Space Weather Modeling Framework User Manual

Center for Space Environment Modeling
The University of Michigan

April 24, 2024

Contents

Chapter 1

Introduction

This document describes a working prototype of the Space Weather Modeling Framework (SWMF). The SWMF was developed to provide a flexible tool serving the Sun-Earth modeling community. In its current form the SWMF contains many domains extending from the surface of the Sun to the upper atmosphere of the Earth:

1. CZ – Convection Zone
2. EE – Eruptive Event generator
3. GM – Global Magnetosphere
4. IE – Ionosphere Electrodynamics
5. IH – Inner Heliosphere
6. IM – Inner Magnetosphere
7. OH – Outer Heliosphere
8. PC – Particle-in-Cell
9. PS – Plasma Sphere (under development)
10. PT – Particle Tracker
11. PW – Polar Wind
12. RB – Radiation Belts
13. SC – Solar Corona
14. SP – Solar Energetic Particles
15. UA – Upper Atmosphere

The core of the SWMF and most of the models are implemented in Fortran 90. Some models are written in Fortran 77, and others are written in C++. A few features of Fortran 2003 and 2008 are also used. The parallel communications use the Message Passing Interface (MPI) library. Some models can also take advantage of multi-threaded execution using the OpenMP library. The SWMF creates a single executable. Note, however, that the models in the SWMF can still be compiled into stand-alone executables. This means that the models preserve their individuality while being compatible with the SWMF.

1.1 Acknowledgments

The first version of the SWMF was developed at the Center for Space Environment Modeling (CSEM) of the University of Michigan under the NASA Earth Science Technology Office (ESTO) Computational Technologies (CT) Project (NASA CAN NCC5-614). The project was entitled as "A High-Performance Adaptive Simulation Framework for Space-Weather Modeling (SWMF)". The Project Director was Professor Tamas Gombosi, and the Co-Principal Investigators are Professors Quentin Stout and Kenneth Powell.

The first version of the SWMF and many of the physics components were developed at CSEM by the following individuals (in alphabetical order): David Chesney, Yue Deng, Darren DeZeeuw, Tamas Gombosi, Kenneth Hansen, Kevin Kane, Ward (Chip) Manchester, Robert Oehmke, Kenneth Powell, Aaron Ridley, Ilia Roussev, Quentin Stout, Igor Sokolov, Gábor Tóth and Ovsei Volberg.

The core design and code development was done by Gábor Tóth, Igor Sokolov and Ovsei Volberg:

- Component registration and layout was designed and implemented by Ovsei Volberg and Gábor Tóth.
- The session and time management support as well as the various configuration scripts and the parameter editor GUI were designed and implemented by Gábor Tóth.
- The SWMF coupling toolkit was developed by Igor Sokolov.
- The SWMF GUI was designed and implemented by Darren De Zeeuw.

The SWMF has undergone major improvements over the years. The current version has a fully automated test suite designed by Gábor Tóth. Empirical models were added in a systematic way. The shared libraries and utilities have been greatly extended. The SWMF has been coupled to the Earth System Modeling Framework (ESMF).

The current physics models were developed by the following research groups and individuals:

- The Eruptive Event generator (EE), Solar Corona (SC), Inner Heliosphere (IH), Outer Heliosphere (OH), and the Global Magnetosphere (GM) components are based on BATS-R-US MHD code developed at CSEM. BATS-R-US is a highly parallel up to 3-dimensional block-adaptive hydrodynamic and MHD code. Currently the main developers of BATS-R-US are Gabor Toth, Bart van der Holst, and Igor Sokolov. The current version of the Solar Corona model was developed by Bart van der Holst, Chip Manchester, Igor Sokolov with contributions from Cooper Downs, Ilia Roussev and Ofer Cohen. The Inner Heliosphere model was mostly developed by Chip Manchester. The Outer Heliosphere model was developed by Merav Opher and Gabor Toth. The Global Magnetosphere model was developed by Darren De Zeeuw, Gabor Toth, Aaron Ridley and many others. The physics based Eruptive Even Generator model is also based on BATS-R-US. It is developed by Fang Fang, Chip Manchester and Bart van der Holst. The EE model already works as a stand-alone code, and it will by coupled to other components in the SWMF in the future.
- The Ionospheric Electrodynamics (IE) model is the Ridley Ionosphere Model developed by Aaron Ridley, Darren De Zeeuw and Gabor Toth at CSEM. RIM is a 2-dimensional spherical electric potential solver. It has two versions. The Ridley_serial version can run on up to 2 processors, the RIM version is fully parallel with many new options, however it is still being developed, and it is not yet fully functional.
- The first Inner Magnetosphere (IM) model in the SWMF was the Rice Convection Model (RCM) developed Dick Wolf, Stan Sazykin and others at Rice University, also modified by Darren De Zeeuw at the University of Michigan. The current version of the model with oxygen and loss is named RCM2. The RCM code is 2-dimensional in space (plus one dimension for energy) and serial. There are 3 more IM models. The Comprehensive Inner Magnetosphere and Ionosphere (CIMI) model was developed by Mei-Ching Fok, Natasha Buzulukova and Alex Glocer at the NASA Goddard Space Flight Center. In addition to inner magnetosphere, CIMI also includes the high energy electrons of the radiation belt

The HEIDI model was developed by Mike Liemohn and Raluca Ilie at the University of Michigan. The RAM-SCB model was developed by Vania Jordanova, Sorin Zaharia and Dan Welling. The CIMI, HEIDI and RAM-SCB models are also 2 dimensional in space, but they resolve energy as well as pitch angle.

- There are three Particle-in-Cell (PC) models in the SWMF. All of them use an energy conserving semi-implicit PIC algorithm. The iPIC3D code was originally developed by Stefano Markidis and his group at KTH Sweden and Giovanni Lapenta and his group at KU Leuven, Belgium. The code was adapted, integrated into the SWMF and coupled with BATS-R-US by Lars Daldorff and Gabor Toth. iPIC3D solves for the electric and magnetic fields on a 3D Cartesian grid and the motion of electron and ion macroparticles. iPIC3D is a parallel code written in C++. Further improvements were made by Yuxi Chen and Gabor Toth to make it energy and charge conserving. The Adaptive Mesh Particle Solver (AMPS) code, was developed by Valeriy Tenishev. It was modified into a PIC solver by Valeriy Tenishev and Yinsi Shou with help from Yuxi Chen. The Flexible Exascale Kinetic Solver (FLEKS) was developed by Yuxi Chen. Both AMPS and FLEKS allow the PIC region to be adapted during the run.

- The Polar Wind (PW) component is the Polar Wind Outflow Model (PWOM) developed by Alex Glocer, Gabor Toth and Tamas Gombosi at the University of Michigan. This code solves the multifluid equations along multiple field lines and it is fully parallel.

- The Particle Tracker (PT) model is the Adaptive Mesh Particle Scheme (AMPS). The main developer of AMPS is Valeriy Tenishev. AMPS solves the motion and interaction of neutral and charged particles on an up to 3D adaptive grid. AMPS is a parallel code written in C++.

- The Radiation Belt Environment (RBE) model is developed by Meiching Fok and Alex Glocer at NASA Goddard. It is a spatially 2-dimensional code with extra two dimensions for pitch angle and energy. It is essentially the same as the energetic electron solver in the CIMI inner magnetosphere model.

- One of the Solar Energetic Particle (SP) models is the Kóta's SEP model by Joseph Kota at the University of Arizona. It solves the equations for the advection and acceleration of energetic particles along a magnetic field line in a 3D phase space using energy and pitch angle as the extra two dimensions. The other SP model is the Multiple Field Line Advection model for Particle Acceleration (MFLAMPA) by Igor Sokolov and Dmitry Borovikov. MFLAMPA also solves for energy distribution but assumes an isotropic pitch angle distribution.

- The Upper Atmosphere (UA) model is the Global Ionosphere-Thermosphere Model (GITM) developed by Aaron Ridley, Yue Deng, and Gabor Toth at CSEM. GITM is 3-dimensional spherical fully parallel hydrodynamic model with ions, neutrals, chemistry etc. The current version is the GITM2 model.

The following empirical models are available:

EEE The Gibbson-Low and the Titov-Demoulin flux rope models can be used to initiate CME-s. The breakout model is also available.

EGM The Tsyganenko 1996 and 2004 models.

EIE The Weimer 1996 and 2000 models, and many more empirical ionospheric electrodynamics models.

EUA The MSIS and IRI models for the upper atmospher and the ionosphere, respectively.

1.2 The SWMF in a Few Paragraphs

The SWMF is a software framework that allows integration of various models into a coherent system. The SWMF allows running and coupling any meaningful subset of the models together. The main applications of the SWMF are related to space physics and space weather, but it can be, and has been, used for other applications that the models allow. The SWMF contains a Control Module (CON), which is responsible for component registration, processor layout for each component as well as initialization, execution and coupling schedules. The SWMF contains templates (examples) and utilities for integrating and coupling models. A component is adapted from user-supplied physics codes, (for example BATS-R-US or RCM), by adding two relatively small units of code:

- A wrapper, which provides the control functions for CON, and
- A coupling interface to perform the data exchange with other components.

Both the wrapper and coupling interface are constructed from the building blocks provided by the framework.

An SWMF component is compiled into a separate library that resides in the directory `lib`, which is created as part of the installation process described later in this document. The executable `SWMF.exe` is created in the directory `bin`, which is created during the compilation. If a user does not want to use some particular component, this component should be configures to an empty version of the component.

The framework controls the initialization, execution, coupling and finalization of components. The execution is done in sessions. In each session the parameters of the framework and the components can be changed. The list of usable (non-empty) components and their processor layout (the array of processors used by that component) and all other input parameters are read from the `PARAM.in` file, which may contain further included parameter files. These parameters are read and broadcast by CON and the component specific parameters are sent to the components. The structure of the parameter file will be described in detail.

If two components reside on different sets of processing elements (PE-s) they can execute in an efficient concurrent manner. This is possible, because the coupling times (in terms of the simulation time or number of iterations) are known in advance. The components advance to the time of coupling and only the processors involved in the coupling need to communicate with each other. The components are also allowed to share some processing elements. The execution is sequential for the components with overlapping layouts. This can be useful when the execution time of the components vary a lot during the run, or when a component needs a lot of processors for memory storage, but it requires little CPU time. Of course, this still allows the individual components to execute in parallel. For steady state calculations the components are allowed to progress at different rates towards steady state. Each component can be called at different frequencies by the control module.

The coupling of the components is done through the SWMF. Couplings are done individually betwee one source and one target component. The frequency of couplings is determined by the input parameters. Two-way coupling is performed as two separate couplings with the source and target roles reversed. Each coupling only involves the processors used by either the source or target component. All other processors can keep working on their tasks. There are several utilities that facilitate component couplings. The most basic approach relies on plain MPI calls using the various functions of the SWMF that define the MPI communicators and information about the layout. While this approach is general, writing correct and efficient MPI code is not easy. The following libraries can make coupling much easier.

Passing scalars and small arrays between the source and target components can be done easily with the functions defined in CON_transfer_data. Here the coupling consists of

- an optional data reduction (MPI_SUM) on the source component
- single source processor to single target processor data transfer
- an optional data broadcast on the processors of the target component

For large amounts of data distributed over many processors, the CON_couple_points utility can be used with efficient N to M parallel communication. The point coupling algorithm consists of the following steps

- checking if the communication pattern needs to be updated
- if yes, then target model sends the list of point coordinates to the source model and the source model responds with the processor indexes.
- source processors interpolates and send their data to the appropriate target processors directly

Finally, the SWMF coupling toolkit also allows N to M coupling between components that describe their grids for the SWMF. Temporal interpolation is not directly supported by any of these utilities, but the components can do that internally.

1.3 System Requirements

In order to install and run the SMWF the following minimum system requirements apply.

- The SWMF runs only under the UNIX/Linux operating systems. This now includes Macintosh system 10.x because it is based on BSD UNIX. The SWMF does not run under any Microsoft Windows operating system.
- A FORTRAN 2008 compiler must be installed.
- A C/C++ compiler must be installed.
- The Perl interpreter must be installed.
- A version of the Message Passing Interface (MPI) library must be installed for parallel execution.
- Some models can use the OpenMP library. The Fortran and C++ compilers need to contain OpenMP to use this feature.
- Some models use HDF5 output. For these the parallel version of the HDF5 library has to be installed. The `share/Scripts/install_hdf5.sh` shell script can be used to do that.
- Some models use the SPICE library to determine position and orientation of various objects. To use these features, the SPICE library needs to be installed.
- In order to generate the documentation, LaTex has to be installed. The PDF generation requires the `dvips` and `ps2pdf` utilities.

One may be able to compile the code and do very small test runs on 1 or 2 processor machines. However, to do most physically meaningful runs the SWMF requires a parallel processor machine with a minimum of 8 processors and a minimum of 8GB of memory. Very large runs require many more processors.

The framework has been ported to many platforms using many different compilers and MPI libraries. The list of Fortran compilers includes Absoft, GNU gfortran, Intel ifort, Lahey, NAG nagfor, PGI pgf90, NVIDIA nvfortran, and IBM xlf90. The list of C compilers include GNU gcc, Apple clang, Intel icc, PGI pgcc, IBM mpxlc. The MPI libraries used so far include many versions of MPICH, MVAPICH and OpenMPI.

In addition to the above requirements, the SWMF output is typically visualized using IDL, Tecplot, python (SpacePy), VisIt, Paraview or yt. Other visualization packages may also be used, but the output file formats and scripts have been designed for these visualization softwares.

Chapter 2

Quick Start

2.1 A Brief Description of the SWMF Distribution

The distribution in the form of the compressed tar image includes the SWMF source code. The top level directory contains the following subdirectories:

- `CON` - the source code for the control module of the SWMF
- `Copyrights` - copyright files
- `ESMF` - the ESMF wrapper for the SWMF
- `CZ, EE, ... UA` - component directories
- `Param` - example input parameter files
- `Scripts` - shell and Perl scripts
- `doc` - the documentation directory
- `output` - reference test results for the SWMF tests
- `share` - shared scripts and source code
- `util` - utilities such as TIMING, NOMPI, empirical models, etc.

and the following files

- `README.md` - a short instruction on installation and usage
- `PARAM.XML` - description of CON parameters
- `Makefile` - the main makefile
- `Makefile.test` - the makefile containing the tests
- `Config.pl` - Perl script for (un)installation and configuration

2.2 General Hints

Getting help with scripts and the Makefile

Most of the Perl and shell scripts that are distributed with the SWMF provide help which can be accessed wit the `-h` flag. For example,

```
./Config.pl -h
```

will provide a detailed listing of the options and capabilities of the `Config.pl` script. In addition, you can find all the possible targets that can be built by typing

```
make help
```

Input commands: PARAM.XML

The input commands used in the `PARAM.in` and their meaning are described in the `PARAM.XML` files. For the SWMF itself it is found in

```
PARAM.XML
```

while the files for the physics components are found in the component's subdirectory. For example, the file for the GM/BATSRUS component can be found at

```
GM/BATSRUS/PARAM.XML
```

This file contains a complete list of all input commands for the component as well as the type, the allowed ranges and default values for each of the input parameters. Although the XML format makes the files a little hard to read, they are extremely useful. A typical usage is to cut and paste commands out of the PARAM.XML file into the PARAM.in file for a run.

