$f_S = exp(-((r - rCollisional)/(rCollisionless - rCollisional)) * *2).$$

In both cases, if the #HEATFLUXCOLLISIONLESS command is set, then the polytropic index in the electron pressure equation is smoothly interpolated between γ in the collisional regime and γ_H in the collisionless regime:

$$\gamma_e = \gamma f_S + \gamma_H (1 - f_S),$$

where γ_H is defined in the #HEATFLUXCOLLISIONLESS command.

The default is UseHeatFluxRegion=.false.

#HEATFLUXCOLLISIONLESS command

```
#HEATFLUXCOLLISIONLESS
T                       UseHeatFluxCollisionless
1.05                    CollisionlessAlpha
```

If UseHeatFluxCollisionless is true, an empirical model is used to mimic the collisionless electron heat conduction (Hollweg, J.V., 1978). This empirical model reduces the polytropic index in the electron pressure equation to

$$\gamma_H = \frac{\gamma + \frac{3}{2}(\gamma - 1)\alpha}{1 + \frac{3}{2}(\gamma - 1)\alpha},$$

where $\gamma = 5/3$ and α is the input parameter CollisionlessAlpha. For the default value $\alpha = 1.05$, the polytropic index for the electron pressure equation is reduced to $\gamma_H \approx 1.33$. The collisionless heat flux only works if the equation module contains the state variable Ehot_. See van der Holst et al. 2014 for more details on this empirical model.

The default is UseHeatFluxCollisionless=.false.

#SECONDBODY command

```
#SECONDBODY
T                       UseBody2 ! Rest of the parameters read if true
1.0                     rBody2
0.                      MassBody2Si [kg] ! If 0, the second body gravity is 0
1.0                     Body2NDim [/cc] density for fixed BC for rho_BLK
1000.0                  Body2TDim [K] temperature for fixed BC for P_BLK
F                       UseBody2Orbit
1.                      xBody2  ! only read if UseBody2Orbit is false
0.                      yBody2  ! only read if UseBody2Orbit is false
0.                      zBody2  ! only read if UseBody2Orbit is false
```

Defines the radius, position, surface density and temperature, of a second body. The second body may also have magnetic field given by the #DIPOLEBODY2 command. This command should appear before the #INNERBOUNDARY command when using a second body. MassBody2Si is used to calculate the gravity force.

If UseBody2Orbit is .true., the orbit of the second body is traced using orbit elements set in CON_planet in the shared module, assuming that the central body is the Sun (or a star set in CON_star), so that the orbit elements are set in the HGI coordinate system. In this case, xBody2, yBody2, zBody2 are not read. Otherwise the position of the second body is defined by xBody2, yBody2, and zBody2.

Default is UseBody2 false.

#DIPOLEBODY2 command

```
#DIPOLEBODY2
0.0                     BdpDimBody2x [nT]
0.0                     BdpDimBody2y [nT]
-1000.0                 BdpDimBody2z [nT]
```

The BdpDimBody2x, BdpDimBody2y and BdpDimBody2z variables contain the 3 components of the dipole vector in the GSE frame. The absolute value of the dipole vector is the equatorial field strength in nano Tesla.

Default is no dipole field for the second body.

3.8.18 Corona specific commands

#FACTORB0 command

```
#FACTORB0
1e-4                    FactorB0
```

FactorB0 is a multiplication factor for the magnetogram based potential field B0. It can be used to correct the magnetic field units (default is Gauss) or to change the strength of the field.

Default value is 1.

#HARMONICSGRID command

```
#HARMONICSGRID
1.0                     rMagnetogram
2.5                     rSourceSurface
F                       IsLogRadius
30                      MaxOrder
30                      nR
72                      nLon
30                      nLat

#HARMONICSGRID
1.0                     rMagnetogram
25.0                    rSourceSurface
T                       IsLogRadius
180                     MaxOrder
400                     nR
180                     nLon
90                      nLat
```

This command determines the grid used in the B0 and B0New lookup tables generated from the spherical harmonics.

The radial grid goes from the inner boundary at rMagnetogram (typically 1) to the source surface radius rSourceSurface where B0 becomes radial. The longitude goes from 0 to 360 degrees, while the latitude from -90 to 90 degrees. Both angular coordinates are uniform (no sine latitude grid).

Traditionally rSourceSurface is 2.5, but this may not be the best choice. Setting rSourceSurface to 25.0 eliminates the non-zero curl of B0 inside the SC domain, so #CURLB0 command is not needed and numerical artifacts are minimized. In essense, B0 should capture the field near the active regions but it does not need to represent the helmet streamer or the heliospheric current sheet. Those features are best captured by the B1 field obtained from solving the MHD equations. When rSourceSurface is much larger than rMagnetogram, it is recommended to use a logarithmic radial grid with IsLogRadius set to true.

MaxOrder sets the maximum harmonics order used. This may get reduced to the order present in the harmonics files read by #HARMONICSFILE and #NEWHARMONICSFILE. If MaxOrder is less than the order present in the files, then the higher order harmonics are ignored.

nR, nLon and nLat give number of grid cells in the radial, longitudinal and latitudinal directions, respectively. The B0 field is stored on the grid (nR+1)*(nLon+1)*(nLat+1) grid nodes.

Default values are shown by the first example.

#HARMONICSFILE command

```
#HARMONICSFILE
Param/CORONA/CR1935_WSO.dat            NameHarmonicsFile
```

NameHarmonicsFile is the name of the file containing the harmonics coefficients.

After reading the harmonics file, the B0 lookup table is generated and saved. By default this lookup table is saved into ”harmonics_bxyz.out” file. The defaults can be changed with the #LOOKUPTABLE command. Once the lookup table file is created, it can be loaded directly and there is no need for this command.

The temporal evolution of the magnetogram can be captured by using an additional B0NEW lookup table. See also the #NEWHARMONICSFILE command.

By default there is no B0 lookup table.

#NEWHARMONICSFILE command

```
#NEWHARMONICSFILE
Param/CORONA/CR1936_WSO.dat    NameHarmonicsFileNew
```

NameHarmonicsFileNew is the name of the file containing the harmonics coefficients for the time at the end of the session.

After reading the harmonics file, the B0NEW lookup table is generated and saved into the ”harmonics_bxyz_new.out” file. The default parameters of the lookup table can be changed with the #LOOKUPTABLE command. Once the lookup table file is created, it can be loaded directly and there is no need for this command.

The potential field contained in the B0 and B0NEW lookup tables will be interpolated in time during the session.

By default there is no B0NEW lookup tables.

#MAGNETOGRAM command

```
#MAGNETOGRAM
T                       UseMagnetogram (rest of parameters read if true)
1.0                     rMagnetogram
2.5                     rSourceSurface
0.0                     HeightInnerBc (not used)
Param/CORONA/CR1935_WSO.dat    NameHarmonicsFile
```

This command is obsolete and has been replaced with the #HARMONICSFILE command.

If UseMagnetogram=T then read the harmonics file for the coronal magnetic field and use it to set B0 to the potential field solution.

rMagnetogram and rSourceSurface are the photosphere and source surface heliocentric radii, respectively. B0 becomes radial at rSourceSurface (typically taken to be 2.5 solar radii).

HeightInnerBc is the height above the photosphere of the boundary surface, non-zero values for this parameter are not recommended to unexperienced users.

NameHarmonicsFile is the name of the file containing the harmonics.

Default is UseMagnetogram=F.

#LDEM command

```
#LDEM
F                       UseLdem (rest of parameters read if true)
LDEM_moments.out        NameLdemFile
1                       iRadiusLdem
```

If UseLdem=T then read the LDEM moments file for the coronal density and temperature.

NameLdemFile is the name of the file containing the Ldem moments.

iRadiusLdem gives the index of the desired radius at which data is extracted. The Ldem moments data is ordered into concentric spherical shells of increasing radius, ranging from 1.035Rs to 1.255Rs, in increaments of 0.01Rs. The user can select the desired radius by varying the iRadiusLdem parameter. The minimal accepted value of iRadiusLdem is 1, corresponding to 1.035Rs. iRadiusLdem=2 corresponds to 1.045Rs, and so forth.

Default is UseLdem=F, iRadiusLdem=1

#EMPIRICALSW command

```
#EMPIRICALSW
WSA             NameModelSW
```

Depending on the expansion factors, calculated using the magnetogram field, for NameModelSW=WSA the spatial distribution of varied gamma is calculated. Through the Bernoulli integral the solar wind at 1AU should fit the WSA solar wind semi-empirical model, with the prescribed distribution of the varied gamma. Default value is NameModelSW=none.

#WSACOEFF command

```
#WSACOEFF
240.0           ConstantSpeed [km/s]
675.0           ModulationSpeed [km/s]
4.5             PowerIndex1
1.0             Coeff1
0.8             Coeff2
2.8             Angle [deg]
1.25            PowerIndex2
3.0             PowerIndex3
0.0             LowerBound
9999.0          UpperBound
```

Read in various parameters for the Wang-Sheely-Arge model. The exact meaning of the parameters should be obtained from publications on the WSA model. Default values are show.