An alternative approach is to use the web browser based parameter editor to edit the PARAM.in file for the SWMF (also for the stand-alone models that have PARAM.XML files). The editor GUI can be started as

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

This editor allows constructing `PARAM.in` files with pull down menus, shows the manual for the edited commands, and checks the correctness of the parameter file and highlights the errors. All this functionality is based on the PARAM.XML files.

Have the working directory in your path

In order to run executable files in the UNIX environment, either the current working directory has to be in the path or the filename has to be typed with the path. In UNIX the current working directory is represented by the period (.). For example

```
./Config.pl -s
```

will execute the Config.pl script if it is in your current directory. If the '.' is added to the path, for example with

```
set path = (${path} .)
```

then one can simply type

```
Config.pl -s
```

Setting the path is best done in the .cshrc or equivalent Unix shell customization file located in the user's home directory.

2.3 Installation

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 SWMF on your system and how
to create and access the documentation.  To learn more about the SWMF,
including how to compile and run the code, please consult the user
manual.  To install the SWMF and create the user manual please follow
the instructions below.

# Obtain SWMF

Get the full source code from GitHub/SWMFsoftware or the open-source code from
GitHub/MSTEM-QUDA.

The minimum requirement is the ‘SWMF‘ repository.

You may also need the open-source ‘SWMF_data‘ repository that contains
large data files and can be downloaded into your home directory (or into
the SWMF directory):

‘‘‘
cd
git clone https://github.com/SWMFsoftware/SWMF_data --depth=1
‘‘‘

For solar applications (solar corona, inner heliosphere, CMEs) the
‘SWMFSOLAR‘ repository can be useful. This is typically downloaded
into the SWMF directory, or to the (scratch/nobackup...) disk of
a supercomputer where the runs are performed:
‘‘‘
cd
git clone https://github.com/SWMFsoftware/SWMFSOLAR --depth=1
‘‘‘

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.

# Getting the open-source MSTEM-QUDA/SWMF:

Clone the SWMF from GitHub/MSTEM-QUDA

‘‘‘
cd {where_you_want_to_have_mstem-quda}
git clone https://github.com/MSTEM-QUDA/SWMF
‘‘‘

The rest of the repositories (share, util, BATSRUS ...)
will be cloned from GitHub/MSTEM-QUDA during the installation.
```

```
# Getting the full SWMF from GitHub/SWMFsoftware (requires access)

Read the
[Git instructions](http://herot.engin.umich.edu/~gtoth/SWMF/doc/Git_instructions.pdf)
about registering, passwordless access, mail notifications, and
using the [gitclone](https://github.com/SWMFsoftware/share/blob/master/Scripts/gitclone) script.

## Clone the SWMF repository

‘‘‘
cd {where_you_want_to_have_the_swmf}
gitclone SWMF
‘‘‘

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

# Install SWMF

Many machines used by UofM are already recognized by the
‘share/Scripts/Config.pl‘ script, which is called by all other ‘Config.pl‘
scripts in the SWMF.
For these platform/compiler combinations installation is very simple:
‘‘‘
./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, 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 type
‘‘‘
make PDF
‘‘‘
in the SWMF directory. The manuals will be in the doc/ directories.

## 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, 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/SWMF.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 on 2 CPU cores by default.
The results of the tests are summarized in ‘test_swmf.res‘.
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 test1 without MPI on a single core use
‘‘‘
make -j test1 MPIRUN=
‘‘‘
```

2.4 Platform specific information

The SWMF is tested every night on several computers. The test page can be found at

```
http://herot.engin.umich.edu/~gtoth
```

This page shows which tests passed and which tests failed. It also shows the combination of compilers and libraries used on various machines (click on the Explanations link at the top).

If you are running on one of the tested machines, you can use the `module load XYZ` command to load the appropriate modules. It is best to do this in the `.cshrc` or equivalent customization file in the home directory. Use

```
module avail
```

to see the list of all available modules.

2.5 Building and Running an Executable

At compile time, the user can select the physics models to be compiled. All components with an `Empty` model version will be unavailable for use at run time. The physics components can be selected with the `-v` flag of the Config.pl script. For example typing

```
Config.pl -v=Empty,SC/BATSRUS,IH/BATSRUS,SP/Kota
```

will select BATSRUS for the SC and IH components and Kóta's model for the SP componens. All the other components are set to the `Empty` model versions that contain empty subroutines for compilation, but cannot be used. Note that `Empty` always has to be listed first. The default configuration uses the Empty version for all components.

The grid size of several components can also be set with the `-g` flag of the `Config.pl` script. For example the

```
Config.pl -g=GM:8,8,8
```

command sets the block size for the GM component to $8 \times 8 \times 8$ cells. The main SWMF Config.pl script actually runs the individual Config.pl scripts in the component versions. These scripts can be run directly, For example try

```
cd GM/BATSRUS
Config.pl -show
```

Compilation rules, library definitions, debugging flags, and optimization level are stored in `Makefile.conf`. This file is created during installation of the SWMF and contains default settings for production runs. The compiler flags can be modified with

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

to debug the code with 0 optimization level, and

```
Config.pl -nodebug -O4
```

to run the code at maximum optimization level and without the debugging flags. Before compiling the SWMF, it is always a good idea to check its configuration with

```
Config.pl -show
```

To build the executable **bin/SWMF.exe**, type:

```
make -j
```

The -j flag allows parallel compilation, which can reduce the compilation time considerably. Depending on the configuration, the compiler settings and the machine, compiling the code can take several minutes. In addition, the PostIDL.exe post processing code can be compiled with

```
make PIDL
```

The `SWMF.exe` executable should be run in a sub-directory, since a large number of files are created in each run. To create this directory use the command:

```
make rundir
```

This command creates a directory called `run`. You can either leave this directory as named, or `mv` it to a different name. It is best to leave it in the same SWMF directory, since keeping track of the code version associated with each run is quite important. On some platforms, however, the runs should be done on a parallel file system (often called scratch or nobackup), while the source code is better kept in the home directory. In this case move the run directory to the scratch disk and create a symbolic link to it, for example

```
mv rundir /p/scratch/MYNAME/SWMF/run_halloween2
ln -s /p/scratch/MYNAME/SWMF/run_halloween2 .
```

The `run` directory will contain links to the codes which were created in the previous step as well as sub-directories where input and output of the different components will reside. On some systems the compute nodes cannot access symbolic links across different file systems. In this case the executable should be copied instead of linked, so in our example the following commands should be done every time after the SWMF has been (re)compiled:

```
rm -f run_halloween2/SWMF.exe
cp bin/SWMF.exe run_halloween2/
```

To run the SWMF change directory into the `run` directory (or the symbolic link to it):

```
cd run_halloween2
```

In order to run the SWMF you must have the input file PARAM.in. The required #COMPONENTMAP command in the PARAM.in file defines the processor layout for the components involved in the future run. In addition, the PARAM.in file contains the detailed commands for controlling what you want the code to do during the run. The `Param` directory contains many example input files. Many of these are used by the nightly test suite.

An example processor map to run the executable with five components on 16 processors is:

```
ID  First Last Stride
#COMPONENTMAP
GM  0  4  1       CompMap
IE  5  6  1       CompMap
IH  7 10  1       CompMap
IM 11 11  1       CompMap
UA 12 15  1       CompMap
```

The syntax is simple. It must start with the directive #COMPONENTMAP followed by the processor layout of active components. Each line specifies the label for component, i.e. IE, GM and etc., its first and last processors, all relative to the world communicator, and the stride. Thus GM will run on 5 processors from 0 to 4, and IM will run on only 1 processor, the processor 11. If stride is not equal to 1, the processors for the component will not be neighboring processors.

It is strongly recommended to check the validity of the `PARAM.in` file before running the code. If the code will be run on 16 processors, type

```
Scripts/TestParam.pl -n=16 run_halloween2/PARAM.in
```

in the main SWMF directory. The Perl script reports inconsistencies and errors. If no errors are found, the script finishes silently. Now you are ready to run the executable through submitting a batch job or, if it is possible on your computer, you can run the code interactively. For example, to run the SWMF interactively:

```
cd run_halloween2
mpiexec -n 16 SWMF.exe | tee runlog
```

The `| tee runlog` splits the output and send it to the screen as well as into the `runlog` file. The SWMF provides example job scripts for several machines. These job script files are found in the

```
share/JobScripts/
```

directory. If the name of some of the job script files matches the name of the machine returned by the `hostname` command (numbers at the end of the machine name are ignored, so pfe23 matches job.pfe), the job script is copied into the `run` directory when it is created. These job scripts serve as a starting point only, they must be customized before they can be used for submitting a job.

To recompile the executable with different compiler settings you have to use the command

```
make clean
```

before recompiling the executables. It is possible to recompile only a component or just one subdirectory if the `make clean` command is issued in the appropriate directory.

2.6 Post-Processing the Output Files

Several components produce output files (plot files) that require some post-processing before they can be visualized. The post-processing collects data written out by different processors, and it can also process and transform the data.

The PostProc.pl script greatly simplifies the post-processing and it also helps to collect the run results in a well contained directory tree. The script can also be used to do post-processing while the code is running. Usually the processed output files are much smaller than the raw output file, so post-processing during the run can limit the amount of disk space used by the raw data files. It also avoids the need to wait for a long time for the post-processing after the run is done.

The `PostProc.pl` script is copied into the run directory and it should be executed from the run directory. To demonstrate the use of the script, here are a few simple examples. After or during a run, you may simply type

```
cd run_halloween2
./PostProc.pl
```

to post-process the available output files. The series of individual IDL plot files can be concatenated into single movie files with

```
./PostProc.pl -M
```

Repeat the post-processing every 360 seconds during the run, and gzip large ASCII files:

```
./PostProc.pl -r=360 -g >& PostProc.log &
```

After the run is finished, create IDL movie files and concatenate various log and satellite files (for restarted runs), and create a directory tree `RESULTS/NewRun` with the output of all the components, the input parameter file, a restart directory tree (if restart information was saved), and the `runlog` file (if present):

```
./PostProc.pl -M -cat -o RESULTS/NewRun
```

The `RESULTS/NewRun` directory will contain the PARAM.in file, the runlog file (the standard output should be piped into that file), the restart directory named `RESULTS/NewRun/RESTART/`, and the output files for each component in a subdirectory named accordingly (eg. `RESULTS/NewRun/GM/`). The output directories of the components (e.g. `GM/IO2/`) will be empty after this.

To see all the options of the script, including parallel processing and syncing results to a remote computer, type

```
./PostProc.pl -h
```

2.7 Restarting a Run

There are several reasons for restarting a run. A run may fail due to a run time error, due to hardware failure, due to software failure (e.g. the machine crashes) or because the queue limits are exceeded. In such a case the run can be continued from the last saved state of SWMF.

It is also possible that one builds up a complex simulation from multiple runs. For example the first run creates a steady state for the SC component. The second run includes both the SC and IH components and it restarts from the results of the first run and creates a steady state for both components. A third run may restart from this solution and include the GM component, etc.

The restart files are saved at the frequency determined in the PARAM.in file. Normally the restart files are saved into the output restart directories of the individual components and subsequent saves overwrite the previous ones (to reduce the required disk space). A restart requires the modification of the PARAM.in file: one needs to include the restart file for the control module of SWMF as well as ask for restart by all the components.

The Restart.pl script simplifies the work of the restart in several ways:

1. The SWMF restart file and the individual output restart directories of the components are collected into a single directory tree, the **restart tree**.

2. The default input restart file of SWMF and the default input directories of the components can be linked to an existing restart tree.

3. The script can run continuously in the background and create multiple restart trees while SWMF is running.

4. The script does extensive checking of the consistency of the restart files.

The Restart.pl script is copied into the run directory and it should be executed in the run directory. Note that the PARAM.in file is not modified by the script: it has to be modified with an editor as needed.

To demonstrate the use of the script, here are a few simple examples. After a successful or failed run which should be continued, simply type

```
cd run_halloween2
./Restart.pl
```

to create a restart tree from the final output and to link to the tree for the next run. The default name of the restart tree is based on the simulation time for time accurate runs, or the time step for non-time accurate runs. But you can also specify a name explicitly, for example

```
./Restart.pl RESTART_SC_steady_state
```

If you wish to continue the run in another run directory, or on another machine, transfer the restart tree as a whole into the new run directory and type

```
./Restart.pl -i=RESTART_SC_steady_state
```

where the `-i` stands for "input only", i.e. the script links to the tree, but it does not attempt to create the restart tree.

To save multiple restart trees repeatedly at an hourly frequency of wall clock time while the SWMF is running, type

```
./Restart.pl -r=3600 &
```

To see all the options of the script type

```
./Restart.pl -h
```

2.8 What's next?

Hopefully this section has guided you through installing the SWMF and given you a basic knowledge of how to run it. However it has probably also convinced you that the SWMF is quite a complex tool and that there are many more things for you to learn. So, what next?

We suggest that you read all of chapter 3, which outlines the basic features of the SWMF as well as some things you really must know in order to use the SWMF. Once you have done this you are ready to experiment. Chapter **??** gives several examples which are intended to make you familiar with the use of the SWMF. We suggest that you try them!

Chapter 3

The Basics

3.1 Configuration of SWMF

Configuration refers to several different ways of controlling how the SWMF is compiled and run. The most obvious is the setting of compiler flags specific to the machine and version of FORTRAN compiler. The other methods refer to ways in which different physics components are chosen to participate in or not participate in a run. Inclusion of components can be controlled using one of several methods:

- The source code can modified so that all references to a subset of the components is removed. This method uses the Scripts/Configure.pl script. In a similar way, some physics components can be individually configured.
- The user may select which version of a physics component, including the Empty version, should be compiled. This is controlled using the Config.pl script.
- When submitting a run, a subset of the non-empty components can be registered to participate in the run with the #COMPONENTMAP command in the PARAM.in file.
- Registered components can be turned off and on with the #COMPONENT command in the PARAM.in file.

Each of these options have their useful application.

Finally, each physics component may have some settings which need to (or can) be individually configured, such as selecting user routines for the IH/BATSRUS or GM/BATSRUS components.

3.1.1 Scripts/Configure.pl

The Scripts/Configure.pl script can build a new software package which contains only a subset of the components. It is a simple interface for the general share/Scripts/Configure.pl script. The configuration can remove a whole component directory and all references to the component in the source code, in the scripts and the Makefiles. This type of configuration results in a smaller software package. The main use of this type of configuration is to distribute a part of SWMF to users. For example one can create a software distribution which includes GM, IE and UA only by typing

```
Scripts/Configure.pl -on=GM,IE,UA -off=SC,IH,SP,IM,PW,RB
```

The configured package will be in the Build directory. Type

```
Scripts/Configure.pl -h
```

to get complete usage information or read about this script in the reference manual.

3.1.2 Selecting physics models with Config.pl

The physics models (component versions) reside in the component directories CZ, EE, GM, IE, IH, IM, OH, RB, PS, PT, PW, SC, SP and UA. Most components have only one working version and one empty version. The empty version consists of a single wrapper file, which contains empty subroutines required by CON_wrapper and the couplers. These empty subroutines are needed for the compilation of the code, and they also show the interface of the working versions.

The appropriate version can be selected with the `-v` flag of the `Config.pl` script, which edits the Makefile.def file. For example

```
Config.pl -v=GM/BATSRUS,IM/RCM2,IE/Ridley_serial
```

selects the BATSRUS, RCM2 and Ridley_serial models for the GM, IM and IE components, respectively. To see the current selection and the available models for all the components type

```
Config.pl -l
```

The first column shows the currently selected models, the rest are the available alternatives.

If a physics component is not needed for a particular run, an Empty version of the component can be compiled. Selecting the Empty version for unused components reduces compilation time and memory usage during run time. It may also improve performance slightly. This is achieved with the `-v` flag of the Config.pl script. For example the Empty UA component can be selected with

```
Config.pl -v=UA/Empty
```

It is also possible to select the Empty version for all components with a few exceptions. For example

```
Config.pl -v=Empty,GM/BATSRUS,IE/Ridley_serial
```

will select the Empty version for all components except for GM and IE. Note that the 'Empty' item has to be the first one.

3.1.3 Clone Components

The EE/BATSRUS, IH/BATSRUS, OH/BATSRUS and SC/BATSRUS models are special, since they use the same source code as GM/BATSRUS, which is stored in the CVS repository. We call the other BATSRUS models **clones** of the GM/BATSRUS code. The source code of the clone models is copied over from the original files and then all modules, external subroutines and functions are renamed. For example ModMain.f90 is renamed to IH_ModMain.f90 in IH/BATSRUS. These steps are performed automatically when the clone model is selected for the first time, for example by typing

```
Config.pl -v=IH/BATSRUS
```

Once the source code is copied and renamed, the clone models work just like any model. They can be configured, compiled, and used in runs.

It is important to realize that code development is always done in the original source code, i.e. in GM/BATSRUS and in IH/BATSRUS/srcInterface/IH_wrapper.f90. If the source code of the clones should be refreshed, for example after an update from the CVS respository, type

```
make cleanclones
Config.pl
```

and the source code will be copied and renamed for the selected clones. The source code of the clones is removed fully when the SWMF is uninstalled with the

```
Config.pl -uninstall
```

command.

3.1.4 Registering components with the #COMPONENTMAP command

The components used in particular run has to be listed (registered) with the #COMPONENTMAP command in the PARAM.in file. Note that empty component versions cannot be registered at all. Component registration allows to run the same executable with different subsets of the components. For example the GM and IE components can be selected with the following command:

```
ID Proc0 ProcEnd Stride nThread
#COMPONENTMAP
IE     0       1      1
GM     2      -1      1
```

The columns contain the component ID, the index of the first (root) processor for the component, the last processor, the stride, and the optional number of threads, respectively. Negative values for the root and last processor ranks are taken as counting backwards from the total number of processors nProc. Negative values for the stride and number of threads are interpreted as the maximum number of threads (MaxThread is defined by the OMP_NUM_THREADS environment variable) divided by the absolute value of Stride or nThread, respectively. This allows the same layout to be used on different nodes with different number of cores per node. The default number of threads is 1 and the maximum value is MaxThread.

The example above has IE running on the first 2 cores, and GM running on the rest of the cores. Changing the command to

```
ID Proc0 ProcEnd Stride nThread
#COMPONENTMAP
GM     0      -1     -1      -1
```

will still use the same executable, but will not allow the IE physics component to participate in the run, on the other hand, GM can possibly use multithreading with one thread per core.

3.1.5 Switching models on and off with PARAM.in

Registered components can be switched on and off during a run with the #COMPONENT command in the PARAM.in file. This approach allows the component to be switched on in a later 'session' of the run. For example, in the first session only GM is running, while in the second session it is coupled to IE. In this example the IE component can be switched off with the

```
#COMPONENT
IE              NameComp
F               UseComp
```

in the first session and it can be switched on with the

```
#COMPONENT
IE              NameComp
T               UseComp
```

command in the second session.

3.1.6 Setting compiler flags

The debugging flags can be switched on and off with

```
Config.pl -debug
```

and

```
Config.pl -nodebug
```

respectively. The maximum optimization level can be set to -O2 with

```
Config.pl -O2
```

The minimum level is 0, the maximum is 5. Note that not all compilers support level 5 optimization. As already mentioned, the code needs to be cleaned and recompiled after changing the compiler flags:

```
make clean
make -j
```

Note that not all the components take into account the selected compiler flags. For example the IM/RCM2 component has to be compiled with the -save (or similar) flag, thus it uses the flags defined in the `CFLAGS` variable. Also some of the compilers produce incorrect code if they compile certain source files with high optimization level. Such exceptions are described in the

```
Makefile.RULES.all
```

files in the source code directories. The content of this file is processed by `Config.pl` into `Makefile.RULES` (according to the selected compiler and other parameters), which is then included into the main Makefile of the source directory.

3.1.7 Configuration of individual components

Some of the components can be configured individually. The `GM/BATSRUS` code, for example, can be configured to use specific equation and user modules. For example

```
cd GM/BATSRUS
Config.pl -e=MhdIonsPe
```

will select the equation module for multiple ion fluids and separate electron pressure. The same can be done with the `Config.pl` script in the main SWMF directory

```
Config.pl -o=GM:e=MhdIonsPe
```

To set the grid for GITM, for example,

```
Config.pl -o=UA:g=36,36,50,16
```

will set the blocks size to $36 \times 36 \times 50$ and the number of blocks to 16 for the UA/GITM2 component. This command runs the `Config.pl` script of the selected UA component. On machines with limited memory it is especially important to set the number of blocks correctly.

Of course, the SWMF code has to be recompiled after any of these changes with

```
make -j
```

Note that in this case there is no need to type 'make clean', because the `make` command knows which files need to be recompiled.

3.1.8 Using stubs for all components

It is possible to compile and run the SWMF without the physics components but with place holders (stubs) for them that mimic their behavior. This can be used as a test tool for the CON component, but it may also serve as an inexpensive testbed for getting the optimal layout and coupling schedule for a simulation. To configure SWMF with stub components, select the Empty version for all physics components (with Config.pl -v=...) and edit the `Makefile.def` file to contain

```
#INT_VERSION = Interface
INT_VERSION = Stubs
```

for the interface so that the real interface in `CON/Interface` is replaced with `CON/Stubs`. The resulting executable will run CON with the stubs for the physics components. For the stubs one can specify the time step size in terms of simulation time and the CPU time needed for the time step. The stub components communicate at the coupling time, so the PE-s need to synchronize, but (at least in the current implementation) there is no net time taken for the coupling itself.

The stub components help development of the SWMF core, but it also allows an efficient optimization of the component layout and coupling schedules for an actual run, where the physical time steps and the CPU time needed by the components is approximately known. In the test runs with the Stubs, one can reduce the CPU times by a fixed factor, so it takes less CPU time to see the efficiency of the SWMF for a given layout and coupling scheme.

An alternative way to test performance with different configurations is to use the Scripts/Performance.pl script. See the help message of the script for information on usage.

3.2 PARAM.in

The input parameters for the SWMF are read from the `PARAM.in` file which must be located in the run directory. This file controls the SWMF and its components. There are many include files in the `Param` directory. These can be included into the `PARAM.in` files, or they can serve as examples.

In the PARAM.in file, the parameters specific to a component are given between the #BEGIN_COMP ID and #END_COMP ID commands, where the ID is the two character identifier of the component. For example the GM parameters are enclosed between the

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

commands. We refer to the lines starting with a # character as commands. For example 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 PARAM.in file indicates the end of the run.

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 syntax and the content of the input parameter files is defined in the PARAM.XML files. The commands controlling the whole SWMF are described in the main directory in the

```
PARAM.XML
```

file. The component parameters are described by the PARAM.XML file in the component version directory. For example the input parameters for the GM/BATSRUS component are described in

```
GM/BATSRUS/PARAM.XML
```

These files can be read (and edited) in a normal editor. The same files are used to produce much of this manual with the aid of the `share/Scripts/XmlToTex.pl` script. The `Scripts/TestParam.pl` script also uses these files to check the PARAM.in file. Copying small segments of the `PARAM.XML` files into `PARAM.in` can speed up the creation or modification of a parameter file.

3.2.1 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.

3.2.2 Commands, Parameters, and Comments

As can be seen from the above examples, the 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. In addition, real numbers can be entered as fractions (5/3 for example). 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...

#DESCRIPTION
My first run            StringDescription (3 spaces or TAB before the comment)

#STOP
-1.                     tSimulationMax
100                     MaxIteration

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

#TIMEACCURATE
T                       DoTimeAccurate

#STOP
10.0                    tSimulationMax
```

```
-1                      MaxIteration

#BEGIN_COMP IH

#GAMMA
5/3                     Gamma

#END_COMP IH
```

3.2.3 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 use only a subset of the components in the first session, and add more components in the later session. Or one can obtain a coarse steady state solution on a coarse grid with a component in one session, and improve on the solution with a finer grid in the next session. Or one can switch from steady state mode to time accurate mode. The SWMF 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.

3.2.4 The Order of Commands

In essence, the order of parameter commands within a session is arbitrary, but there are some important restrictions. We should note that the order of the parameters following the command is not arbitrary and must exactly match what the code requires. Here we restrict ourselves to the restrictions on the commands read by the control module of SWMF. There may be (and are) restrictions for the commands read by the components, but those are described in the documentation of the components.

The only strict restriction on the SWMF commands is related to the 'planet' commands. The default values of the planet parameters are defined by the #PLANET command. For example the parameters of Earth can be selected with the

```
#PLANET
Earth            NamePlanet
```

command. The true parameters of Earth can be modified or simplified with a number of other commands which **must occur after the #PLANET command**. These commands are (without showing their parameters)

```
#IDEALAXES
#ROTATIONAXIS
#MAGNETICAXIS
#MAGNETICCENTER
#ROTATION
#DIPOLE
```

Other than this strict rule, it makes sense to follow a 'natural' order of commands. This will help in interpreting, maintaining and reusing parameter files.

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

```
#ECHO
T                DoEcho
```

If the code starts from restart files, it usually reads in a file which was saved by SWMF. The default name of the saved file is RESTART.out and it is written into the run directory. It should be renamed, for example to RESTART.in, so that it does not get overwritten during the run. It can be included as

```
#INCLUDE
RESTART.in
```

The SWMF will read the following commands (the parameter values are examples only) from the included file:

```
#DESCRIPTION
Create startup for GM-IM-IE-UA from GM steady state.

#PLANET
EARTH                        NamePlanet

#STARTTIME
    1998                     iYear
       5                     iMonth
       1                     iDay
       0                     iHour
       0                     iMinute
       0                     iSecond
 0.000000000000              FracSecond

#NSTEP
    4000                     nStep

#TIMESIMULATION
```

```
 0.00000000E+00              tSimulation

#PRECISION
8                              nByteReal
```

The #PLANET command defines the selected planet. The #STARTTIME command defines the starting date and time of the whole simulation. The current simulation time (which is relative to the starting date and time) and the step number are given by the #TIMESIMULATION and #NSTEP commands. Finally the #VERSION and #PRECISION commands check the consistency of the current version and real precision with the run which is being continued. For sake of convenience, the #IDEALAXES, #ROTATEHGR and #ROTATEHGI commands are also saved into the restart file if they were set in the run.

As it was explained above, all modifications of the planet parameters should follow the #PLANET command, i.e. they should be after the #INCLUDE RESTART.in command. In case the description is changed it should also follow, e.g.

```
#INCLUDE
RESTART.in

#DESCRIPTION
We continue the run for another 2 hours
```

When the run starts from scratch, the PARAM.in file should start similarly with the

```
#DESCRIPTION
This is the start up run

#PLANET
SATURN

#STARTTIME
    2004                    iYear
       8                    iMonth
      15                    iDay
       1                    iHour
      25                    iMinute
       0                    iSecond
 0.000000000000             FracSecond
```

commands (the parameters are examples only). These commands are typically followed by the planet parameter modifying commands, if any, and setting time accurate mode (if changed from default true to false or relative to the previous session). For example:

```
! Align the rotation and magnetic axes with Z_GSE
#IDEALAXES

#TIMEACCURATE
F                           DoTimeAccurate
```

All the commands which are written into the RESTART.out file and all the planet modifying commands can only occur in the first session. These commands contain parameters which should not change during a run. In the PARAM.XML file these commands are marked with an `if="$_IsFirstSession"` conditional. If any of these parameters are attempted to be changed in later sessions, a warning is printed on the screen and the code stops running (except when the code is in non-strict mode).

Most command parameters have sensible default values. These are described in the PARAM.XML files, and in chapter 4 (which was produced from them). The `PARAM.XML` file also defines which commands are required with the `required="T"` attribute of the `<command...>` tag. For the control module the only required command in every session is the #STOP command (or this can be replaced with the #ENDTIME command in the last session), which defines the final time step in steady state mode or the final time of the session in time accurate mode.

3.2.5 Iterations, Time Steps and Time

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 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 seconds 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 explicitly repeat the command in the first time accurate session with the time frequency set.

The purpose of this subsection is to try to help the user understand the difference between the iteration number used for stopping the code and the time step which is used to define the frequencies of various actions. After using BATS-R-US over several years, it is clear to the authors that this distinction is useful and the most reasonable implementation. The SWMF has inherited these features from the BATS-R-US code.

We begin by defining several different quantities and the variables that represent them in the code. The variable `nIteration`, 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 `nStep` 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 `tSimulation` 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.