#POYNTINGFLUX command

```
#POYNTINGFLUX
0.3E-6          PoyntingFluxPerBSi [J/m^2/s/T]
```

The boundary condition for the Alfven wave energy density is empirically set by prescribing the Poynting flux S_A of the outgoing waves. The wave energy density w (w_+ for positive radial magnetic field B_r and w_- for negative B_r) then follows from $S_A = V_A w \propto B_\odot$, where V_A is the Alfven speed, $B_\odot$ is the field strength at the inner boundary and the proportionality constant is estimated in Sokolov et al. (2013) as $(S_A/B)_\odot = 1.1 \times 10^6$ W m^{-2} T^{-1}. Under the assumption of sufficiently small returning flux, this estimate of the Poynting-flux-to-field ratio is equivalent to the following averaged velocity perturbation

$$(\delta \mathbf{u}_\perp \cdot \delta \mathbf{u}_\perp)^{1/2} \approx 15 \text{ km s}^{-1} \left(\frac{3 \cdot 10^{-11} \text{ kg m}^{-3}}{\rho} \right)^{1/4}, \tag{3.3}$$

where the mass density $3 \cdot 10^{-11}$ kg m^{-3} (ion number density $N_i = 2 \cdot 10^{16}$ m^{-3}) corresponds to the upper chromosphere. This value is compatible with the Hinode observations of the turbulent velocities of 15 km s^{-1}. Hence, the energy density of the outgoing wave is set to $w = (S_A/B)_\odot \sqrt{\mu_0 \rho}$.

Default value for PoyntingFluxPerBSi is 1.0E-6.

#CORONALHEATING command

```
#CORONALHEATING
exponential     TypeCoronalHeating
0.0575          DecayLengthEXP      [Rsun]         (read for exp heating only)
7.285E-05       HeatingAmplitudeCgs [ergs/cm^3/s] (read for exp heating only)

#CORONALHEATING
unsignedflux    TypeCoronalHeating
0.0575          DecayLength       [Rsun] (read for unsignedflux heating only)
1.0             HeatNormalization [none] (read for unsignedflux heating only)

#CORONALHEATING
alfvenwavedissipation   TypeCoronalHeating
7.5E4                   LperpTimesSqrtBSi (read for alfvenwavedissipation only)
0.04                    Crefl             (read for alfvenwavedissipation only)

#CORONALHEATING
turbulentcascade        TypeCoronalHeating
1.5e5                   LperpTimesSqrtBSi (read for turbulentcascade only)
0.0                     rMinWaveReflection
F                       UseReynoldsDecomposition
1.0                     KarmanTaylorBeta  (for UseReynoldsDecomposition only)

#CORONALHEATING
usmanov                 TypeCoronalHeating
T                       UseTransverseTurbulence (read for usmanov only)
-1/3                    SigmaD (read for usmanov only)
1.0                     KarmanTaylorAlpha (read for usmanov only)
0.5                     KarmanTaylorBeta2AlphaRatio (read for usmanov only)
```

If UseCoronalHeating is false, no CoronalHeating is included. If UseCoronalHeating is true, then one can select a simple exponential scale height heating model or B weighted heating model normalized to the amount of unsigned flux measured at the soalr surface (Abbett 2007). Each model applies a cell based source term to the Energy equation.

For TypeCoronalHeating='exponential' coronal heating is applied using an exponential scale height model. DecayLengthExp is the e-folding length in units of Solar Radii and HeatingAmplitudeCgs is the heating rate at r=1.0

For TypeCoronalHeating='unsignedflux' the coronal heating term is calculated using the unsigned flux model presented in (Abbett 2007). DecayLengthExp is the e-folding length in units of Solar Radii to limit the range of influence of this function. Because the total power in X-Ray emission is not well constrained to total heating power in the corona, the term HeatNormalization is used to uniformly multiply the heating rate by this factor (default 1.0).

For TypeCoronalHeating='NonWKB' coronal heating is applied using the wave dissapation model of Cranmer 2010. No additional input parameters are needed.

For TypeCoronalHeating='alfvenwavedissipation' coronal heating is applied using an anisotropic formulation of the Kolmogorov-type dissipation.

For TypeCoronalHeating='turbulentcascade', the Alfven wave energy density equations account for the partial reflection of Alfven waves due to Alfven speed gradients and field-aligned vorticity. The resulting counter propagating waves are responsible for the nonlinear turbulent cascade. The dissipation rate for the wave energy density, w_+, is controlled by the amplitude of the oppositely propagating wave, $|\mathbf{z}_-| = 2\sqrt{w_-/\rho}$, and is inversely proportional to the correlation length, $L_\perp$, in the transverse (with respect to the magnetic field) direction. Similar to Hollweg (1986) we use a simple scaling law $L_\perp \propto B^{-1/2}$ with the proportionality constant $L_\perp\sqrt{B}$ as input parameter LperpTimesSqrtBSi. off in the cells, at which R_BLK(i,j,k,iBlock) $<$ rMinWaveReflection. If UseReynoldsDecomposition is set true, there are two options, depending on how the code is configured. If the extra state variable, WDiff is set up, then switching on UseReynoldsDecomposition will result in solving three wave energy equations, for W_+, W_- and W_D, the latter being a difference between the turbulent kinetic and magnetic energy densities, with the intermode exchange ("reflection") coefficients are properly limited to avoid spurios oscillations in the numerical solution. Otherwise, if WDiff state variable is not introduced, however, UseReynoldsDecomposition is set to true, then the only element of the Reynolds-decomposed new model is employed, namely the limiter for the reflection coefficient, which, again, may be used to avoid spurious oscillations.

For TypeCoronalHeating='usmanov' (to be continued: I more or less know what does this mean, however, it is better to ask Bart to comment on this - may be not right now -IS).

The default is TypeCoronalHeating="none"

#LIMITIMBALANCE command

```
#LIMITIMBALANCE
2.0             MaxImbalance
```

This command allows the user to adjust (usually, reduce) the reflection of Alfven waves in the coronal hole, if the "turbulentcascade" type of coronal heating is applied. In brief, the reflected energy flux is limited by the 1/(MaxImbalance**2) fraction of the outgoing turbulent energy flux. If MaxImbalance=2 (default value), this agrees with the usual assumption that 80=1/(2**2) of the outward propagated turbulence) is reflected toward the Sun.

#LONGSCALEHEATING command

```
#LONGSCALEHEATING
T               DoChHeat (rest of parameters read only if set to true)
7.285E-05       HeatChCgs       [ergs/cm^3/s]
0.0575          DecayLengthCh   [Rsun]
```

If DoChHeat is false, no long scale height heating is included. If DoChHeat is true, one supplies parameters for a simple exponential scale height heating model like that in the CORONALHEATING command. HeatChCgs sets the base heating rate ar r=1.0 [Rsun] and DecayLengthCh is the e-folding length in units of Solar Radii. The idea is to use this commmand in conjunction with any short scale height heating model selected by the CORONALHEATING command.

The default is DoChHeat=.false.

#ACTIVEREGIONHEATING command

```
#ACTIVEREGIONHEATING
T               UseArComponent (rest of parameters read only if set to true)
4.03E-05        ArHeatFactorCgs [ergs/cm^3/s]
30.0            ArHeatB0        [Gauss]
5.0             DeltaArHeatB0   [Gauss]
```

If UseArComponent is false, no ActiveRegion heating component is used. If UseArComponent is true, one supplies parameters for a linear B weighted heating model used to supply strong heating to regions of high magnetic field strength. This model multiplies ArHeatFactorCgs by the cell magnetic field strength in gauss to determine a heating rate. ArHeatB0 is the central field strength for the tanh transition function that selects between the exponential heating model supplied by the CORONALHEATING command and the ArHeating term. DeltaArHeatB0 is the width of this transition function. This transition function has values: approx 0.1 at b0-deltab0, 0.5 at b0, and approx 0.9 at b0+deltab0.

This heating is ONLY applied when CORONALHEATING is set to the exponential heating model at the moment.

The default is UseArComponent=.false.

#OPENCLOSEDHEAT command

```
#OPENCLOSEDHEAT
T               DoOpenClosedField
```

If DoOpenClosedHeat=.true., then the heating function or the turbulent heating rate are modulated from closed to open magnetic field. Exponential heating function as well as the unsigned flux model function are switched off in the open field region. With the Cranmer heating function, the reflection coefficient in the closed field region is set to one, intensifying the heating.

Default is DoOpenClosedField = .false.

#NONLINAWDISSIPATION command

```
#NONLINAWDISSIPATION
T                       UseNonLinearAWDissipation
```

Intensifies the Alfven wave disssipation in the regions of weak field

#HEATPARTITIONING command

```
#HEATPARTITIONING
uniform                 TypeHeatPartitioning
0.6                     QionRatio
0.0                     QionParRatio (if used)

#HEATPARTITIONING
stochasticheating       TypeHeatPartitioning
0.21                    StochasticExponent
0.18                    StochasticAmplitude
```

If the #CORONALHEATING command is used in combination with more than one pressure state variable, then the heat partitioning is automatically called. The type of heat partitioning can be selected with the #HEATPARTITIONING command.

TypeHeatPartitioning='uniform' is the default. QionRatio is the fraction of the coronal heating that is used for the ion heating, while QionParRatio is the fraction of the coronal heating that is used for the parallel ion heating. The fraction of electron heating is 1.0-QionRatio.

If TypeHeatPartitioning='stochasticheating', then the heat partitioning follows a strategy based on the dissipation of kinetic Alfven waves. In particular we employ the stochastic heating mechanism for the perpendicular proton temperature (chandran, 2011). In this mechanism, the electric field fluctuations due to perpendicular turbulent cascade can disturb the proton gyro motion enough to give rise to perpendicular stochastic heating, assuming that the velocity perturbation at the proton gyro-radius scale is large enough. See van der Holst et al. (2014) for details of the StochasticExponent and StochasticAmplitude input parameters. The maximum possible StochasticExponent is 0.34 for randomly phased kinetic Alfven waves.

#CHROMOSPHERE command

```
#CHROMOSPHERE
F                       UseChromosphereHeating
2e11                    NeChromosphereCgs
5e4                     TeChromosphereSi
```

Set plasma parameters at top of chromosphere. If desired, the special heating function may be applied to maintain a hydrostatic density profile in the chromosphere at constant electron temperature TeChromosphereSi. May be used if the chromosphere region is included into a computational domain or to specify the boundary condition for the analytic emission model from the transition region.

#RADIATIVECOOLING command

```
#RADIATIVECOOLING
T                       UseRadCooling
```

Switches the radiation cooling on and off. For coronal solar plasma the emissivity calculated in the "coronal" approximation (optically thin plasma with no radiation-induced excitations and ionization). The radiation loss rate is approximated using CHIANTI tables or approximate interpolation formula (see comments in src/ModRadiativeCooling.f90). Default value for UseRadCooling is .false.

3.8.19 Threaded low solar corona

#FIELDLINETHREAD command

```
#FIELDLINETHREAD
T               UseFieldLineThreads
200             nPointThreadMax
0.002           DsThreadMin
```

If the logical, UseFieldLineThreads is set to .true., then, from center of physical cell near the inner boundary, the magnetic field line (tread) is traced toward photosphere, by integrating equation dx/ds = B/ —B— with a step, ds = DsThreadMin.

While integrated, the line is not allowed to turn back (outward the Sun). Except for this, no other means is used to help the line to reach the photosphere. If within nPointThreadMax steps the photosphere is not reached by any line, it is traced again, with the integration step being ds = 2*DsThreadMin now, and within this second and last (for the given line) integration, the angle is limited between the line and radial direction, so that the line, is guaranteed to reach the photosphere within nPointThreadMax in the course of the second tracing. The line shape is arbitralily distorted in this case, that is why the product, nPointThreadMax*DsThreadMin, which is the maximum length of the undistorted should be not too small: it should well exceed the straight line distance, D, from the physical cell center to the photosphere:

On the other hand, the physical length of "bad" lines, which may be as long as 2*DsThreadMin*nPointThreadMax, should not be too long, to prevent the heating instability, which cannot be balanced by heat conduction when the

boundaries are too far. With this regard, the right hand side of the above inequality provides both lower and, at the same time, upper estimate for the product in the left hand side.

The set of points on the line obtained in the course of integration form equally spaced grid on the thread (with the mesh egual to DsThreadMin for the most of threads, and twice this for the other) on which to solve the governing equations. The minimum number of gridpoints on thread is reported each time when the threads are generated. If this number is too small, the resolution and approximation are bad. Particularly, if the above settings are applied with the low boundary for SC grid at 1.05 Rs, then the distance from the physical cell center to the photosphere may be about 0.06, so that the grid point nmber, in principle, can be as low as 0.06/(2*0.02) = 15, which is evidently too small (nPointThreadMax = 150 and DsThreadMin = 0.001 are preferred).

#PLOTTHREADS command

```
#PLOTTHREADS
T       DoPlotThreads (read rest if true)
10      nGUniform
T       UseTRCorrection
F       UsePlanarTriangles
```

Used for plotting images in the solar corona. The threaded gap contributes to the line-of-site integrals determining the intensity of the image pixel, if DoPlotThreads is true. The threaded gap is split into nGUniform intervals uniformly in the first generalized coordinate.

If UseTRCorrection is true correct the contribution frop the threaded gap to the LOS plots is corrected to account for the contribution from transition region.

When triangulation on sphere as described above is completed, interpolation weights can be assigned in two ways: via the areas of spherical triangles (UsePlanarTriangle=F) or via areas of planar triangles (UsePlanarTriangle=T). The latter is the ”original” interpolation algorithm by Renka, who proved its good theretical properties, such as continuity of the interpolated valiable across the boundary of the interpolation stencil.

Default values are shown above.

#THREADEDBC command

```
#THREADEDBC
T                       UseAlignedVelocity
F                       DoConvergenceCheck
limited                 TypeBc (first/second/limited)
1e-6                    Tolerance
20                      MaxIter
```

This command sets things for the threaded field line algorithm. Ask Igor Sokolov if you want to learn more. Default values are shown.

#CHROMOEVAPORATION command

```
#CHROMOEVAPORATION
F         UseChromoEvaporation
```

By default, this logical is .false. the entholpy increase needed to heat the plasma flow across the transition region to the top temperature is neglected. If logical set is true, the energy frlux to/from the first control volume is accounted for

#TRANSITIONREGION command

```
#TRANSITIONREGION
T                    DoExtendTransitionRegion
3.0E+5               TeTransitionRegionSi
1.0E+4               DeltaTeSi  (read if DoExtendTransitionRegion is true)
```

The artificial expansion of the transition region is needed to resolve the Transition Region (TR) which is an extremely thin region in reality. To achieve the expansion, at temperatures below TeTransitionRegionSi the heat conduction coefficient is artificially enhanced and the radiation loss rate is modified accordingly. The profile of temperature and density in this case are maintained to be the same as in the actual transition region, however, the spatial scale becomes much longer, so that the TR may be modelled with feasible grid resolution.

If DoExtendTransitionRegion is false, the #TRANSITIONREGION command can be used to set the temperature of the top of the transition region. Then the special boundary condition (REB - radiation energy balance) is used at the "coronal base", while the temperature is fixed at Te=TeTopTransitionRegion.

Default value is DoExtendTransitionRegion = .false. and TeTransitionRegionSi = 4e5.

#THREADRESTART command

```
#THREADRESTART
T              DoThreadRestart
```

If the logical DoThreadRestart is set to true, at the initial iteration the plasma state on the threaded field lines is recovered from the saved files otherwise it is calculated from scratch.

3.8.20 Heliosphere specific commands

#THINCURRENTSHEET command

```
#THINCURRENTSHEET
F                        DoThinCurrentSheet
```

The thin current sheet option is based on the thin current sheet method of the ENLIL code. Numerical reconnection of magnetic field about the heliospheric current sheet is avoided by reversing the field direction in one hemisphere (the hemisphere for which the radial magnetic is negative). This method assumes that there is no guide field, which would otherwise start to reconnection. It is only intended for inner and outer heliosphere simulations, assuming no coronal mass ejections are present.

This method requires an equation model that contains the SignB variable. This variable is used to track where the field is reversed and where the current sheet is located by using a level set method for the sign.

Default value is DoThinCurrentSheet = .false.

#ALIGNBANDU command

```
#ALIGNBANDU
T                        UseChGL
2.5                      RSourceChGL
2.5                      RMinChGL
```

To use this command, the SignB variable of the state vector should be declared in the ModEquation (the possible choice is -e=AwsomChGL). Given proper boundary conditions, in steady state the streamlineas and magnetic field lines are aligned and the ratio of magnetic flux to mass flux is constant along the magnetic flux tube, hence, the ratio of the magnetic field to velocity vectors obeys a conservation law.

Below RSourceChGL this ratio (Chew-Golberger-Low state variable is set $U.B/U^2$. If RSourceChGL is zero, then the ChGl variable is either obtained from the model below, for example from SC to IH, or set to zero identically.

Above RMinChGL, the magnetic field is enforced to be aligned with the velocity vector and equal to the velocity vector multipled by the local value of the ChGL variable. If RMinChGL is zero, then the magnetic field is aligned everywhere. If it is negative or larger than the size of the domain, then it is not aligned anywhere. The above setting is recommended for the steady-state SC. The recommended setting for the coupled steady state IH is

```
#ALIGNBANDU
T                       UseChGL
0.0                     RSourceChGL    ! Coupled
0.0                     RMinChGL       ! Applied everywhere
```

For the time accurate run in SC one can set UseChGL to .false., while in IH one may want to gradually increase RMinChGL to keep good steady-state solution in the region not affected by the time-accurate perturbation prropagating outward.

#HELIOUPDATEB0 command

```
#HELIOUPDATEB0
-1.0                    DtUpdateB0 [s]
```

Set the frequency of updating the B0 field for the solar corona. A negative value means that the B0 field is not updated.

#HELIODIPOLE command

```
#HELIODIPOLE
-0.0003                 HelioDipoleStrengthSi [Tesla]
 0.0                    HelioDipoleTilt     [deg]
```

Variable HelioDipoleStrengthSi defines the equatorial field strength in Tesla, while HelioDipoleTilt is the tilt relative to the ecliptic North (negative sign means towards the planet) in degrees.

Default value is HelioDipoleStrengthSi = 0.0.

#UNIFORMB0 command

```
#UNIFORMB0
5.0e-4                  UniformB0Si
0.0                     UniformB0Si
0.0                     UniformB0Si
```

Three components of uniform B0 field. In the example above there is a uniform field of 5E-4 T = 5 G along the x axis. Exception is the case of rz-geometry, in which the intensity of Phi (third) field component should be divided by dimensionless r.

#HELIOBUFFERGRID command

```
#HELIOBUFFERGRID
2               nRBuff
90              nLonBuff
45              nLatBuff
19.0            RBuffMin
21.0            RBuffMax
```

Define the radius and the grid resolution for the uniform spherical buffer grid which passes information from the SC(IH) component to the IH(OH) component. The resolution should be similar to the grid resolution of the coarser of the SC(IH) and IH(OH) grids. The buffer grid will only be used if 'buffergrid' is chosed for TypeBcBody in the #INNERBOUNDARY command of the target (IH or OH) component. This command can only be used in the first session by the IH(OH) component. Default values are shown above.