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   nStep    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. 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. Note that a restart file will also be written at time 1000.0 seconds since this is the end of a run.

In the next example we want to restart from 1000.0 seconds and continue with a time accurate run.

```
                                          Restart Files Written at:
==SESSION 1                          Session   nITER   nStep    time_simulation
                                     --------  ------  -------  --------------
#INCLUDE                                  1       0     2000          1000.0
RESTART.in

#TIMEACCURATE
T            DoTimeAccurate

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

#STOP
-1           MaxIteration
1400.0       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 400.0 seconds more simulation, the original 1000.0 seconds must be taken into account. The final output at 2000.0 seconds is written because the run ended.

Throughout this subsection, we have used the frequency of writing restart files as an example. The frequencies of coupling components and checking stop files work similarly. In the SWMF, and potentially in any of the components, the frequencies are handled by the general

```
share/Library/src/ModFreq
```

module which is described in the reference manual.

3.3 Execution and Coupling Control

The control module of SWMF controls the execution and coupling of components. The control module is controlled by the user through the input parameter file PARAM.in. Defining the most efficient component layout, execution and coupling control is not an obvious task. In the current version of SWMF the processor layout of the components is static. This restriction is somewhat mitigated by the possibility of restart, which allows to change the processor layout from one run to another.

3.3.1 Processor Layout

Within one run the layout is determined by the #COMPONENTMAP command in the PARAM.in file. The command is documented in the PARAM.XML file. Here we provide several examples which will help to develop a sense of using optimal layouts. An optimal layout is one that maximizes the use of all processors and does not leave processors with nothing do while waiting for other processors to finish their work.

First of all we have to define the processor rank: it is a number ranging from 0 to $N-1$, where N is the total number of processors in the run. A component can run on a subset of the processors, which is defined by the rank of the first (root) processor, the rank of the last processor, and the stride. For example if the root processor has rank 4, the last processor has rank 8, and the stride is 2 than the component will run on 3 processors with ranks 4, 6 and 8.

One component

In the simplest case a single component, say the Global Magnetosphere (GM) is running. The layout should be the following

```
ID   Proc0 ProcEnd Stride
#COMPONENTMAP
GM   0     -1    1
```

Here the -1 is interpreted as the rank of the last processor, which is $N-1$ if the SWMF is running on N processors.

One serial and one parallel component

When two components are used, their layouts may or may not overlap. An example for overlapping the layouts of the GM and the Inner Magnetosphere (IM) components is

```
ID   proc0 last stride
#COMPONENTMAP
IM    0       0    1
GM    0      -1    1
```

When the component layouts overlap, the two components can run sequentially only. Since IM is using a single processor only (because it is not a parallel code), all the other processors will be idling while IM is running. This can be rather inefficient, especially if the CPU time required by IM is not negligible. A more efficient execution can be achieved with a non-overlapping layout:

```
ID   proc0 last stride
#COMPONENTMAP
IM    0       0    1
GM    1      -1    1
```

Note that this layout file will work for any number of processors from 2 and up.

Two parallel components with different speeds

It is not always possible, or even efficient to use non-overlapping layouts. For example both the SC and IH components require a lot of memory, but the IH component runs much faster (say 100 times faster) in terms of cpu time than the SC component (this is due to the larger cells and smaller wave speeds in IH). If we tried to use concurrent execution on 101 processors, SC should run on 100 and IH on 1 processors to get good load balancing. However the IH component needs much more memory than available on a single processor. It is therefore not possible to use a non-overlapping layout for SC and IH on a reasonable number of processors.

Fortunately both the Solar Corona (SC) and Inner Heliosphere (IH) components are modeled by BATS-R-US, which is a highly parallel code with good scaling. The following layout can be optimal:

```
ID   proc0 last stride
#COMPONENTMAP
IH    0    -1    1
SC    0    -1    1
```

Although IH and SC will execute sequentially, they both use all the available CPU-s, so no CPU is left waiting for the others.

Two parallel components with similar speeds

If two parallel components need about the same CPU time/real time on the same number of processors, the optimal layout can be

```
ID   proc0 last stride
#COMPONENTMAP
GM    0     -1    2
SC    1     -1    2
```

Here GM is running on the processors with even rank, while SC is running on the processors with odd ranks. By using the processor stride, this layout works on an arbitrary number of processors.

When more serial and parallel codes are executing together, finding the optimal layout may not be trivial. It may take some experimentation to see which component is running slower or faster, how much time is spent on coupling two components, etc. It may be a good idea to test the components separately or in smaller groups to see how fast they can execute.

A complex example with four components

Here is an example with 4 components: the Ionospheric Electrodynamics (IE) component can run on 2 processors and runs about 3 times faster than real time. The serial Inner Magnetosphere (IM) component runs even faster, on the other hand the coupling of GM and IM is rather computationally expensive. The Upper Atmosphere (UA) component can run on up to 32 processors, and it runs twice as fast as real time. The Global Magnetosphere model (GM) needs at least 32 processor to run faster than real time. If we have a lot of CPU-s, we may simply create a non-overlapping layout. Since GM has no restriction on the number of processors, it can be the last component in the map

```
ID   proc0 last stride
#COMPONENTMAP
IM    0       0    1
IE    0       1    1
UA    2      33    1
GM   34     999    1
```

This layout will be optimal in terms of speed for a large (more than 100) number of PE-s, and actually the maximum speed is going to be limited by the components which do not scale. On a more modest number of PE-s one can try to overlap UA and GM:

```
ID   proc0 last stride
#COMPONENTMAP
IM    0       0    1
IE    1       2    1
UA    0      31    1
GM    3     999    1
```

Using OpenMP threads

Some of the models, such as BATS-R-US, can use OpenMP threads in addition to the MPI paralelization. Typically one should run one OpenMP thread on each core, and the number of MPI processes should be 1 or 2 (or possibly more) for each node. The most efficient arrangement depends on the hardware architecture and the model. The number of maximum threads MaxThread is set by the environment variable OMP_NUM_THREADS. Typically one wants to use nThread=MaxThread threads for the components that can use OpenMP. This can be easily achieved by setting the stried and the number of threads in the last (optional) column to -1:

```
ID   proc0 last stride nthread
#COMPONENTMAP
GM    0      -1    -1      -1
```

For example, if the node has 56 cores split to two independent slots, the optimal setting is likely to be OMP_NUM_THREADS=28. In this case both stride and nthread will be 28.

If OMP_NUM_THREADS is not in advance, it is best to set the root of the multithreaded component to proc0=0, so that the stride is properly aligned with the cores of the nodes. This means that other components that can only use a fixed number of processors should be put to the last processors, for example

```
ID   proc0 last stride nthread
#COMPONENTMAP
GM    0      -3    -1      -1
IE   -2      -1     1
```

In this layout GM is running with multiple threads on cores 0 to $N-3$, while IE is using cores $N-2$ and $N-1$.

3.3.2 Steady State vs. Time Accurate Execution

The SWMF can run both in time accurate (default) and steady state mode. This sounds surprising first, since many of the components can run in time accurate mode only. Nevertheless, the SWMF can improve the convergence towards a steady state by allowing the different components to run at different speeds in terms of the physical time. In BATS-R-US the same idea is used on a much smaller scale: local time stepping allows each cell to converge towards steady state as fast as possible, limited only by the local numerical stability limit.

Steady state session

The steady state mode should be signaled with the

```
#TIMEACCURATE
F                DoTimeAccurate
```

command, usually placed somewhere at the beginning of the session. Since the SWMF runs in time accurate mode by default, this command is required in the first steady state session of the run.

When SWMF runs in steady state mode, the SWMF time is not advanced and tSimulation usually keeps its default initial value, which is zero. The components may or may not advance their own internal times. The execution is controlled by the step number `nStep`, which goes from its initial value to the final step allowed by the MaxIteration parameter of the #STOP command. The components are called at the frequency defined by the #CYCLE command. For example

```
#CYCLE
GM                NameComp
1                 DnRun

#CYCLE
IM                NameComp
2                 DnRun
```

means that IM runs in every second time step of the SWMF. By defining the DnRun parameter for all the components, an arbitrary relative calling frequency can be obtained, which can optimize the global convergence rate to steady state. The default frequency is DnRun=1, i.e. the component is run in every SWMF time step.

The relative frequency can be important for numerical stability too. When GM and IM are to be relaxed to a steady state, the GM/BATSRUS code is running in local time stepping mode, while IM/RCM runs in time accurate mode internally. Since GM and IM are coupled both ways, an instability can occur if both GM and IM are run every time step, because the GM physical time step is very small, and the MHD solution cannot relax while being continuously pushed by the IM coupling. This unphysical instability can be avoided by calling the IM component less frequently.

The coupling frequencies should be set to be optimal for reaching the steady state. If the components are coupled too frequently, a lot of CPU time is spent on the couplings. If they are coupled very infrequently, the solution may become oscillatory instead of relaxing into a (quasi-)steady state solution. For example we used the

```
#COUPLE2
IM                      NameComp1
GM                      NameComp2
10                      DnCouple
-1.                     DtCouple
```

command to couple the GM and IM components in both directions in every 10-th SWMF iteration. Note that according to the above #CYCLE commands, GM and IM do 10 and 5 steps between two couplings, respectively. GM/BATSRUS uses 10 local time steps, while IM advances by 5 five-second time steps.

Another example is the relaxation of SC and IH components. Under usual conditions the solar wind is supersonic at the inner boundary of the IH component, thus the steady state SC solution can be obtained first, and then IH can converge to a steady state using the SC solution as the inner boundary condition. In this second stage SC does not need to run (assuming that it has reached a good steady state solution), it is only needed for providing the inner boundary condition for IH. This can be achieved by

```
! No need to run SC too often, it is already in steady state
#CYCLE
SC                      NameComp
1000                    DnRun

! No need to couple SC to IH too often
#COUPLE1
```

```
SC                      NameSource
IH                      NameTarget
1000                    DnCouple
-1.0                    DtCouple
```

Since SC and IH are always coupled at the beginning of the session, further couplings are not necessary.

Time accurate session

The SWMF runs in time accurate mode by default. The

```
#TIMEACCURATE
T                       DoTimeAccurate
```

command is only needed in a time accurate session following a steady state session. In time accurate mode the components advance in time at approximately the same rate. The component times are only synchronized when necessary, i.e. when they are coupled, when restart files are written, or at the end of session and execution. Since the time steps (in terms of physical and/or CPU time) of the components can be vastly different, this minimal synchronization provides the most possibilities for efficient concurrent execution.

In time accurate mode the coupling times have to be defined with the DtCouple arguments. For example

```
#COUPLE2
GM                      NameComp1
IM                      NameComp2
-1                      DnCouple
10.0                    DtCouple
```

will couple the GM and IM components every 10 seconds.

In some cases the models have to be coupled every time step. An example is the coupling between the MHD model GM/BATSRUS and the Particle-in-Cell model PC/IPIC3D. This can be achieved with

```
#COUPLE2TIGHT
GM                      NameMaster
PC                      NameSlave
T                       DoCouple
```

command. In this case the master component (GM) tells the slave component (PC) the time step to be used. The tight coupling requires models and couplers that support this option.

By default the component time steps are limited by the time of couplings. This means that if GM can take 4 second times steps, and it is coupled with IE every 5 seconds, then every second GM time step will be truncated to 1 second. There are two ways to avoid this. One is to choose the coupling frequencies to be round multiples of the time steps of the two components involved. This works well if both components have fixed time steps and/or much smaller time steps than the coupling frequency.

In certain cases the efficiency can be improved with the #COUPLETIME command, which can allow a component to step through the coupling time. For example

```
#COUPLETIME
GM                      NameComp
F                       DoCoupleOnTime
```

will allow the GM component to use 4 second time steps even if it is coupled at every 5 seconds. Of course this will make the data transferred during the coupling be first order accurate in time.

3.3.3 Coupling order

The default coupling order is usually optimal for accuracy and consistency, but it may not be optimal for speed. In particular, the IE/Ridley_serial component solves a Poisson type equation for the data received from the other components (GM and UA). For sake of accuracy IE always uses the latest data received from the other components. If GM, UA and IE are coupled in the default order

```
#COUPLEORDER
4             nCouple
GM IE         NameSourceTarget
UA IE         NameSourceTarget
IE UA         NameSourceTarget
IE GM         NameSourceTarget
```

and the to-IE and from-IE coupling times coincide, e.g.

```
#COUPLE2
GM            NameComp1
IE            NameComp2
10.0          DtCouple
-1            DnCouple

#COUPLE2
UA            NameComp1
IE            NameComp2
10.0          DtCouple
-1            DnCouple
```

then GM and UA will have to wait until IE solves the Poisson equation, because IE receives new data and it is required to produce results immediately. With the reversed coupling order

```
#COUPLEORDER
4               nCouple
IE UA       NameSourceTarget
IE GM       NameSourceTarget
GM IE       NameSourceTarget
UA IE       NameSourceTarget
```

IE will provide the solution from the previously received data, and it will have time to work on the new data while GM and UA are working on their time steps. The reversed coupling order allows the concurrent execution of IE with other components. The temporal accuracy, on the other hand, will be somewhat worse.

To demonstrate that the coupling order is important, here is a very **inefficient** coupling order

```
#COUPLEORDER
4             nCouple
GM IE         NameSourceTarget
IE GM         NameSourceTarget
UA IE         NameSourceTarget
IE UA         NameSourceTarget
```

in case the coupling times with GM and UA coincide (always at the beginning of a the sessions). With this coupling order, IE first receives information from GM, then solves the Poisson equation and returns the information based on the solution to GM while GM is waiting. Then IE receives extra information from UA, solves the Poisson equation again, and sends back information to UA, while UA is waiting.

An alternative way to achieve concurrent execution is to stagger the coupling times. For example the

```
#COUPLE2SHIFT
GM                  NameComp1
IE                  NameComp2
-1                  DnCouple
10.0                DtCouple
-1                  nNext12
0.0                 tNext12
-1                  nNext21
5.0                 tNext21
```

will schedule a GM to IE coupling at 0, 10, 20, 30, ... seconds, and the IE to GM coupling at 5, 15, 25, ... seconds. This provides IE half the GM time to solve the Poisson equations. If IE runs at least twice as fast as GM, this solution will produce concurrent execution. The temporal accuracy is somewhat better than in the reversed coupling case. Note that GM and IE will be synchronized at 0, 5, 10, ... seconds, which works best if the GM time step is an integer fraction of 5 seconds.

Chapter 4

Complete List of Input Commands

The content of this chapter is generated from the PARAM.XML file of the CON module and the PARAM.XML files in the component version directories (e.g. UA/GITM2/PARAM.XML). These XML files can be read with an editor and they can be used for creating PARAM.in files by copying small parts from them.

The XML files have been written by several developers at the Center of Space Environment Modeling. The transformation of the XML format into LaTex is done with the share/Scripts/XmlToTex.pl script. This script generates index terms for all commands, which are used to create an alphabetical index at the end of this chapter.

4.1 Input Commands for the Control Module

CON reads input parameters from the PARAM.in file and the files included into PARAM.in. All commands interpreted by CON start with the # character followed by capital letters and numbers. The commands can have an arbitrary number of parameters, which are written into the lines following the command. Other lines are ignored, and can be used for remarks. The general format of the parameter file is