#RESTARTBUFFERGRID command

```
#RESTARTBUFFERGRID
T               DoRestartBuffer
HGR             TypeCoordSource
```

If .true. the MHD state on the buffer grid is restored from the restart file. The coordinate system is defined by TypeCoordSource. This command is usually read from the restart.H file.

Default value is DoRestartBuffer = .false.

3.8.21 Wave specific commands

#ADVECTWAVES command

```
#ADVECTWAVES
T                   DoAdvectWaves
```

If DoAdvectWaves = .true. the waves are advected in the energy dimension. This term may be very small and it can be switched off for purposes of testing or comaparison with other codes that do not have this term.

The default is false.

#ALFVENWAVES command

```
#ALFVENWAVES
T                   UseAlfvenWaves
```

If UseAlfvenWaves = .true. the waves are separated into two sets, one of them ('plus') propagate parallel to the magnetic field, the second one ('minus') is for waves propagating anti-parallel to the field. The propagation speed with respect to the background plasma is $\pm V\_A = \pm|B|/\sqrt{\rho}$.

#AWREPRESENTATIVE command

```
#AWREPRESENTATIVE
F                       UseAwRepresentative
```

If UseAlfvenWaveRepresentative is .true., in ExplicitAdvance at the beginning the wave energy densities are divided by sqrt(Rho)*PoyntingFluxPerB, to get representative functions, for which the equations and boundary conditions are easier to handle. After the stage loop, the functions are converted back to physically meaningful wave energy densities.

3.8.22 Particles

#PARTICLELINE command

Recommended version for IH/OH (the field line and particle numbers depend on both pratical needs and available computational resources)

```
#PARTICLELINE
T                       UseParticles
16                      nFieldLineMax
1000                    nParticlePerLine
-1                      SpaceStepMin
-1                      SpaceStepMax
import                  InitMode
T                       UseBRAlignment
```

```
0.1                     CosBRAngleMax
T                       UseBUAlignment
0.85                    CosBUAngleMax
```

Recommended version for SC

```
#PARTICLELINE
T                       UseParticles
16                      nFieldLineMax
1000                    nParticlePerLine
-1                      SpaceStepMin
-1                      SpaceStepMax
import                  InitMode
T                       UseBRAlignment
0.7                     CosBRAngleMax
F                       UseBUAlignment
```

The command defines the class of advected magnetic field lines, consisting of "particles" which are essentially the Lagrangian grid points. Initially, the magnetic field lines are traced starting from the origin point set, determined by the value of InitMode parameter (their coordinated are imported from another model or just preset in the parameter file).

Based on values of SpaceStepMin and SpaceStepMax space step may be

Field lines may need to be corrected during tracing. If the line 'too much deviates' from radial direcction (turns toward the Sun or tends to turn), its direction is corrected to keep outward direction. If the tracing occurs just after computing the steady state solution of the solar wind the corrrection limits the angle between the line direction (= the magnetic field vector direction) and that of the solar wind velocity in the corotating frame of reference. The magnetic field and velocity vectors may be parallel or antiparallel. In case the magnetic field line intersects the current sheet, the traced line is gradually switched between parallel and antiparallel directions, keeping the general direction along the Parker spiral.

3.8.23 Script commands

#INCLUDE command

```
#INCLUDE
GM/restartIN/restart.H          NameIncludeFile
```

Include a file. The most useful application is including the restart header file as show by the example. Including this file helps making sure that the original and restarted runs use consistent settings. The #INCLUDE command can also be useful if a sequence of commands is used many times in different parameters files. For example one can define a typical grid for some application and reuse it. Nested include files are allowed but not recommended, because it makes things difficult to track. Using #INCLUDE can make the main PARAM.in file shorter. On the other hand, distributing the input information over several files is more error prone than using a single file. A PARAM.in file with included files can be expanded into a single file with the

```
share/Scripts/ParamConvert.pl run/PARAM.in run/PARAM.expand
```

script. Note that the include command for the restart header file is not expanded.

The default is to use a single PARAM.in file.

Chapter 4

Output files

4.1 Restart files

Restart files contain all the necessary information that enables BATS-R-US to continue a simulation from a given point. The parameters following the #SAVERESTART command in PARAM.in determine when the restart files are written to the disk. The restart files are written into the output restart directory, which has the default name

```
restartOUT/
```

The default name can be changed with the #RESTARTOUTDIR command in PARAM.in. The state variables are stored in the binary files

```
restartOUT/blkGLOBALBLKNUMBER.rst
```

Note that the files are written out block by block, so that at restart the blocks can be distributed among the processors differently than they were at the time of the save. This means that a simulation can be continued with a different number of processors after restart. On the other hand the file names do not contain the time step, so they get overwritten every time new restart files are produced. This avoids the problem of filling up the disk with restart files. The only way to keep restart files is to rename the restartOUT directory. For example

```
mv restartOUT restart_n120000
mkdir restartOUT
```

Remember to create an empty restartOUT directory, otherwise the code will crash when it attempts to write a restart. Moving the restartOUT file can be done even while the code is running, except when the restart files are being written. Alternatively you can create several directories in advance and change the name of the output restart directory with the #RESTARTOUTDIR command from session to session.

If you want BATS-R-US to read the restart files in, they have to be located in the input restart directory. The default name is

```
restartIN/
```

Rather than moving the files into this directory, we suggest the use of a symbolic link. For example

```
ln -s restart_n120000 restartIN
```

will make the code read the files from the `restart_n120000` directory if a restart is initiated in PARAM.in. Alternatively, you can use the #RESTARTINDIR command in the PARAM.in file to change the name of the input restart directory.

In addition to the block data files, the restart directory contains two more files. The octree grid structure is described by the binary file

```
restartOUT/octree.rst
```

while the ASCII header file

```
restartOUT/restart.H
```

contains time step and time information for the restart. It also signals whether the restart files contain extra data for face centered magnetic field variables for the constrained transport scheme.

4.1.1 Conversion of binary restart files with ConvertRestart.pl

The binary restart files, like all binary Fortran files, are platform dependent. Platforms differ in how they order the bytes in the 4 or 8 byte data units, such as integers, real numbers, logicals and record length markers. For example the Linux PC, the SGI Altix and the Compaq OSF1 machines are 'little endian', while the SGI Origin is 'big endian'.

We have developped a tool that makes the conversion of the restart files rather easy. The `Scripts/ConvertRestart.pl` perl script can convert a complete restart directory, for example

```
mkdir run/restart_converted
Scripts/ConvertRestart.pl run/restart_original run/restart_converted
```

Depending on the number and size of the restart files conversion may take from a few minutes to an hour.

4.2 Logfiles

The logfiles are very useful to check conservation of quantities, to see when not-a-numbers (NaN) first appeared before the code crashed, or it can contain physically relevant pointwise values. Writing the logfile is fairly fast, and the logfile is relatively small, so it can be easily done every time step if that is necessary (e.g. for debugging).

The ASCII logfile is written into the plot directory. The default name is

```
IO2/
```

The name of the directory is historical (there used to be an IO directory), and it can be changed with the #PLOTDIR command in the PARAM.in file. The frequency of saves and the content of the logfile are controlled by the #SAVELOGFILE command in PARAM.in. The logfile is named

```
IO2/log_timestep.log
```

where `timestep` refers to the time step when the logfile is opened. The logfile is only closed at the end of the run. The logfile has a very simple structure:

```
headerline
it t var1 var2 var3 ...
...
```

where the `headerline` is a string describing the content. The second line is a list of the function names that are saved usually starting with time step number and time (see the #SAVELOGFILE command for details). All subsequent lines contain the values for these functions at the given time step. The logfile can be viewed with the UNIX `more` command or with any editor (**but do not use an editor if BATS-R-US is running and the logfile is still open**), or it can be read with the `.r getlog` script into IDL and plotted.

4.3 Satellite Files

Satellite files are used to extract information along a satellite trajectory. This is most useful in time accurate runs when saving the full 3D files frequently enough to extract the data along the trajectory in post processing is prohibitive. However, they have other uses.

The ASCII satellite files are also written into the plot directory. The frequency of saves and the content of the satellite files are controlled by the #SATELLITES command in PARAM.in. The satellite files are

```
IO2/satelite_NN_satelitename.sat
```

where NN refers to the number of the satellite and satellitename is obtained from the input from PARAM.in. These files are closed at the end of the run. The satellite files have the same structure as the log file:

```
headerline
it t var1 var2 var3 ...
...
```

where the headerline is a string describing the content. The second line is a list of the function names that are saved starting with time step number and time (see the #SATELLITE command for details). All subsequent lines contain the values for these functions at the given time step. The satllite files can be viewed with the UNIX more command or with any editor (**but do not use an editor if BATS-R-US is running and the files are still open**).

4.4 Plotfiles

Plot files are used for visualization of spatially and temporarily distributed data. They can contain the values of various functions along 1, 2 or 3 dimensional cuts of the computational domain. A series of plot files can be used to animate the time evolution.

All the plotfiles are written into the plot directory, which has the default name IO2/ which can be changed with the #PLOTDIR command. The type and number of plot files and the frequency of saves are controlled by the #SAVEPLOT command in PARAM.in.

Separate plot files are written out by each processor. The file names ensure that different plot files have different names, and previously written saves do not get overwritten. The name looks like this

```
IO2/plotarea_plotvar_plotnumber_timestep_PEnumber.plotform
```

where plotarea and plotvar are given in PARAM.in, plotnumber is the serial number of the plot file based on the order of plot file definitions after the #SAVEPLOT command; timestep is the number of time steps for the whole simulation; PEnumber is the processor (starting from 0); and finally plotform is 'idl' for IDL and 'tec' for TEC files.

The IDL files contain cell centered data, such as cell size, cell center position, and function values, while the TEC files contain data interpolated to cell corners. The TEC files are always in ASCII format. The IDL files can be ASCII or binary as determined by the #SAVEBINARY command in PARAM.in. Binary files are smaller, faster to write, and there is no loss of accuracy, so this is the default. The ASCII files on the other hand, are human readable, and transferable between machines.

In addition to the '*.idl' and '*.tec' files, an ASCII header file is written for each plotfile with a name

```
IO2/plotarea_plotvar_plotnumber_timestep.headextension
```

where headextension is 'h' for the IDL and 'T' for the Tecplot file formats. The header files contain basic information, like time step, time, variable names, variable units etc. An exception to the above header extension is for the spherical cut plot files (plotarea='sph'). The Tecplot header files for these plots have the headextension 'S'.

4.5 Postprocessing the IDL plot files

Since the plot files are produced by each processor separately, it is necessary to collect the data and produce a single plot file. Furthermore, we can produce a single structured grid with a fixed resolution defined by the DxSavePlot parameter in the #SAVEPLOT command. The requested resolution is saved in the .h header file. The differently sized cells must be restricted and prolonged (with first order accuracy) into equally sized cells, which then must be arranged into a structured grid. When the files contain a 2D cut (e.g. y=0), the data in the cells on the two sides of the cut plane must be averaged. This is done both for structured and unstructured grids.

All the data collection and transformation described above are done by the

```
PostIDL.exe
```

code. To make this executable simply type

```
make PIDL
```

in the main directory. In the run directory there is a symbolic link to PostIDL.exe. To post process all the .idl files for a single plot, type for example

```
PostIDL.exe < IO2/y=0_MHD_1_n0001200.h
```

in the run directory. The PostIDL code will read the headerfile, and all the corresponding .idl files, and it will produce a single binary file

```
IO2/y=0_MHD_1_n0001200.out
```

This file contains all the header information and the information collected from the .idl files. It can be read and visualized with the IDL macros provided in the Idl/ directory. Therefore the .h and .idl files can be deleted.

When there are many plotfiles produced by a simulation, all the Postprocessing can be done with a single command by running the

```
pIDL
```

script. There is a link in the run directory to this script. The `pIDL` script does the post processing for all plots with a corresponding .h header file in the IO2/ directory. After post processing it deletes the .h and .idl files automatically. This default behavior can be modified with the two optional arguments. The first argument limits the post processing to a subset of the plots, e.g.

```
pIDL IO2/x=0
```

will only process the plotfiles whose names start with 'x=0'. The second argument tells pIDL that the .h and .idl files should be kept. For example

```
pIDL IO2/ KEEP
```

will process all the plots, but it keeps the original data files too.

The PostIDL.exe program can read both ASCII and binary .idl files. For binary .idl files, however, it is important to have the same precision for reals as in BATS-R-US, ie. PostIDL.exe and BATSRUS.exe should be compiled with the same PRECISION definition in the Makefile. The output of PostIDL.exe is a binary file, because IDL reads binary files much faster than ASCII files. It also saves disk space. For testing purposes, however, `/src/PostIDL.exe` can be edited to contain

```
logical, parameter :: write_binary=.false.
```

instead of the default .true. value. After recompilation with `make PIDL`, the modified PostIDL.exe code will write ASCII output files.

4.5.1 Conversion of binary .out files with FixEndian.pl

The **Scripts/IDL/FixEndian.pl** Perl script solves the problems of transporting the binary .out files between different platforms. The script can tell whether a machine is big endian or little endian (these differ in the ordering of bytes for integers and reals), and it can test and convert the endianness of binary .out files, both in double and single precisions. As an additional benefit, the 8-byte integers used by Cray can be converted to 4 byte integers and vice-versa.

Note that the 'assign -F f77 u=12' command has to be used on the Cray before running PostIDL.exe, so that the output file is in Fortran 77 binary format. Then the file should be FTP-d to a workstation/PC, and the FixEndian.pl script should be executed there. The script does not work on the Cray, because even the Perl interpreter is non-standard on a Cray.

Type **Scripts/IDL/FixEndian.pl** without any parameters to see the syntax of usage. An example of usage is to check the content of a data file with the **-t** flag (test):

```
Scripts/IDL/FixEndian.pl -t run/IO2/y=0_mhd_1_n0001200.out

This is a big endian machine.

run/IO2/y=0_mhd_1_n0001200.out is a big endian single precision file.

headline=normalized variables
it=1200 t=0 ndim=2 neqpar=2 nw=11
nx=256,128
eqpar=1.66666,0
names=x z rho ux uy uz bx by bz p jx jy jz g cuty
...
```

The above output was obtained on an SGI workstation for a .out file obtained with an SGI Origin. In this case no conversion is required.

4.6 Postprocessing the TEC plot files

Since the plot files are produced by each processor separately, it is necessary to collect the data and produce a single plot file. Furthermore, Tecplot output files contain cell corner data rather than the cell centered data of IDL. Node numbering and block edge synchronization occur prior to output so that very little processing needs to be done with the output files.

Processing the `.tec` files happens in two separate steps. First, the individual files from each processor for each plot have to be concatenated together and renamed as `.dat` files. Next, the files can be be processed with `preplot`, an application included with Tecplot, which creates a binary file in tecplot native format.

A script has been created to more easily process Tecplot data. There is a link to this script called

```
pTEC
```

in the run directory. This script handles the processing of several different output data products including standard 2 or 3 dimensional output slices and spherical slice files in the `IO2/` directory. Typing

```
pTEC -help
```

gives the following documentation on the type of arguments to `pTEC` and their functionality.

```
Usage:

pTEC [p,r,g,I,S,T,A,b=BASENAME]
```

```
The order of flags does not matter.  A maximum of 6 flags can be used.

- No arguments default to 'T'. See below.
- If 'S' then spherical tecplot files are processed.
- If 'T' then 2D and 3D tecplot files are processed.
- If 'A' then all three file types are processed.
- If 'p' and preplot is available in the PATH then it will also be run.
     - If 'r' is also specified, the .dat file will be deleted after preplot.
     - If no 'r' is specified, the .dat file will be gzip'd
- If 'g' and not 'p' then the .dat files will be gzip'd.  Ignored if 'p'.
- If 'b=BASENAME' then only files starting with the path BASENAME are
     processed.
     - To process all files in the IO2 directory use 'b=IO2/'   not 'b=IO2'
     - If 'b=' is not used 'T' and 'S' files are processed in IO2/
```

The `pTEC` script can be used with up to 6 of the 8 flags in any combination or order. However, some of the combinations are not meaningful (we outline these below).

The capital flags control which type of files will be processed: (`S`) spherical slice, (`T`) 2D and 3D tecplot slices and (`A`) all types (`S+T`). If none of these flags is given, then by default only 2D and 3D tecplot slices (`T`) will be processed. Using more than one of the flags will have the intuitive result. By default the files are processed in the `IO2/` directory. This default behavior can be changed as described below.

For each type of file the individual file from each processor are concatenated or processed into `.dat` files. This is an ASCII file that can be read by tecplot. Creation of 2D and 3D files just requires concatenation of the `.tec` files (the associated `.T` header files are required for determining the number and names of files but are not included in the concatenation). For spherical files the program `PostSPH.exe` is run. This programs must be compiled before spherical files can be processed. Go to the main directory of the distribution and type

```
make PSPH
```

For spherical plot files the header file (`.S`) and the various other files from the different processors (`.tec`) are read into `PostSPH.exe` and written out in the correct format.

Lower case flags control how much processing should take place after the creation of the `.dat` file. The '`p`' flag indicates that the file should be converted to the tecplot native binary format using the program `preplot` provided in the tecplot distribution. The resulting binary files have the extension `.plt` and are machine independent (see below for a description of preplot). When '`p`' is set then the '`r`' flag can be used. With '`p`' only, the `.plt` files are created and the original `.dat` files are gzipped to `.dat.gz` and saved. If in addition the '`r`' flag is set then the `.dat` files are removed and only the `.plt` files remain.

Frequently the code will be run on a remote machine that does not have `preplot` installed. In this case the '`g`' flag can be used. If this flag is set the `.dat` files are gzipped to `.dat.gz` so that they are more easily sent over the internet. This flag is ignored if '`p`' is set.

Finally, the `b=BASENAME` flag can be used to process files in any directory or to process a subset of the files. For this flag `BASENAME` is a string that indicates the path and the initial part of the filename that should be processed. The `BASENAME` does not accept wild cards. As an example, the command

```
pTEC S b=IO2_save/spN
```

will process only spherical slice files in the directory `IO2_save/` whose filename starts with `spN` (in other words `spN*`). The command

```
pTEC S b=IO2_save/
```

would process all the spherical plot slices in the directory `IO2_save/`. Note that the trailing '/' is required so that the string `IO2_save` is interpreted as a directory and not the base filename to be used in the current directory. Also note that there is only one '`b=`' flag, but that there are three different file types that are typically located in two

different directories. If the 'b=BASENAME' flag is used, all file types will be looked for the the directory indicated by BASENAME. For example, the command

```
pTEC A b=IO2/
```

will try to process all three file types (I+S+T) out of the IO2/ directory. It will find and process the 'S' and 'T' files which would normally be located in this directory, but would not find and process the 'I' files unless they had been moved. We point out that the command

```
pTEC A
```

will process each file type in the default plot directory IO2/.