```
#COMMANDNAME1
variable1
variable2

#COMMANDNAME2

#COMMANDNAME3
variable3
```

The #BEGIN_COMP ID and #END_COMP ID commands are exceptional in the sense that their parameters are written in the same line as the command itself. This exception makes the parameter file more readable. The parameter is the two-character component ID. There must be exactly one space between the #BEGIN_COMP or #END_COMP string and the ID. The lines between the #BEGIN_COMP ID and the matching #END_COMP ID commands are passed to the component with the corresponding ID. For example

```
#BEGIN_COMP GM

#AMR
-1

#END_COMP GM
```

The parameters passed to the components can be of arbitrary format. The only restriction is that the length of the lines cannot exceed 100 characters (extra characters will be ignored).

4.1.1 General commands

#INCLUDE command

```
#INCLUDE
RESTART.in              NameIncludeFile
```

The NameIncludeFile parameter contains the name of the file to be included. The file name may be followed with a trailing comment if it is separated with at least 3 spaces or one TAB character. The #INCLUDE command can be used anywhere in the parameter file, even in the sections which contain the component specific parameters. For example the information in the run/GM/restartIN/restart.H file or parameters specific to a component can be included.

#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.

#STRICT command

```
#STRICT
T                       UseStrict
```

If UseStrict is true, the SWMF does not attempt to correct problems, but it stops after the warning message. If it is set to false, SWMF attempts to correct the problems after the warning message is issued. It is possible to switch back and forth between strict and non-strict mode. A typical use is to switch off strict mode when a component is switched off in a session so some of the processors are idling. Once all processors are used, it is a good idea to switch back to strict mode.

The default is strict mode.

#DESCRIPTION command

```
#DESCRIPTION
This is a test run for GM-IE-UA coupling.
```

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.

4.1.2 Time and session control

The execution of SWMF is done in consecutive sessions. Each session is executed with a set of parameters read from the parameter file. After the session is finished, CON reads and distributes the parameters for the next session. Parameters from previous sessions are carried over, so only the changes relative to the previous session need to be given.

#RUN command

```
#RUN
```

The #RUN command does not have any parameters. It signals the end of the current session, and makes CON 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 CON execute the last session.

#TIMEACCURATE command

```
#TIMEACCURATE
F               IsTimeAccurate
```

If IsTimeAccurate is set to true, the SWMF is solving 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 the frequency of coupling, saving 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.

The steady-state mode also allows the components to use various acceleration techniques. For example the BATSRUS code (in the GM, IH, SC components) can use local time stepping to accelerate convergence to steady state. In steady state mode the various components are allowed to use different number of iterations per global iteration, thus they can converge to steady state more optimally (see the #CYCLE command).

The default value is the time accurate mode.

#STARTTIME command

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

The #STARTTIME command sets the initial date and time for the simulation in Greenwich Mean Time (GMT) or Universal Time (UT). This time is stored in the restart files. It can be overwritten with another #STARTTIME command if necessary.

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

#ENDTIME command

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

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.

#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 file, so that the simulation can continue from the same time when the restart was saved. It can be overwritten with another #TIMESIMULATION command if necessary. The default value is tSimulation=0.

#NSTEP command

```
#NSTEP
100                     nStep
```

The nStep variable contains the number of time steps since the beginning of the simulation (including all restarts). The #NSTEP command and the nStep variable are saved into the restart file, so that the simulation can continue from the same time step when the restart was saved. It can be overwritten with another #NSTEP command if necessary. The default value is nStep=0.

#STOP command

```
#STOP
100                     MaxIter
1.5 hour                TimeMax
```

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. If TimeMax is negative in time accurate mode or MaxIter is negative in steady state mode, the code stops with an error message. If the values are set to zero, the code will run and stop as normal, but not advance the solution. The #STOP command must be used in every session. The only exception is the last session, where the #ENDTIME command can be used instead of #STOP in time-accurate mode.

If the code completes the last session of the run successfully, both the SWMF.SUCCESS and the SWMF.DONE files are created in the run directory.

#CHECKTIMESTEP command

```
#CHECKTIMESTEP
T               DoCheckTimeStep (rest is read if true)
10              DnCheckTimeStep (2 or more)
-1.0            TimeStepMin [s] (negative value means SessionTime/10,000,000)
```

This command is only effective in time accurate mode. The goal is to avoid a simulation stalling with infinitesimal time steps.

If DoCheckTimeStep is true, then check if the average time advance over DnCheckTimeStep iterations is smaller than TimeStepMin, and if it is, stop the simulation with an error message. The check is performed every nCheckTimeStep iterations on all active processors of the SWMF. A positive TimeStepMin is the minimum average time advance in units of seconds. A negative TimeStepMin value is replaced with the simulation time of the session divided by hundred million, which assumes that a session should be completed in less than hundred million iterations.

Default values are shown above, so the check is active by default.

#CHECKKILL command

```
#CHECKKILL
GM              NameCompCheckKill
```

The SWMF can check periodically if the SWMF.KILL file exists in the run directory, and kill the execution if it does. This file gets removed at the beginning of the run so that new runs don't get killed accidentally. This check is done from the root processor of the component NameCompCheckKill in the SWMF time loop. This means that the component doing the check will not get interrupted at an arbitrary point of execution (e.g. writing output files). The other components sharing the same processor will also be safe. Components running on a separate subset of processors may get interrupted at a more-or-less arbitrary point of execution.

If NameCompCheckKill is set to the string '??' then all processors check for the SWMF.KILL file. This makes sure that the run gets terminated essentially immediately. If NameCompCheckKill is set to the string '!!' then no check is performed at all. Otherwise NameCompCheckKill should contain the component ID of a component present in the #COMPONENTMAP command.

If the code gets killed, no SWMF.SUCCESS or SWMF.DONE files are created.

The default value is NameCompCheckKill='!!', i.e. no check is performed.

#CHECKSTOP command

```
#CHECKSTOP
T                       DoCheckStop
-1                      DnCheckStop
10.0                    DtCheckStop
```

The DoCheckStop variable controls whether CON should check the CPU time or the existence of the SWMF.STOP file in the run directory. If it is set to false, there is no checking. If it is set to true, the stop conditions are checked at the frequencies given by the DnCheckStop and DtCheckStop variables. The DnCheckStop variable determines the frequency in terms of the time step number nStep, while DtCheckStop determines the frequency in terms of the simulation time tSimulation. Negative values for either variable mean that the corresponding condition is not checked. For time accurate mode DtCheckStop, for steady-state mode DnCheckStop is the relevant frequency.

The default value is DoCheckStop=.false., because the checks require synchronization of the components. The more frequent the checks are the less efficient the execution. On the other hand the less frequent the checks are, the less control the user has to stop the code at a given time.

If the code is stopped this way for any reason (for example the CPU time maximum is reached or the SWMF.STOP file was used), an SWMF.SUCCESS file is created but no SWMF.DONE file is created.

#CHECKSTOPFILE command

```
#CHECKSTOPFILE
T                       DoCheckStopFile
```

If DoCheckStopFile is true (and DoCheckStop is set to true in the #CHECKSTOP command) then the code checks if the SWMF.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, for example, the "touch SWMF.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. (but the default for DoCheckStop is .false.).

#CPUTIMEMAX command

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

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 and the DoCheckStop variable also must be set to .true. in the #CHECKSTOP command.

4.1.3 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 CON_stop 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
do_session                       StringTest
```

A space separated list of subroutine and function names to be tested. Only subroutines containing the 'call CON_set_do_test(...)' statement can be tested. The first argument is the name of the subroutine, usually defined as the string parameter 'NameSub'. Default is an empty string.

#TESTPROC command

```
#TESTPROC
1                        iProcTest
```

The test information will be written from iProcTest when DoTestMe is used in the SWMF. Default value is 0.

#VERBOSE command

```
#VERBOSE
100                     lVerbose
```

The lVerbose variable sets the verbosity of CON.
If lVerbose=0 the verbose information is minimized.
If lVerbose=1 some verbose information is printed in CON_main and CON_io from processor zero.
If lVerbose=10, processor zero will produce a line on the standard output with the name of the subroutine and the iteration number for all subroutines which call CON_set_do_test.
If lVerbose=100, all processors and all subroutines which call CON_set_do_test will produce a line on the standard output with the name of the subroutine, the iteration number and the processor number.
The default value is lVerbose=1.

#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
tree all                TypeTimingReport 'cumu','list','tree', +optional 'all'
```

If UseTiming=.true., the execution is timed by the TIMING utility.
If UseTiming=.false., the execution is not timed.

The DnTiming parameter determines the frequency of timing reports:
If DnTiming is positive, a timing report is produced every DnTiming steps.
If DnTiming is -1, a timing report is shown at the end of each session.
If DnTiming is -2, a timing report is shown at the end of the whole run.
If DnTiming is -3, no timing report is shown.

The nDepthTiming parameters defines the depth of the timing tree.
A negative value means unlimited depth.
If nDepthTiming is 1, only the total SWMF execution is timed.

The TypeTimingReport parameter 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. By default only the root CPUs of the components do timings. When all the CPU-s are timed, it is probably a good idea to direct the output into separate files (see the #STDOUT command).

The default values are shown above, except for the TimingReportStyle, which is 'cumu' by default without the 'all'.

#PROGRESS command

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

The DnShowProgressShort and DnShowProgressLong variables determine the frequency of showing short and long progress reports in terms of the number of time steps nStep. The short progress report consists of a single line which shows the number of time steps, simulation time and CPU time. The long progress report also shows a small timing report on the root processor. Negative values indicate that no report is requested.

The default values are DnShowProgressShort=10 and DnShowProgressLong=100.

#PRECISION command

```
#PRECISION
8                       nByteReal
```

The nByteReal variable gives the number of bytes in the default real variable. Possible values are 4 and 8. This command serves to check consistency with binary data, such as binary restart files. The #PRECISION command and the nByteReal variable are saved into the restart file. If the compiled precision differs from the one defined by nByteReal, the code stops with an error message in strict mode. If the strict mode is switched off with the #STRICT command, only a warning is printed.

There is no default value. If the command is not used, the precision of the real numbers is not checked.

#VERSION command

```
#VERSION
1.0                     VersionSwmf
```

This command is obsolete. We now use Git references to identify the code version. The only use of this command is to be compatible with restart files produced before VersionSwmf was removed.

4.1.4 Component control

The session model allows concurrent execution of the components. The components are synchronized only when conditions for the actions of coupling, saving restart files, or stopping execution are met. This is possible because it is known in advance by each component when these actions will occur. In time accurate mode the components are required to make a time step which does not exceed the next synchronization time. In steady state mode there is no limit on the time step, and components can be called at different frequencies. Components which are not used in a session can be switched off.

#COMPONENTMAP command

ID Proc0 ProcEnd Stride nThread TAB SEPARATED comment

```
#COMPONENTMAP
IE  0  0  1           CompMap IE runs on PE 0
PW  1 32  4  4        CompMap PW runs on PEs 1..32 with  4 threads
GM 33 -1 16 16        CompMap GM runs on PEs 33... with 16 threads
```

ID Proc0 ProcEnd Stride nThread TAB SEPARATED comment

```
#LAYOUT
GM  0 -3 -2 -2        CompMap GM runs on PEs ..nProc-3 with MaxThread/2 threads
PC  0 -3 -1 -1        CompMap PC runs on PEs ..nProc-3 with MaxThread threads
IM -2 -2  1           CompMap IM runs on PE  nProc-2
IE -2 -1  1           CompMap IE runs on PEs nProc-2 and nProc-1
```

This command can only occur in the first session, where it is required. The #COMPONENTMAP (or #LAYOUT) command lists the active components and their processor layout including the number of OpenMP threads for components that can run with OpenMP. To use multithreading, the code needs to be configured with the -openmp flag. The list of components should be ended with an empty line.

The columns contain the component ID, the index of the first (root) processor for the component, the last processor, the stride, and the optional number of threads, respectively. Negative values for the root and last processor ranks are taken as counting backwards from the total number of processors nProc. Negative values for the stride and number of threads are interpreted as the maximum number of threads (MaxThread is defined by the OMP_NUM_THREADS environment variable) divided by the absolute value of Stride or nThread, respectively. This allows the same layout to be used on different nodes with different number of cores per node. The default number of threads is 1 and the maximum value is MaxThread.

This command is required in the first session of the PARAM.in file.

#COMPONENT command

```
#COMPONENT
IE                      NameComp
F                       UseComp
```

The NameComp variable contains the two-character component ID, while the UseComp variable defines if the component should be used in the current session or not. It is possible that in the initial sessions some components are not used, and they are turned on later. Unused components should not be coupled, and they should not be given any parameters.

The default is that all the components listed in the #COMPONENTMAP command are used.

#CYCLE command

```
#CYCLE
IH                      NameComp
10                      DnRun
```

The DnRun variable defines the frequency of calling component NameComp during a steady state run. In the example IH will be called for nStep = 10, 20, 30, ... For time accurate runs this command has no effect. The default is DnRun = 1 for all the active components.

4.1.5 Coupling control

#COUPLEORDER command

```
#COUPLEORDER
5                       nCouple
IE GM                   NameSourceTarget
IE IM                   NameSourceTarget
GM IE                   NameSourceTarget
GM IM                   NameSourceTarget
IM GM                   NameSourceTarget
```

The nCouple variable determines the maximum number of couplings among the components. The NameSourceTarget string contains the two two-character IDs for the source and target components separated by a space. The order of the couplings is most important for the couplings done at the beginning of the session. Some components need to get information from the others before they can initialize properly. For example IM/RCM2 gets information from GM/BATSRUS.

With respect to coupling between IE and the other components there are two strategies:

1. All components send information to IE first, then IE solves for the potential and sends it back to all the other components.

2. IE sends information based on its current state first, then receives info from the other components, and solves for the potential in parallel with the other components.

The default coupling order follows the first strategy. In this case IE runs in a serial mode, so it should share processors with one of the components that are coupled to IE.

The example shown above follows the second strategy, which leads to concurrent execution IF the layout puts IE on a processor that is NOT shared with the slowest of the components (typically GM, sometimes IM). To use the second strategy it is simplest to add the