As indicated above Tecplot understands both ASCII files and files in its native binary format. Tecplot can read the binary files more quickly the the ASCII files. We indicated above that .dat ASCII files can be converted to Tecplot binaries by running the command

```
preplot newplotarea_plotvar_plotnumber_timestep.dat
```

This will create the file

```
newplotarea_plotvar_plotnumber_timestep.plt
```

without deleting the .dat file. These binary files are read more quickly by Tecplot. Preplot is a Tecplot program that comes with the distribution of Tecplot. Note that preplot only converts a singe file at a time and does not accept wild cards. In order to process several files at once you must write a script that will do this for you. The pTEC script above can preplot the files for you while processing all the plot output. Alternatively, in the Scripts/TEC/ directory there are several scripts that process an entire directory worth of tecplot files with preplot.

```
ppA
```

converts all .dat files in the current directory to .plt files using preplot. The .dat files remain.

```
ppAr
```

converts all .dat files in the current directory to .plt files and then deletes the .dat files.

```
ppgz KEEP
```

gunzips all .dat.gz files in the current directory. It then converts them to .plt files using preplot. If the KEEP argument is present the .dat files are again compressed and retained. It the KEEP argument is missing the scripts will delete the .dat files.

To finish, let us give two additional examples of processing Tecplot files. First, assume that BATS-R-US has been run on 5 processors and has written files for a single plot at y=0 at time step 100. From the run directory, the command

```
ls IO2/*
```

would give the following output

```
IO2/y=0_mhd_1_n0000100.T
IO2/y=0_mhd_1_n0000100_pe0000.tec
IO2/y=0_mhd_1_n0000100_pe0001.tec
IO2/y=0_mhd_1_n0000100_pe0002.tec
IO2/y=0_mhd_1_n0000100_pe0003.tec
IO2/y=0_mhd_1_n0000100_pe0004.tec
```

This shows the header file and the individual Tecplot files, one for each processor. Typing the command

```
pTEC
```

will process these data files. Listing the IO2 directory again would show the following file:

```
IO2/y=0_mhd_1_n0000100.dat
```

The file is simply the concatenation of each of the individual files. To convert to Tecplot binary files we could then type

```
pTEC p
```

listing the directory will show the original file plus the new binary file.

```
y=0_mhd_1_n0000100.dat.gz
y=0_mhd_1_n0000100.plt
```

Of course this last step assumes that the current machine has preplot installed. The same effect could have been achieved by typing one of the following commands

```
pTEC p
pTEC p T
pTEC p b=IO2/
pTEC p T b=IO2/
pTEC p T b=IO2/y=
```

As a second example, assume again that BATS-R-US has been run on 5 processors. This time, at step 100, 2 files have been written: x=0 and a spherical slice. We are going to assume that the run was done on a remote machine that does not have tecplot installed. We wish to process the files as much as possible on the remote machine, bring them back to a machine that does have tecplot and preplot installed and then finish the processing locally. We will also assume that for some reason the directories in which the files are stored on the remote machine are not the standard ones. For this example the files that would normally have been in `IO2/` have been moved to `IO2_run1/`.

To begin we need to compile the post processing codes. Go to the main directory of the distribution and type

```
make PSPH
```

This creates `PostSPH.exe`, which is used to process the spherical plot files.

Now move to the run directory and view the files. The command

```
ls IO2_run1/*
```

would give the following output

```
IO2_run1/spN_mhd_2_n0000100.S            IO2_run1/x=0_mhd_1_n0000100.T
IO2_run1/spN_mhd_2_n0000100_pe0000.tec   IO2_run1/x=0_mhd_1_n0000100_1_pe0000.tec
IO2_run1/spN_mhd_2_n0000100_pe0001.tec   IO2_run1/x=0_mhd_1_n0000100_1_pe0001.tec
IO2_run1/spN_mhd_2_n0000100_pe0002.tec   IO2_run1/x=0_mhd_1_n0000100_1_pe0002.tec
IO2_run1/spN_mhd_2_n0000100_pe0003.tec   IO2_run1/x=0_mhd_1_n0000100_1_pe0003.tec
IO2_run1/spN_mhd_2_n0000100_pe0004.tec   IO2_run1/x=0_mhd_1_n0000100_1_pe0004.tec
IO2_run1/spS_mhd_2_n0000100.S            IO2_run1/x=0_mhd_1_n0000100_2_pe0000.tec
IO2_run1/spS_mhd_2_n0000100_pe0000.tec   IO2_run1/x=0_mhd_1_n0000100_2_pe0001.tec
IO2_run1/spS_mhd_2_n0000100_pe0001.tec   IO2_run1/x=0_mhd_1_n0000100_2_pe0002.tec
IO2_run1/spS_mhd_2_n0000100_pe0002.tec   IO2_run1/x=0_mhd_1_n0000100_2_pe0003.tec
IO2_run1/spS_mhd_2_n0000100_pe0003.tec   IO2_run1/x=0_mhd_1_n0000100_2_pe0004.tec
IO2_run1/spS_mhd_2_n0000100_pe0004.tec
```

Note that the northern and southern hemispheres are separated for the spherical plot files. In addition note that for each 'PE' there are two x=0 files

```
IO2_run1/x=0_mhd_1_n0000100_1_pe0000.tec
IO2_run1/x=0_mhd_1_n0000100_2_pe0000.tec
```

The first of these contains the data for this processor and the second contains the tree information for connecting the blocks together.

Since tecplot and preplot are not installed on this machine we can only create the `.dat` files here (the 'p' option will not do anything). Because the files have been moved from their standard directories (`IO2/` we will have to use the 'b=' flag. We can process the files using the command

```
./pTEC g T S b=IO2_run1/
```

Note that the three file types cannot be processed with a single command since they have been moved to new directories. The output from the command would be

```
================================================================================
Beginning cat of .tec files
================================================================================
--Working in directory: IO2_run1/   on files: ./*.T ./*.tec
   working on x=0_mhd_1_n0000100 ...
   compressing all *.dat files in directory: IO2_run1
================================================================================
Beginning processing of spherical slice files
================================================================================
--Working in directory: IO2_run1/   on files: ./*.S ./*.tec
   working on spN_mhd_2_n0000100 ...
   working on spS_mhd_2_n0000100 ...
   compressing all *.dat files in directory: IO2_run1
```

We would now bring the files home using ftp or scp. In this example we copy all the files into an already existing directory on our home machine. From the run directory type

```
scp IO2_run1/*.dat.gz myname@mymachine.edu:testdir/
```

This would secure copy the `.dat.gz` files to a machine named `mymachine.edu` to an account with username `myname` into the directory `testdir/`.

On my home machine I have placed the following files in the `bin` directory in my home directory (which is included in my PATH).

```
pTEC
ppgz
PostSPH.exe
preplot
```

Since all these programs can now be found in the path, to finish processing the files I simply go to the directory `testdir` on my home machine and do one of two things. The first option is to type

```
gunzip *
pTEC A b=
```

which will gunzip all the files and then process them with `pTEC`. Since they have already be concatenated, the preplotting step will be the only on executed. This command will leave the original files and a listing of files in the directory will give

```
spN_mhd_2_n0000100.dat.gz
spN_mhd_2_n0000100.plt
spS_mhd_2_n0000100.dat.gz
```

```
spS_mhd_2_n0000100.plt
x=0_mhd_1_n0000100.dat.gz
x=0_mhd_1_n0000100.plt
```

Alternately, the simple command

```
ppgz
```

will gunzip all the files, preplot them and then delete the original files. A listing of the directory would give.

```
in000100N.plt
spN_mhd_2_n0000100.plt
spS_mhd_2_n0000100.plt
x=0_mhd_1_n0000100.plt
```

Chapter 5

Visualization

5.1 Tecplot

Tecplot is a visualization package created by Amtec Engineering, Inc. out of Bellevue, Washington. The package was originally designed for visualization of the output of computational fluid dynamics (CFD) codes and now is a good multipurpose visualization package. The strengths of Tecplot are the point and click interface, the wide range of options and the ability to produce high quality three dimensional images in postscript, encapsulated postscript, tiff and other formats. The software comes with detailed documentation for processing data and creating images. Starting in 2017, Tecplot provides the PyTecplot (at https://www.tecplot.com/docs/pytecplot/) package for connecting Python scripts to the Tecplot 360 visualization engine.

Currently BATS-R-US supports output in Tecplot ASCII format, which can be processed by preplot into `.plt` binary format. There is also an option of `.dat` output of ASCII header with binary data and connectivity, which is more efficient but cannot be processed by preplot. This file can be read and converted into VTK files by the Julia package mentioned in Section **??**. The generated unstructured grid `.vtu` files can be read by Tecplot. The direct support for Tecplot binary formats `.plt` and `.szplt` may be included in the future.

Tecplot requires a license.

5.2 IDL

IDL is well suited to visualize 1D and 2D data on structured and unstructured (AMR) grids, and 3D data on structured grids. It is a full programming language that can perform complex data processing and visualization tasks efficiently. The IDL programs developed for the SWMF can be found in the `share/IDL/` directory including detailed documentation.

IDL requires a license, although it can be used in demo mode for free. In demo mode each session is limited to 7 minutes, which is actually sufficient to create plots.

5.3 MATLAB

MATLAB is a multi-paradigm numerical computing environment and proprietary programming language developed by MathWorks. Link to the MATLAB package VisAnaMatlab is can be found in the `share/MATLAB/` directory. This package can read, process, and plot SWMF data, similarly to the IDL scripts.

MATLAB requires a license.