```
#INCLUDE
Param/CoupleOrderFast
```

command into the PARAM.in file. The Param/CoupleOrderFast contains the coupling order that allows concurrent execution of IE.

NOTE: The #COUPLEORDER command defines only the order of the couplings, so it can list couplings that are not active in a run.

NOTE: The order of the 'SC SP' and 'IH SP' couplings should not be reversed!

The default coupling order is specified by the iCompCoupleOrder_II array initialized in the CON/Coupler/src/CON_coupler.f90 file.

#COUPLE1 command

```
#COUPLE1
IE                      NameSource
IM                      NameTarget
-1                      DnCouple
10.0                    DtCouple
```

The NameSource and NameTarget variables contain the two-character component ID-s for the source and target components. The DnCouple variable determines the frequency of couplings in terms of the number of time steps nStep for steady state runs, while DtCouple defines the frequency in terms of the simulation time tSimulation in seconds for time-accurate runs. Setting both frequencies to a negative value means that there is no coupling.

The default is no coupling between the components.

#COUPLE2 command

```
#COUPLE2
GM                      NameComp1
IE                      NameComp2
-1                      DnCouple
10.0                    DtCouple
```

The NameComp1 and NameComp2 variables contain the two-character component ID-s for the two components which are coupled both ways. The DnCouple variable determines the frequency of couplings in terms of the number of time steps nStep for steady state runs, while DtCouple defines the frequency in terms of the simulation time tSimulation in seconds for time-accurate runs. Setting both frequencies to a negative value means that there is no coupling.

The default is no coupling between the components.

#COUPLE1SHIFT command

```
#COUPLE1SHIFT
IH                      NameSource
GM                      NameTarget
-1                      DnCouple
10.0                    DtCouple
-1                      nNext12
3.0                     tNext12
```

The NameSource and NameTarget variables contain the two-character component ID-s for the source and target components. The DnCouple variable determines the frequency of couplings in terms of the number of time steps nStep for steady state runs, while DtCouple defines the frequency in terms of the simulation time tSimulation in seconds for time-accurate runs.

For steady-state simulations the nNext12 variable determines in which time step the first coupling occurs after the initial coupling, namely when mod(nStep,DnCouple) equals nNext12. For time accurate simulations the tNext12 variable determines at what simulation time the first coupling occurs after the initial coupling, namely when mod(tSimulation,DtCouple) equals tNext12.

The above example will couple IH to GM at simulation times 3, 13, 23, etc.

The default is no shifting.

#COUPLE2SHIFT command

```
#COUPLE2SHIFT
GM                  NameComp1
IE                  NameComp2
-1                  DnCouple
10.0                DtCouple
-1                   nNext12
3.0                 tNext12
-1                  nNext21
6.0                 tNext21
```

The NameComp1 and NameComp2 variables contain the two-character component ID-s for the two components which are coupled both ways. The DnCouple variable determines the frequency of couplings in terms of the number of time steps nStep for steady state runs, while DtCouple defines the frequency in terms of the simulation time tSimulation in seconds for time-accurate runs.

For steady-state simulations the nNext12 variable determines in which time step the first coupling occurs from NameComp1 to NameComp2 after the initial coupling, namely when mod(nStep,DnCouple) equals nNext12. For time accurate simulations the tNext12 variable determines at what simulation time the first coupling occurs from NameComp1 to NameComp2 after the initial coupling, namely when mod(tSimulation,DtCouple) equatls tNext12.

The first coupling step and time for the NameComp2 to NameComp1 coupling is determined by the nNext21 and tNext21 variables in a similar fashion. This command allows to shift the couplings relative to each other.

The above example will couple GM to IE at simulation times 3, 13, 23, etc, while IE will be coupled to GM at simulation times 6, 16, 26 etc. This way IE can solve the potential problem while GM advances by 3 seconds. That can improve the parallelization and efficiency.

The default is no shifting.

#COUPLE1TIGHT command

```
#COUPLE1TIGHT
```

```
GM                      NameMaster
PT                      NameSlave
T                       DoCouple
```

Couple two components tightly one-way. Tight coupling means that the two components must take identical time steps, and they are coupled every time step.

The NameMaster and NameSlave parameters contain the two-character component ID-s for the master and slave components. The tight coupling can be switched on or off with the DoCouple parameter. Use the COUPLE2TIGHT command for two-way coupling.

The default is no coupling between the components.

#COUPLE2TIGHT command

```
#COUPLE2TIGHT
GM                      NameMaster
PC                      NameSlave
T                       DoCouple
```

Couple two components tightly two-way. Tight coupling means that the two components must take identical time steps, and they are coupled every time step.

The NameMaster and NameSlave parameters contain the two-character component ID-s for the master and slave components. The tight coupling can be switched on or off with the DoCouple parameter. Use the COUPLE1TIGHT command for one-way coupling.

The default is no coupling between the components.

#COUPLETIME command

```
#COUPLETIME
GM                      NameComp
F                       DoCoupleOnTime
```

The NameComp variable contains the two-character component ID, while the DoCoupleOnTime parameter defines if the time step of the component should be limited such that it does not exceed the next coupling time. If DoCoupleOnTime is true, the component will limit the time step for couplings, if it is false, the time step is only limited by the final time, the time of saving restarts and the time of checking stop conditions.

The default is that all components limit their time steps to match the coupling time.

#FIELDLINE command

```
#FIELDLINE
SP                      NameTarget
3                       nSource
SC                      NameSource
1.05                    RScMin
21                      RScMax
IH                      NameSource
19.7                    RIhMin
230.0                   RIhMax
OH                      NameSource
220.0                   RIhMin
1500.0                  RIhMax
```

The NameSource and NameTarget variables contain the two-character component ID-s for the source and target components coupled via MFLAMPA. RScMin and RScMax radii for SC componet, and RIhMin and RIhMax for IH component, and ROhMin and ROhMax for OH component bound the domains in SC, IH, and OH corrrespondingly, coupled to SP or PT particle-treated components.

#COUPLEFIELDLINE command

```
#COUPLEFIELDLINE
-1                      DnCouple
10.0                    DtCouple
```

For all components coupled via 'field lines' via command '#FIELDLINE' the DnCouple variable determines the frequency of couplings in terms of the number of time steps nStep for steady state runs, while DtCouple defines the frequency in terms of the simulation time tSimulation in seconds for time-accurate runs. Setting both frequencies to a negative value means that there is no coupling.

The default is no coupling.

#LOOKUPTABLE command

```
#LOOKUPTABLE
PT GM                   StringCompTable
ChargeExchange          NameTable
load                    NameCommand
OH/Param/ER.dat         NameFile
ascii                   TypeFile
```

The first parameters contains a space separated list of comoponents that need this particular lookup table. "CON" refers to the control module, so all processors read in the lookup table. The example shows loading the ExchangeRate lookup table for two components GM and PT from an ASCII file OH/Param/ExchangeRate.dat. For more complete description, see the #LOOKUPTABLE command description in GM/BATSRUS/PARAM.XML.

This command can occur multiple times. By default no lookup tables are used.

4.1.6 Restart control

CON needs to coordinate the saving of restart information for itself and all the components. It is important that all components save the necessary information at the same simulation time.

#SAVERESTART command

```
#SAVERESTART
T                       DoSaveRestart (Rest of parameters read if true)
1000                    DnSaveRestart
-1.                     DtSaveRestart
```

The DoSaveRestart variable determines whether restart information should be saved or not. The rest of the parameters are read only if DoSaveRestart is true. For steady state runs the frequency of saving restart information is given by the DnSaveRestart variable in terms of the number of time steps nStep, while for time accurate run, the DtSaveRestart variable determines the frequency in terms of the simulation time tSimulation in seconds. The code stops with an error message if DoSaveRestart is true but DnSaveRestart is negative in steady state mode or DtSaveRestart is negative in time accurate mode.

Irrespective of the frequencies, final restart files are always saved if DoSaveRestart is true.

Defaults are DoSaveRestart=true, DnSaveRestart=100000 and DtSaveRestart=1e30. For typical runs this means that only a final restart file is saved. It is a good idea, however, to save restart files multiple times during long runs.

#RESTARTOUTDIR command

```
#RESTARTOUTDIR
SWMF_RESTART.YYYYMMDD_HHMMSS            NameRestartOutDir
```

The main purpose of this command is to allow saving multiple restart trees in time accurate mode without running the Restart.pl script in the background.

NameRestartOutDir sets the name of the output restart directory tree for the SWMF. If the YYYYMMDD_HHMMSS string is part of NameRestartOutDir, it will be replaced with the date-time information corresponding to the restart data, for example "SWMF_RESTART.20150209_124900/". The SWMF output restart file (see #RESTARTFILE) will be written into this directory. The components should also write their restart information into a subdirectory named by the component ID, for example "SWMF_RESTART.20150209_124900/GM/". This feature is currently only implemented for BATSRUS and IM/RCM2.

The default value is an empty string, so the SWMF restart file is written into the run directory and all components write into their respective output restart directories. The Restart.pl script can be used to move this information into a restart tree. The Restart.pl script works for all the components.

#RESTARTFILE command

```
#RESTARTFILE
RESTART.out
```

The NameRestartFile variable contains the name of the SWMF restart file. This file contains information such as initial date and time, simulation time, time step, version number, description of the simulation, name of the planet, etc. This file is usually written into the main run directory, but this can be changed with the #RESTARTOUTDIR command. The Restart.pl script can be used to link this file to the RESTART.in, which is normally included into PARAM.in with the #INCLUDE command for restarts.

The default value for NameRestartFile is "RESTART.out".

4.1.7 Output control

#ECHO command

```
#ECHO
T                       DoEcho
```

If the DoEcho variable is true, the input parameters are echoed back. The echoing either goes to the standard output or into the log files depending on the UseStdout variable, which can be set in the #STDOUT command.

The default value for DoEcho is true and it is a good idea to leave it that way.

#FLUSH command

```
#FLUSH
F                       DoFlush
```

If the DoFlush variable is true, the output is flushed when subroutine ModUtility::flush_unit is called. This is typically used in log files. The flush is useful to see the output immediately, but on some systems it may be very slow.

The default is to flush the output, i.e. DoFlush=T.

#MAKEDIR command

```
#MAKEDIR
F                       DoMakeDir
```

If the DoMakeDir variable is true, the code will make directories as needed. On some machines (Pleiades), however, this can fail, so it is better to create all directories in advance and not trying to do it from the code. This is, of course, incompatible with a dynamic restart directory name in #RESTARTOUTDIR.

The default is DoMakeDir=T.

#STDOUT command

```
#STDOUT
F                       UseStdout
```

If the UseStdout variable is true, the echoed input parameters and other verbose information produced by the components are written to the standard output. To distinguish between the output produced by the various components, a prefix string is written at the beginning of the lines. Usually the prefix string contains the component ID, the processor number if the line is written by a processor which is not the root processor of the component, and a colon (for example "GM0001:"). Even with the prefix, it may be difficult to collect the output from the various components and processors. The order of the output depends on how the MPI library buffers that, which is platform dependent.

If the UseStdout variable is false, the echoed input parameters and other verbose information produced by the components are written into separate files in the STDOUT directory (the name of this directory can be changed by the #STDOUTDIR command). The files are named similarly to the prefix string: the component ID is followed by the global processor number and a ".log" extension is added. For example the root processor of GM may write into "STDOUT/GM0014.log" if GM's root processor has global rank 14.

Note that warnings and error messages should always be written to the standard output, and they should always have a prefix which identifies the component issuing the warning or error. CON itself always writes to the standard output and it does not use a string prefix.

The default value for UseStdout is true.

#STDOUTDIR command

```
#STDOUTDIR
STDOUT/Test             NameStdoutDir
```

The NameStdoutDir variable contains the name of the directory where the log files with the redirected standard output of the components are written if UseStdout is set to .false. in the #STDOUT command. The directory must exist before the run is started.

The default value of NameStdoutDir is "STDOUT".

4.1.8 Solar coordinate commands

We allow an offset for the HGR and HGI/HGC systems so that the interesting features are aligned with the primary axis. One common option is to have the planet in the -X,Z plane. Another option would be to move an active region into an appropriate plane.

#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.

4.1.9 Planet commands

Several components share information about the planet for which the simulation is done. It is important that the various components use compatible information about the planet. It is also useful for the couplers that they can globally access this information, such as radius and orientation of the planet, or its magnetic field. The SWMF is designed to work for an arbitrary planet. It also allows to change some parameters of the planet relative to the real values.

By default the SWMF works with Earth and its real parameters. Another planet can be selected with the #PLANET command. The real planet parameters can be modified and simplified with the other planet commands listed in this subsection. These modifier commands cannot proceed 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, 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. It TypeBField is set to "DIPOLE" than 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", which is currently the only recognized planet.

#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).

#DIPOLE command

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

The DipoleStrength 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

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

```
#ROTATIONAXIS
T               IsRotAxisPrimary
0.0             RotAxisTheta
0.0             RotAxisPhi

#MAGNETICAXIS
F               IsMagAxisPrimary
```

but much shorter.

#ORBIT command

```
#ORBIT
3.14e7          OrbitalPeriodPlanet [s]
1.49e11         rOrbitPlanet [m]
0.016           Excentricity
174.9           RightAscension [deg]
7.155           Inclination    [deg]
288.9           ArgPeriapsis   [deg]
```

#TIMEEQUINOX command

```
#TIMEEQUINOX
2000                    iYear
1                       iMonth
1                       iDay
0                       iHour
0                       iMinute
```

Reset the time of eqinox for any planet

4.1.10 Star commands

Several components share information about the star for which the simulation is done. It is important that the various components use compatible information about the star. It is also useful for the couplers that they can globally access this information, such as radius and rotation period. The SWMF is designed to work for an arbitrary star.

By default the SWMF works with the Sun and its real parameters. Another star can be selected with the #STAR command.

#STAR command

```
#STAR
SUN                     NameStar (upper case, up to 3 letters)
1.0                     RadiusStar (in solar radii)
1.0                     MassStar   (in solar masses)
0.0                     RotationPeriodStar (in days)
```

Modify the parameters of the central star (when SWMF or BATSRUS is running in corona/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.

#HGRALIGNMENTTIME command

```
#HGRALIGNMENTTIME
1965                    iYear
1                       iMonth
1                       iDay
0                       iHour
0                       iMinute
```

Reset the time at which HGR and HGI are aligned. In application to stars it may be needed to match the magnetogram (in HGR coords) to the exoplanet orbital elements (in HGI coords)

4.1.11 Stub components

If SWMF is compiled with the interface in srcCON/Stubs, the stub components recognize only one command #TIMESTEP.

#TIMESTEP command

```
#TIMESTEP
0.01       DtRun (the typical time step of the component)
0.12       DtCpu (the CPU time needed for 1 time step)
```

The DtRun variable defines the typical time step of the component in terms of simulation time. The DtCpu variable determines the CPU time needed to execute one time step for the component. Both variables are given in seconds.

Of course it is not necessary to put in the actual CPU times. One can take the same fraction for all components to accelerate the run.

4.2 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

4.2.1 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.

#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.

4.2.2 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.

4.2.3 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.

4.2.4 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.

4.2.5 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 strans-formation 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 UseOutflow-Pressure=.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 #OUTFLOWPRESSURE
'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 information.
'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'.

#INNERBCPE command

```
#INNERBCPE
0.1             RatioPe2P
```

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 RatioRe2P. The default value of RatioPe2P is 0.

#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*:

$$F_{O^+} = \alpha S_{\parallel}^{\beta} \tag{4.1}$$

...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:

$$n_{O^+} = F_{O^+}/U_R \tag{4.2}$$

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.

4.2.6 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.

4.2.7 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.

4.2.8 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.

4.2.9 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).

4.2.10 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.

4.2.11 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' only)
         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/rfr/lin/eqr/e
 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.

4.2.12 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.

4.2.13 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 String-Shape='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 #AMRCRI-TERIARESOLUTION 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/AMRCRITERIARESOLUTIO 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):

```
'gradT'               - gradient of temperature
'gradP'               - gradient of pressure
'gradlogrho'          - gradient of log10(rho)
'gradlogP'            - gradient of log10(P)
'gradE'               - gradient of electric field magnitude
'curlV','curlU'       - magnitude of curl of velocity
'curlB'               - magnitude of current
'J2'                  - current squared
'currentsheet'        - current sheet (radial B changes sign)
'divU', 'divV'        - divergence of velocity
'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:

$$E = \frac{\frac{d^2 U}{dx^2}}{\frac{1}{DX}\left(\frac{dU}{dx}\right) + \frac{SmallError}{DX^2} * U + Epsilon}$$

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).