5.4 Julia

sec:julia)

Julia is a high-level, high-performance, dynamic programming language well-suited for high-performance numerical analysis and computational sciences. The VisAnaJulia package, available in `share/Julia`, is developed to read, process and visualize SWMF output files. It also provides the functionality of converting Tecplot `.dat` format into VTK format. Documentations can be found inside the package.

Julia is open source, as is the VisAnaJulia package.

5.5 VisIt

VisIt is a high performance visualization package built upon the VTK library. It can read the HDF5 output with extension .batl.

Visit is free.

5.6 ParaView

ParaView is another high performance visualization package built upon the VTK library. The Tecplot `.dat` files from BATS-R-US can be converted into VTK files with the VisAnaJulia package for processing in ParaView.

Paraview is free.

5.7 Python

Python is a very popular object oriented programming language. SpacePy has been developed to read and visualize SWMF output. The swmfpy package in `share/Python` can perform various tasks related to the SWMF.

Python, SpacePy, swmfpy are all open source.

Chapter 6

The Synoptic Solar Wind Model

6.1 General Description of the Model

The synoptic solar wind model is designed to provide the the ambient physical conditions for the Solar Corona (SC), the Inner Heliosphere (IH), and the Solar Wind (SW). The model is called "synoptic" due to the fact that it is driven by synoptic maps of the observed surface radial magnetic field of the Sun throughout a whole Carrington Rotation (one rotation period of the Sun in approximately 27 days). These maps are used to provide the inner boundary conditions for the steady-state MHD solution in the domain between the Sun and the Earth.

There are two main issues when one tries to create numerical model for the solar corona. First, one needs to specify the initial condition for the three-dimensional configuration of the magnetic field. Since the only constrain on the magnetic field is the observed radial field on the surface, we use the common "Potential Field" approximation to specify the initial field in the whole domain (see Section 6.2). The other issue is the solar wind heating and accelerating mechanism. This issue is discussed in Section 6.3.

6.2 The potential field approximation

The solar corona is dominated by its magnetic field. Therefore, it is important to know what is the three dimensional structure of the magnetic field in order to study the physical processes in the corona. Since the solar magnetic field can be routinely measured only at the photosphere, where the plasma density is high enough for measuring the Zeeman Splitting, one needs to find a way to approximate the global structure of the coronal magnetic field. The most commonly used method to approximate the coronal magnetic field is the so-called 'potential field' approximation (Altschuler and Newkirk, Solar Physics, 9:131-149, 1969). In this approximation, it is assumed that the Alfvén speed is much larger than the speed of convective motions in the low corona, so the field line relaxation time is much shorter than the typical advection time scale. In other words, the field line will respond quickly to any motion we apply on it (this motion can be seen as electric current) so in practice the magnetic field is static. Under the assumption that there are no currents in a physical system, we can write Ampere's law as follows:

$$\nabla \times \mathbf{B} = 0, \tag{6.1}$$

and we can write $\mathbf{B}$ as a gradient of some scalar potential ψ:

$$\mathbf{B} = -\nabla\psi. \tag{6.2}$$

Since we also know that

$$\nabla \cdot \mathbf{B} = 0, \tag{6.3}$$

combining eq. 6.2 with eq. 6.3 results in the Laplace equation for the scalar potential:

$$\nabla^2\psi = 0. \tag{6.4}$$

The general solution for this equation in spherical coordinates for $R_0 < r < R_s$ is:

$$\psi(r,\theta,\phi) = \sum_{n=1}^{\infty}\sum_{m=0}^{n}\left[R_0\left(\frac{R_0}{r}\right)^{n+1} + R_s \cdot c_n\left(\frac{r}{R_s}\right)^{n}\right] \times \left(g_n^m \cos m\phi + h_n^m \sin m\phi\right) P_{nm}(\theta), \tag{6.5}$$

which gives $\psi = 0$ at $r = R_s$ for the choice of $c_n = -\left(\frac{R_0}{R_s}\right)^{n+2}$, particularly as $R_s \to \infty$. P_{nm} are the associated Legendre Polynomials, which are a function of $\cos\theta$. The coefficients g_n^m and h_n^m can be determined from the photospheric magnetic field data and have magnetic field dimension.

Following the above solution, we can obtain the solution for the magnetic field components (Altschuler et al. 1969, eqs. 8-10):

$$B_r = -\frac{\partial\psi}{\partial r} = \sum_{n=1}^{\infty}\sum_{m=0}^{n}\left[(n+1)\left(\frac{R_0}{r}\right)^{n+2} - n\left(\frac{r}{R_s}\right)^{n-1} c_n\right] \times \left(g_n^m \cos m\phi + h_n^m \sin m\phi\right) P_n^m(\theta), \tag{6.6}$$

$$B_\theta = -\frac{1}{r}\frac{\partial\psi}{\partial\theta} = -\sum_{n=1}^{\infty}\sum_{m=0}^{n}\left[\left(\frac{R_0}{r}\right)^{n+2} + c_n\left(\frac{r}{R_s}\right)^{n-1}\right] \times \left(g_n^m \cos m\phi + h_n^m \sin m\phi\right)\frac{\partial P_n^m(\theta)}{\partial\theta}, \tag{6.7}$$

$$B_\phi = -\frac{1}{r\sin\theta}\frac{\partial\psi}{\partial\phi} = \sum_{n=1}^{\infty}\sum_{m=0}^{n}\left[\left(\frac{R_0}{r}\right)^{n+2} + c_n\left(\frac{r}{R_s}\right)^{n-1}\right] \times \frac{m}{\sin\theta}\left(g_n^m \sin m\phi - h_n^m \cos m\phi\right) P_n^m(\theta). \tag{6.8}$$

We can determine the harmonic coefficients from the observed photospheric values of B_r as follows. The orthogonality relation over a sphere with $r = 1$ for the general Legendre functions is:

$$\frac{1}{4\pi}\int_{\theta=0}^{\pi}\int_{\phi=0}^{2\pi} P_{nm}(\theta)\left\{\begin{array}{c}\cos m\phi\\ \sin m\phi\end{array}\right\} P_{n'm'}(\theta)\left\{\begin{array}{c}\cos m'\phi\\ \sin m'\phi\end{array}\right\}\sin\theta d\theta d\phi = W\delta_{nn'}\delta_{mm'}, \tag{6.9}$$

where W is the normalization factor. For the general Legendre functions,

$$W = \frac{2}{2n+1}\frac{(n+m)!}{(n-m)!}, \tag{6.10}$$

and

$$W = \frac{1}{2n+1} \tag{6.11}$$

for the Schmidt normalization, so the relation between the Schmidt (P_n^m) and the general Legendre functions (P_{nm}) is:

$$P_n^m(\theta) = \left\{2\frac{(n-m)!}{(n+m)!}\right\}^{1/2} P_{nm}(\theta). \tag{6.12}$$

In BATS-R-US, the polynomials are calculated with the Schmidt normalization. For $r = R_0 = 1$, the radial magnetic field becomes:

$$B_r(\theta,\phi) = \sum_{n=1}^{\infty}\sum_{m=0}^{n} R_n \left\{ \begin{array}{c} g_n^m \\ h_n^m \end{array} \right\} P_n^m(\theta) \left\{ \begin{array}{c} \cos m\phi \\ \sin m\phi \end{array} \right\}, \tag{6.13}$$

where $R_n = \left[(n+1) + n\left(\frac{1}{R_s}\right)^{2n+1}\right]$.

Following eq. 6.9, we can obtain the harmonic coefficients from the photospheric magnetic data, assuming the Schmidt normalization of the Legendre functions (Altschuler et al. 1969):

$$\left\{ \begin{array}{c} g_n^m \\ h_n^m \end{array} \right\} = \frac{2n+1}{4\pi R_n} \int_{\theta=0}^{\pi}\int_{\phi=0}^{2\pi} B_r(r=R_\odot,\theta,\phi) P_n^m(\theta) \left\{ \begin{array}{c} \cos m\phi \\ \sin m\phi \end{array} \right\} \sin\theta d\theta d\phi, \tag{6.14}$$

where

$$B_r = \left\{ \begin{array}{ll} B_{magnetogram} & \text{for radial magnetogram,} \\ \frac{B_{magnetogram}}{\sin\theta} & \text{for Line-of-Sight magnetogram.} \end{array} \right\}.$$

The discretized version of eq. 14 is (Altschuler et al. 1969, eq. 15):

$$\left\{ \begin{array}{c} g_n^m \\ h_n^m \end{array} \right\} = \frac{1}{A}\frac{2n+1}{R_n} \sum_{i=0}^{N_\theta-1}\sum_{j=0}^{N_\phi-1} B_r(i,j)\cdot da_{i,j}\cdot P_n^m(\theta_i) \left\{ \begin{array}{c} \cos m\phi_j \\ \sin m\phi_j, \end{array} \right\} \tag{6.15}$$

where $da_{i,j} = \sin\theta_i d\theta d\phi$ and $A = \sum_{i=0}^{N_\theta-1}\sum_{j=0}^{N_\phi-1} da_{i,j} = 4\pi$ for $r = R_\odot$.