4.2.14 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 StringLowOrderRegion 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 nCheck-TimeStep 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 $u = \sum\_s(\rho\_s u\_s)/\sum\_s \rho\_s$ 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.

#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
```

4.2.15 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.

4.2.16 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).

4.2.17 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. 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 = $Nm^2/(A^2s)$. 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
```

If UseEntropy is true, 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.

The default value of UseEntropy is false, except when the #SHOCKHEATING command is used, which requires and sets UseEntropy=T.

#SHOCKHEATING command

```
#SHOCKHEATING
0.5                     PeShockHeatingFraction (read if electron pressure is used)
1.0                     PparShockHeatingFraction (read if ion pressure is anisotropic)
```

If PeShockHeatingFraction is positive, a fraction of the non-adiabatic heating is deposited into the electron thermal energy Pe/(GammaE-1), and the same amount of energy is removed from the ion thermal energy P/(Gamma-1). If PparShockHeatingFraction is positive, 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 PeShockHeatingFraction 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 the command is used, both UseEntropy and UseElectronEntropy are set to true (see #ENTROPY and #ELECTRONENTROPY).

Default values are 0, which means that all shock heating goes to the (perpendicular) ion pressure.

#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.

#IMPLICITCORONALHEATING command

```
#IMPLICITCORONALHEATING
F                       UseImplicitCoronalHeating
```

If UseImplicitCoronalHeating is false, the coronal heating is added explicitly. If UseImplicitCoronalHeating is true, the coronal heating is solved point implicitly together with the heat conduction.

The default is UseImplicitCoronalHeating=.false.

#FIXISOTROPIZATION command

```
#FIXISOTROPIZATION
F                       UseFixIsotropization
```

If UseFixIsotropization is true, an addtional term is added to isotropize the proton temperature anisotropy due to Coulomb collisions.

The default is UseFixIsotropization=.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.

4.2.18 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

$$(\delta \mathbf{u}_\perp \cdot \delta \mathbf{u}_\perp)^{1/2} \approx 15 \text{ km s}^{-1} \left(\frac{3 \cdot 10^{-11} \text{ kg m}^{-3}}{\rho} \right)^{1/4}, \tag{4.3}$$

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.

4.2.19 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          (optional)
20        MaxIteration       (optional)
```

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.

4.2.20 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.

4.2.21 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 $\pm V\_A = \pm|B|/\sqrt{\rho}$.

#WAVEREPRESENTATIVE command

```
#WAVEREPRESENTATIVE
F                       UseAlfenWaveRepresentative
```

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.

4.2.22 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.

4.2.23 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.

4.3 Input Commands for the Ridley Ionosphere Model: IE Component

4.3.1 Testing

#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.

#DEBUG command

```
#DEBUG
2                       iDebugLevel
3                       iDebugProc
```

The iDebugLevel variable sets the level of debug information for the processor selected by iDebugProc.
Default is iDebugLevel=-1 which is no debug info on any and iDebugProc=0.

4.3.2 Input and output

#RESTART command

```
#RESTART
T                       DoRestart
```

Read restart file if DoRestart is true. This is useful when the ionosphere model runs in parallel with the other models in the SWMF (fast coupling). Default is false.

#IONODIR command

```
#IONODIR
IE/Plots                NameIonoDir
```

The NameIonoDir variable contains the name of the directory to store output files. Default value is "IE/ionosphere".

#SAVEPLOT command

```
#SAVEPLOT
2                       nPlotFile
min idl                 StringPlot
-1                      DnSavePlot
10.0                    DtSavePlot [sec]
max tec                 StringPlot
-1                      DnSavePlot
20.0                    DtSavePlot [sec]
```

The StringPlot variable consists of two string parts: the TypePlotVar string can be 'min', 'max', 'uam' or 'aur' corresponding to a minimum, maximum, upper atmosphere or auroral set of plot variables. The other

part TypePlotForm can be 'idl' or 'tec' corresponding to plot files for IDL or TecPlot. The DnSavePlot and DtSavePlot variables determine the frequency of saves in terms of time step or physical time.

The default is that no plotfiles are saved.

The 'min' variable set includes: Theta [deg], Psi [deg], SigmaH [mhos], SigmaP [mhos], Jr [mA/m^2], Phi [kV] The 'max' variable set includes: X [Re], Y [Re], Z [Re], Theta [deg], Psi [deg], SigmaH [mhos], SigmaP [mhos], E-Flux [W/m^2], Ave-E [eV], Jr [mA/m^2], Phi [kV], Ex [mV/m], Ey [mV/m], Ez [mV/m], Jx [microA/m^2], Jy [microA/m^2], Jz [microA/m^2], Ux [km/s], Uy [km/s], Uz [km/s], JouleHeat [mW/m^2], IonNumFlux [/cm^2/s], RT 1/B [1/T], RT Rho [amu/cm^3], RT P [Pa], conjugate dLat [deg], conjugate dLon [deg] The 'uam' variable set includes: Theta [deg], Psi [deg], SigmaH [mho], SigmaP [mho], Jr [mA/m^2], Jr(NW) [mA/m^2], E-Flux [W/m^2], Ave-E [eV], Phi [kV] The 'aur' variable set includes: Theta [deg], Psi [deg], SigmaH [mho], SigmaP [mho], Jr [mA/m^2], Phi [kV], E-Flux [W/m^2], Ave-E [eV], RT 1/B [1/T], RT Rho [amu/cm^3], RT P [Pa], JouleHeat [mW/m^2], IonNumFlux [/cm^2/s], conjugate dLat [deg], conjugate dLon [deg]

#SAVEPLOTNAME command

```
#SAVEPLOTNAME
F                       IsPlotName_e
```

Plot files are named with the new substring including full year, ie. _e20000321-104510-000

#SAVELOGNAME command

```
#SAVELOGNAME
F                       IsLogName_e
```

Log files are named with the new substring including full year, ie. _e20000321-104500-000

#SAVELOGFILE command

```
#SAVELOGFILE
F                       DoSaveIELogfile
```

If true, every time that iono_solve is called, iteration, time, and solution information is written to a logfile.

4.3.3 Physical parameters

#IONOSPHERE command

```
#IONOSPHERE
5                       iConductanceModel
F                       UseFullCurrent
F                       UseFakeRegion2
-1.0                    F107Flux
0.25                    StarLightPedConductance
0.25                    PolarCapPedConductance
```

The iConductanceModel variable determines which ionosphere model is used:
0 - uses a constant Pedersen conductance which is set by StarLightPedConductance
1 - uses a constant Pedersen conductance which is set by StarLightPedConductance, and a constant Hall conductance which is set by PolarCapPedConductance
2 - uses a solar EUV combined with a nightside conductance, so it uses F107Flux and StarLightPedConductance

3 - uses solar EUV, nightside, and crude oval, so uses F107Flux, StarLightPedConductance, and PolarCapPedConductance, since a polar cap is defined with the oval.
4 - restricted oval, uses same variables as 3.
5 - more realistic oval, uses same variables as 3.
8 - MAGNIT (requires #TRACEIE to be switched on in BATSRUS!!!) 9 - Powerlaw (simple powerlaw dependence of conductance on FAC) 10 - Pedersen conductance North-South 11 - Bob Robinson model

Model 4 and 5 differ in the way the conductances are limited to the fitted oval. Model 4 is more restrictive, while model 5 is somewhat more relaxed.

The time variation of the F10.7 flux can be taken into account by interpolating it from a lookup table. If F107Flux is set to -1.0, then the code is using a lookup table. This can be loaded explicitly in the main part of the PARAM.in file, for example

```
#LOOKUPTABLE
IE                      StringComp
F107                    NameTable
load                    NameCommand
MyF107.txt              NameFile
log                     TypeFile
```

If there is no table loaded and F107Flux is -1.0, then the code loads the Param/f107.txt table automatically.

The UseFullCurrent and UseFakeRegion2 logicals were used in the past to test various algorithmic choices. They should have a false value now. The default values are shown by the example above.

#IM command

```
#IM
average                 TypeImCouple
0.5                     FractionImJr
```

The TypeImCouple parameter determines which hemisphere the IM component is coupled to. If the value is 'north' or 'south', the potential and radial current are sent from the corresponding magnetic hemisphere. For 'cpcpmin' the hemisphere with the lower cross polar cap potential is selected. For TypeImCouple='average' the potential and radial current are averaged for the north and south hemispheres.

The FractionImJr parameter multiplies the field aligned currents received from IM (when the IM to IE coupling is swithed on) so they do not shield completely the weaker GM field aligned currents.

The default values are 'north' and 1.0 (full strength IM currents), which is backward compatible, and it requires no communication between the IE processors.

#UA command

```
#UA
T               DoCoupleUaCurrent
45.0            LatBoundary [deg] (only read if DoCoupleUaCurrent is true)
```

The DoCoupleUaCurrent parameter determines if the field aligned current calculated by the UA component should be used. Usually the currents are dominated by the field aligned currents provided by the GM component. The coupling with the UA currents is still experimental. If DoCoupleUaCurrent is set to true, the lower latitude boundary for the potential solver should be given with the LatBoundary paramter.

The default value is DoCoupleUaCurrent=.false, i.e. the UA currents are not included.

#AMIEFILES command

```
#AMIEFILES
IE/amie.north
IE/amie.south
```

Set the files to read the AMIE data from.

Default is not reading AMIE files.

#SPS command

```
#SPS
T                       UseSPS
```

The UseSPS parameter indicates if the serial potential solver is used. This is the default in the SWMF and it cannot be modified in the SWMF.

#BACKGROUND command

```
#BACKGROUND
dir                     NameOfModelDir
weimer96                NameOfEFieldModel
ihp                     NameOfAuroralModel
xyz                     NameOfSolarModel
```

This command cannot be used in the SWMF.

#CONDUCTANCEFILES command

```
#CONDUCTANCEFILES
cond_hal_coeffs.dat     NameHalFile
cond_ped_coeffs.dat     NamePedFile
```

Set the file names for the coefficients used in the empirical conductance model. Default values are shown for iConductanceModel set to 4 or 5 in the #IONOSPHERE command. For iConductanceModel=9 the defaults are cond_hal_coeffs_power.dat and cond_ped_coeffs_power.dat. Any files read with this command are expected to be placed into the IE folder of the run directory. The defaults files are automatically copied from IE/Ridley_serial/input/ directory.

The empirical conductance model estimates the auroral conductance between latitudes 60 degs and 90 degs using an empirical function of the form $\Sigma = \Sigma\_0 - \Sigma\_1 e^{-\Sigma\_2|J_{||}|}$, where the coefficients $\Sigma_{0,1,2}$ are constants for different lat-MLT configurations. The coefficient Σ_1, and occasionally Σ_0 are subsequently enhanced by the auroral oval function. The degree and type of enhancement is dependent on the kind of auroral oval/conductance model used (see #IONOSPHERE and/or #AURORALOVAL commands for more information).

The default coefficient files are stored in IE/Ridley_serial/input; they are copied to the run directory's IE folder automatically. These files are based off of curve fits between the Hall/Pedersen Conductance (in mho) at 110 km altitude and Upward/Downward Field Aligned Current (in $\mu A/m^2$) derived from AMIE results for the month of Jan 1997 (refer Ridley et al, 2004 for more info).

Files must be placed into the IE folder of the run directory. They are required to have a uniform MLT/lon grid. Header information, such as the number of points in both the latitude and magnetic local time directions, must be included. Points in MLT should not overlap midnight (i.e., there should not be an entry for both 00 MLT and 24 MLT).

Examples of the required format can be found in Ridley_serial/input/.

#USECMEE command

```
#USECMEE
T             UseCMEEFitting (rest of parameters read if true)
45            LatNoConductanceSI (default)
7.5           FactorHallCMEE (default)
5             FactorPedCMEE (default)
```

This command allows modifications in the auroral oval properties and addition of a baseline conductance when using CMEE. When the fitting option is true, a boundary condition at the boundary latitude (LatNoConductanceSI) is added to limit auroral conductance values equatorward of this boundary. The baseline conductance, added to the oval adjusted conductance to avoid spikes in CPCP, has two values for Hall and Pedersen conductance. Default values are shown above.

NOTE: If and when changing FactorHallCMEE and FactorPedCMEE from their default values, it is recommended that a higher value for both are chosen in order to avoid unrealistic potential values.

#AURORALOVAL command