In SWMF, there is a utility tool to calculate the spherical harmonic coefficients from raw magnetogram. The tool is located at:

```
SWMF_DIR/util/DATAREAD/srcMagnetogram
```

This directory contains the following README file with instructions how to create the input harmonics file needed for the SC model:

```
#############################################################################
# How to create a magnetogram input file for SWMF from a raw magnetogram #
# fits file:                    #
#############################################################################

These are the steps for creating a magnetogram file for SWMF from
any raw magnetogram fits file.

1. Install SWMF (Config.pl -install).
2. Compile the HARMONICS executable by typing:
make HARMONICS
in the directory SWMF_path/util/DATAREAD/srcMagnetogram. This will
create the HARMONICS.exe executable in the directory SWMF_path/bin

3. For convenient, you can create a link to this executable from the path
SWMF_path/util/DATAREAD/srcMagnetogram by typing:
ln -s ../../../bin/HARMONICS.exe HARMONICS.exe
4. Type:
cp your_magnetoram_file.fits fitsfile.fits
5. Convert the fits file to ASCII format by running the idl program
```

```
fits_to_ascii.pro. You will be asked to insert the maximum order of
harmonics and the Carrington Rotation number. It is recommended (but not
required ) to use not more than 90 harmonics, since the computation time
can be very long.
   The idl routine generates three files:
*fitsfile.dat - ASCII file to be used by HARMONICS.exe to calculate
 the harmonic coefficients.
*fitsfile.H - the header of the original fits file with information
 about the magnetogram source.
*fitsfile_tec.dat - a Tecplot file to display the original magnetogram.
6. Run HARMONICS.exe. This executable can be run in parallel mode for faster
computation. This run will generate a file called harmonics.dat that
can be used in SWMF. For convenient, it is recommended to rename the file with
the following naming format:
cp harmonics.dat CRxxxx_OBS.dat
where xxxx is the Carrington Rotation number and OBS is the observatory name
(MDI,WSO,MWO,GONG etc.)
7. Move the magnetogram harmonics file to the path defined in the PFSSM flag
in PARAM.in file (usually run/SC).
```

Note: this routine does not interpolate missing data or the polar flux. You have to make sure that the raw magnetogram is properly processed!!!!

6.3 Semi-empirical Model for the Solar Wind

Numerical reproduction of the solar corona steady-state conditions has been extensively investigated since the famous work by (Pneuman and Kopp, Solar Physics, 18:258, 1971).Traditionally, the deposition of energy and/or momentum into the solar wind has been described by means of some empirical source terms (Usmanov 93, McKenzie 97, Mikic 99, Suess 99, Wu 99, Groth 00). In these models, the sources of plasma heating and solar wind acceleration are typically modeled in a qualitative sense, and the spatial profiles for the deposition of the energy or momentum are usually modeled by exponentials in radial distance. In more realistic models, the solar wind is heated and accelerated by the energy and momentum interchange between the solar plasma and large-scale Alfvén turbulence (Jacques 77, Dewar 70, Barnes 92, Usmanov 00 and 03).

Due to the small number of observations at 1 AU, it is reasonable to adopt semi-empirical models. Assimilating a long history of solar wind observations, these models are very efficient and quite accurate. A particular example is the Wang-Sheeley-Arge model (WSA, Arge and Pizzo 00,Arge et. al 04). This model uses the observed photospheric magnetic field to determine the coronal field configuration, which is then used to estimate the distribution of the final speed of the solar wind, u_{sw}. The common disadvantage of semi-empirical models is that they are physically incomplete.

We use the model by Cohen et. al (07) to obtain the steady-state SC and IH solution. The SC and IH modules of SWMF are versions of the BASTRUS global MHD code, which is fully parallel and has adaptive mesh refinement capabilities (see Powell 99). Our SC model is driven by high-resolution SOHO MDI magnetograms. We use the magnetogram data to calculate the potential magnetic field, assuming the source surface is at $R_{ss} = 2.5R_{\odot}$, where $R_{\odot}$ is the solar radius, and use this distribution of the magnetic field as an initial condition.

The heating and acceleration of the solar wind plasma are achieved by using a non-uniform spatial distribution of γ. In order to obtain a more realistic distribution, we use the empirical Wang-Sheeley-Arge (WSA) model as an input to our model. The WSA model uses the potential field distribution to obtain the magnetic flux tube expansion factor defined as (Wang and Sheeley 90):

$$f_s = \frac{|B(R_s)|R_s^2}{|B(R_0)|R_0^2}. \tag{6.16}$$

The WSA model provides an empirical relation for the spherical distribution of the solar wind speed at 1AU as a function of f_s and the angular distance of a magnetic field footpoint from the coronal hole boundary, θ_b. In our model, we use the following formula (Arge et. al 2004):

$$u_{sw} = 265 + \frac{1.5}{(1+f_s)^{1/3}} \left\{ 5.9 - 1.5 e^{\left[1-(\theta_b/7)^{5/2}\right]} \right\}^{7/2} \text{ km s}^{-1}. \tag{6.17}$$

A more up-to-date formula (after personal communication with N. Arge 2006) is:

$$u_{sw} = 240 + \frac{675}{(1+f_s)^{1/4.5}} \{1 - 0.8 \frac{e^{[1-(\theta_b/2.8)^{5/4}]}}{e^1}\}^3 \; km \; s^{-1} \tag{6.18}$$

We should note two important issues about the WSA model. First, one should be aware of the fact that the WSA solution depends on the magnetogram resolution and the is not the same for different observatories. This is due to the different mapping of the potential field and the expansion factor. Second, the WSA fitting is done for 1AU, while we use it in the model at the source surface. This is the main reason for deviations of the MHD solution from the WSA solution.

In order to relate the surface value of γ to the WSA solar wind speeds, we assume that far from the Sun the total energy is dominated by the energy of the bulk motion and that the thermal and gravitational energy are negligible. We also assume that at the coronal base the bulk kinetic energy is zero. Due to energy conservation, we can use the Bernoulli equation to relate the two ends of a streamline (or magnetic field line):

$$\frac{u_{sw}^2(\theta,\phi)}{2} = \frac{\gamma_0(\theta_0,\phi_0)}{[\gamma_0(\theta_0,\phi_0)-1]} \frac{p_0(\theta_0,\phi_0)}{\rho_0(\theta_0,\phi_0)} - \frac{GM_\odot}{R_\odot}. \tag{6.19}$$

Here u_{sw} is the input solar wind speed from the WSA model, G is the gravitational constant, and $M_\odot$ is the solar mass. γ_0, p_0, and ρ_0 are the photospheric values for the polytropic index, pressure, and mass density. The coordinates θ_0, ϕ_0 represent the location of the field line footpoint, $u_{sw}(\theta,\phi)$ originated from. We interpolate γ from its photospheric value to a spherically uniform value of 1.1 on the source surface at $r = 2.5R_\odot$. γ is linearly varied from 1.1 to 1.5 for $2.5R_\odot < r < 12.5R_\odot$, and $\gamma = 1.5$ above $12.5R_\odot$.Once the spatial distribution of γ is obtained, we solve the MHD equations self-consistently using this location dependent polytropic index in the energy equation to obtain the steady state solution for the SC and solar wind.

The above distribution of γ enables us to reproduce the bi-modal structure of the solar wind speed. However, the distributions of the coronal density and temperature are still not determined. It is known that the faster wind originates from coronal holes, where the density is lower than the density in the closed field regions. In order to obtain this observed property, we scale the base density, ρ_0, and the base temperature, T_0, at each point on the solar surface with the inverse of the input speed from the WSA model. We would like our model to be driven only by the magnetogram data without any particular parameterization for each Carrington Rotation (CR). Therefore, we parameterize the model for the general cases of solar minimum and solar maximum conditions.

The general method to obtain steady-state solution from the Sun to 1AU is to run the SC component to steady-state, then turn on IH and couple the two components for some time (1 iteration or more). the coupling will drive IH through the inner boundary conditions (provided by SC) and the since the solar wind is supersonic at this point, a steady-state is obtained quickly in IH as well. In principle, it is enough to couple SC-IH for only one iteration. It is possible to do the coupling for longer period in order to obtain slightly higher magnetic field close to the equator due to the different boundary conditions with and without the coupling. Our experience however showed that the difference is minimal.

The SC-IH runs can be done in the HGR coordinate system (the frame rotating with the Sun) or in the HGI frame (inertial frame). The HGR run can be done in a local time stepping mode, while the HGI run must be done in a time-accurate mode. There are sample PARAM.in files to obtain SC-IH steady-state solution in either frame:

`SWMF_DIR/Param/PARAM.in.test.start.SCIH.HGR`

`SWMF_DIR/Param/PARAM.in.test.start.SCIH.HGI`

In principle, you only need to change the parameters related to the particular CR such as magnetogram file name, #STARTTIME and satellite files. However, from our experience the free parameters of the model should be changed as well.

6.4 Model Parameterization

The SC model was originally planned to have fixed parameters so the only change from one CR to another is the input magnetogram. Our experience has shown however, that a better solution can be obtained for a particular CR by modifying the base density (BodyNDim in the #BODY command) and the magnetogram scaling factor (UnitB in the #PFSSM command). The value range for BodyNDim should be $1 \times 10^8 - 5 \times 10^8$ (in cm^{-3}) and for UnitB should be $1 - 4$. The recommended scaling factor for MDI magnetograms is 1.8 and our experience showed that a value of 2.5 is better for solar minimum rotations of 1997. For solar maximum, we recommend to use higher value of 4.

We should note that the fine tuning is important for obtaining good agreement with 1AU data, which is very hard when using global model. The parameterization should be easier in the case of simulations of the solar corona only. A more detailed validation of the model can be found in:

```
Cohen, O.; Sokolov, I. V.; Roussev, I. I.; Gombosi, T. I.,
Validation of a synoptic solar wind model,
Journal of Geophysical Research, VOL. 113, A03104, doi:10.1029/2007JA012797, 2008
```

All parameterization above was done using MDI magnetograms. For other data sources one should use different values. WSO, MWO, and GONG data seems to have weaker field than MDI and SOLIS data. Therefore, a larger scaling factor should be used.