```
#AURORALOVAL
T             UseOval (rest of parameters read if true)
T             UseOvalShift
F             UseSubOvalConductance
F             UseAdvancedOval
T             DoFitCircle (read if UseAdvancedOval is true)
```

This command controls the behavior of the auroral oval for conductance iModel=5. If UseOval is set to True, an auroral oval is constructed from the upward FAC pattern and added to the other sources of ionospheric conductance. UseOvalShift dictates whether or not the oval has a day-night positional shift. If False, the oval will always be centered about the magnetic pole. If True, the oval will be shifted daywards or nightwards to best fit the FAC pattern. UseSubOvalConductance controls if conductance from FACs is restricted at latitudes below the auroral oval. If True, conductance is not restricted at lower latitudes. If False, conductance below auroral latitudes falls off exponentially, producing a sharp conductance gradient about the oval. If UseAdvancedOval is set to True, an updated oval calculation is used. The behavior of the new approach is controlled by DoFitCircle. If true, a real circle is fit to the upward FACs via a minimization technique. If false, an approach that is similar to the original is used, but leverages FAC information from all local times. Both approaches result in an oval that more realistically reflects ionospheric dynamics and is less prone to position jumps as a function of time.

Default values are shown above.

#CONDUCTANCE command

```
#CONDUCTANCE
1.0                   OvalWidthFactor
1.0                   OvalStrengthFactor
1.7                   ConductanceFactor
```

Modifies the conductance by adjusting the oval width/strength for conductance models 4 and 5. OvalWidthFactor scales the width of the auroral oval. OvalStrengthFactor scales only the conductance applied to the auroral oval. This parameter only acts when the conductance model is iModel 5. ConductanceFactor scales the over all conductance from both FACs and the aurora.

Default values are shown above.

#BOUNDARY command

```
#BOUNDARY
10.0                    LatBoundary
```

Define the low latitude boundary for the potential solver. The default value is shown.

4.3.4 Scheme parameters

#SOLVER command

```
#SOLVER
bicgstab                NameSolver (gmres or bicgstab)
```

Select which solver to use. Default is bicgstab since it is faster.

#KRYLOV command

```
#KRYLOV
T                       UsePreconditioner
T                       UseInitialGuess
0.01                    Tolerance
100                     MaxIteration
```

This command controls the parameters for the Krylov solver used to solve the Poisson type equation for the electric potential. If UsePreconditioner is true the solver uses a preconditioner. If UseInitialGuess is true, the previous solution is used as an initial guess. The Tolerance parameter sets the second norm of the final (preconditioned) residual. The MaxIteration parameter sets the maximum number of iterations before the linear solver gives up. In most cases the default values should work fine.

The default values are shown above.

4.4 Input Commands for the CIMI: IM Component

List of IM commands used in the PARAM.in file

4.4.1 General commands

#ECHO command

```
#ECHO
T                       DoEcho
```

If the DoEcho variable is true, the input parameters are echoed back. The echoing either goes to the standard output. 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.

#INCLUDE command

```
#INCLUDE
IM/restartIN/restart.H          NameCIMIIncludeFile
```

The NameCIMIIncludeFile parameter contains the name of the file to be included. The file name may be followed with a trailing comment if it is separated with at least 3 spaces or one TAB character. The #INCLUDE command can be used anywhere in the parameter file, even in the sections which contain the component specific parameters. For example the information in the run/IM/restartIN/restart.H file or parameters specific to a component can be included.

#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.

4.4.2 Timing control

#TIMESIMULATION command

```
#TIMESIMULATION
0.0                     TimeSimulation
```

This command specifies the current simulation time. It is typically read from the IM/restartIN/restart.H file. Default value is 0.

#STOP command

```
#STOP
1 hour                  TimeSimulation
```

This command specifies the simulation stop time.

#STARTTIME command

```
#STARTTIME
2012                    iYear
11                      iMonth
12                      iDay
18                      iHour
00                      iMinute
00                      iSecond
```

The #STARTTIME command sets the initial date and time for the simulation in Greenwich Mean Time (GMT) or Universal Time (UT) in stand alone mode. This time is stored in the CIMI restart header file. CIMI has no default values.

#IMTIMESTEP command

```
#IMTIMESTEP
1.                      IMDeltaT [s]
1.                      IMDeltaTMax [s]
```

The IMTIMESTEP command controls for the time stepping of CIMI, but does not control the sub-cycling based on the courant condition. Useful for setting large time-steps for basic tests. Defaults shown.

4.4.3 Restart control

#RESTART command

```
#RESTART
T                       IsRestart
F                       DoReadRestartSatellite (read if IsRestart=T)
```

If IsRestart is true, read in restart files from a previous run. If DoReadRestartSatellite is also true, read the satellite buffer file found in IM/restartIN/restart.sat for tracing satellites. This command is usually read from the IM/restartIN/restart.H file. The default is to start the simulation from scratch, so IsRestart is false.

#SAVERESTART command

```
#SAVERESTART
100.0                   DtSaveRestart
```

Command controls how often restart files are saved. This only works when CIMI is in standalone mode.

#PRERUNFIELD command

```
#PRERUNFIELD
F                       DoWritePrerun
F                       UsePrerun
60.                     DtRead
```

Command controls the saving or reading of the Prerun magnetic field (IM/PrerunField_********.dat) and ionospheric potential (IM/PrerunIE_********.dat) files, where ******** is the simulation time in seconds from the simulation start time. While coupled with GM, the user can save the calculated field line traces to IM/. As the field line traces can be the most computationally expensive part routine in CIMI, this allows for

quick reconfiguration of a GM-coupled run using CIMI in standalone while still using the BATSRUS fields. Default values shown.

****NOTE****

Files are only saved when DoWritePrerun=.true. in the GM-coupled run; UsePrerun and DtReadSat are NOT read while DoWritePrerun=.true. When performing a restart, set UsePrerun=.true. with the time cadence given by DtRead. Consult input/testfiles/PARAM.in.test.Prerun usage of the PrerunSat files in conducting the Prerun test.

#PRERUNSAT command

```
#PRERUNSAT
F                       DoWritePrerunSat
F                       UsePrerunSat
60.                     DtReadSat
```

Command controls the saving or reading of the Prerun satellite trace files ((IM/PrerunSat_********.dat)), where ******** is the simulation time in seconds from the simulation start time. While coupled with GM, the user can save the calculated satellite traces to IM/. As the satellite traces can be quite computationally expensive as the number of satellites increases, this allows for quick reconfiguration of a GM-coupled run using CIMI in standalone while still using the BATSRUS fields. Default values shown.

****NOTE****

Files are only saved when DoWritePrerunSat=.true. in the GM-coupled run; UsePrerunSat and DtReadSat are NOT read while DoWritePrerunSat=.true. When performing a restart, set UsePrerunSat=.true. with the time cadence given by DtReadSat. Consult input/testfiles/PARAM.in.test.Prerun for usage of the PrerunSat files in the Prerun test.

4.4.4 Numerical scheme

#LIMITER command

```
#LIMITER
F                       UseMcLimiter
2                       BetaLimiter
```

Set whether or not the MC limiter is used. If it is not, the super bee limiter is used. Also set the Beta parameter for the MC limiter. The default value is shown.

#DRIFTSCHEME command

```
#DRIFTSCHEME
2               iOrderLat
2               iOrderLon
```

Specifies the spatial advection scheme. If iOrderLat = 7 and iOrderLon = 7, then construct latitude/longitude inter flux in 7th order scheme, using ULTIMATE advection scheme (Lagrangian interpolation + Universal Limiter).

If iOrderLat and iOrderLon are both equal to 2, then use the default 2nd order scheme and Superbee limiter.

Default values shown.

NOTES:

1. Users wanting to use the 7th order scheme in latitude should compile CIMI with the GridUniformL option.

2. iOrderLat and iOrderLon can take integer values of [1-7], but have only been tested and developed with iOrderLat=iOrderLon=2 and iOrderLat=iOrderLon=7.

#HIGHERORDERDRIFT command

```
#HIGHERORDERDRIFT
F                       UseHigherOrder
2                       iOrderLat
2                       iOrderLon
```

This command has the same functionality as the updated #DRIFTSCHEME, but is included for backwards compatibility. Default values shown.

#STRICTDRIFT command

```
#STRICTDRIFT
F                       IsStrictDrift
```

This command specifies whether to force phase space density to be ≥ 0. If set to TRUE and phase space density goes negative for any species, the code will immediately stop. Default shown.

4.4.5 Initial and boundary conditions

#INITIALF2 command

```
#INITIALF2
F                       IsEmptyInitial
F                       IsGmInitial
T                       IsDataInitial
F                       IsRBSPData (read if IsDataInitial=T)
```

Determines whether to fill the fluxes in the simulation domain based on a Maxwellian determined from MHD quantities (IsGmInitial=T) or to set the initial fluxes in the simulation domain to zero (IsEmptyInitial=T, not recommended) or to set the initial fluxes in the simulation to values from AMPTE/CCE data (IsDataInitial=T). One can also put RBSP observed fluxes into the same format as the AMPTE/CCE data to initialize CIMI with RBSP observations which is controlled with the IsRBSPData logical, which is only read if IsDataInitial=T.

The default IsGmInitial=T and all others are false.

#INITIALLSTAR command

```
#INITIALLSTAR
F                       DoLstarInitialization
```

Determines whether to initialize the PSD array from dipolar L values or CIMI calculated Lstar. Default is shown.

#PLASMASHEET command

```
#PLASMASHEET
F                       UseYoungEtAl
T                       UseBoundaryEbihara
```

Command determines if the empirical boundary conditions for ions to be applied at the plasmasheet boundary. Variable UseBoundaryEbihara is only read if CIMI is compiled in stand-alone mode. If UseBoundaryEbihara is .true., uses Ebihara & Ejiri, 2000 and Borovsky et al., 1998 models for boundary density and temperatures, respectively. When false, uses the Tsyganenko-Mukai plasmasphere model. Default shown.

#BMODEL command

```
#BMODEL
mhd                     NameModel
T                       UseCorotation
T                       UsePotential
```

Specify the magnetic field model that CIMI will use. Acceptable values for NameModel are:

- 'dip' - Static Dipolar magnetic field. (Stand-alone)
- 't96' - Tsyganenko 1996 magnetic field model (Stand-alone)
- 't04' - Tsyganenko-Sitnov 2004 storm-time magnetic field model. (Stand-alone)
- 'mhd' - MHD calculated magnetic fields. (Coupled)

Variables UseCorotation and UsePotential are only read when the dipole field is requested. Defaults shown.

#IEMODEL command

```
#IEMODEL
F                       UseWeimer
```

Command sets whether the Weimer ionospheric potential is used. Default shown.

#NGDC_INDICES command

```
#NGDC_INDICES
Indices.dat             NameNGDCFile
```

Command points to the file containing the DST quick look Index and the F10.7 to be used during the run. Command is used only in stand-alone runs. Example file can be seen in input/testfiles/Indices.dat Default shown.

#MHD_INDICES command

```
#MHD_INDICES
imf.dat                 UpstreamFile
```

Command specifies the file containing the upstream solar wind parameters to be used during the run. Formatting is identical to the solar wind input files used in BATSRUS/SWMF, but an example can be found in input/testfiles/imf.dat. Command is used only in stand-alone runs. Default shown.

#KYOTO_DST command

```
#KYOTO_DST
T               UseDstKyoto
Dst_Kyoto.dat   NameDstFile
```

Command points to the file containing the DST Index from the Kyoto website in the IAGA2002 format.

#KYOTO_AE command

```
#KYOTO_AE
T               UseAeKyoto
Ae_Kyoto.dat    NameAeFile
```

Command points to the file containing the AE Index from the Kyoto website (https://wdc.kugi.kyoto-u.ac.jp/aeasy/index.html) in the IAGA2002 format.

#POTSDAM_KP_AP_F107 command

```
#POTSDAM_KP_AP_F107
T               UseKpApF107IndicesFile
```

Command tells CIMI to use the KP,AP,F107 values from https://www.gfz-potsdam.de/en/section/geomagnetism/data-products-services/geomagnetic-kp-index. Note that this file contines data from 1937 to the present, but must be periodically updated with the latest values.

#SOLARWIND command

```
#SOLARWIND
5.0                     DensitySW
400.0                   VelSW
0.0                     BxSW
0.0                     BySW
-5.0                    BzSW
```

Command sets the constant solar wind values to be used through out the simulation. Command is used only in stand-alone runs. VelSW specifies the x-component of the solar wind velocity.

#SMOOTH command

```
#SMOOTH
F                       UseSmooth
60.                     SmoothWindow
```

Command sets the box car averaging smooth window [in seconds] of the input solar wind file. Default is UseSmooth=.false., but SmoothWindow variable is only read if UseSmooth=.true.

4.4.6 Output parameters

#SAVEPLOT command

```
#SAVEPLOT
8                       nCIMIFileType
fls ions                StringPlot
```

```
300.                    DtOutput
F                       DoSaveSeparateFiles
fls e                   StringPlot
60.                     DtOutput
T                       DoSaveSeparateFiles
psd all                 StringPlot
60.                     DtOutput
F                       DoSaveSeparateFiles
vl H                    StringPlot
60.                     DtOutput
T                       DoSaveSeparateFiles
vpdrift ions            StringPlot
60.                     DtOutput
F                       DoSaveSeparateFiles
preci all               StringPlot
60.                     DtOutput
F                       DoSaveSeparateFiles
2d both                 StringPlot
60.                     DtOutput
2d lstar                StringPlot
60.                     DtOutput
F                       DoSaveSeparateFiles
```

The #SAVEPLOT command determines the number, type, and frequency of output from CIMI.

The nCIMIPlotType sets the number of plot types to be read in and configured. For each plot type, the StringPlot parameters define the content of each file and species or domain to be plot.

StringPlot must contain 2 parts:

```
PlotType PlotOption
```

PlotType can take values:

```
‘fls’           - PARTICLE flux information. Also accepts ‘flux’
‘psd’           - PARTICLE Phase Space Density (PSD).
‘vl’            - PARTICLE radial drift. Also accepts ‘vldrift’
‘vp’            - PARTICLE poloidal drift. Also accepts ‘vpdrift’
‘preci’         - PARTICLE precipitation to the ionosphere.
                  Also accepts ’precipitation’ or ‘precip’
‘2d’            - VARIABLE Data on the 2d simulation plane
```

PlotOption controls species information where relevant or controls for domain of output (in the case of ‘2d’). PlotOption takes on values for the aforementioned PARTICLE output types:

```
‘all’                   - Output all particle species.
‘ions’                  - Output only the ion species’ information.
‘e’ or ‘electrons’      - Output only the electron’s information.
‘h’                     - Output only the hydrogen’s information.
‘o’                     - Output only the oxygen’s information.
                          (Only available if CIMI is compiled with ‘-EarthHO’ option.)
‘he’                    - Output only the helium’s information.
                          (Only available if CIMI is compiled with ‘-EarthHOHe’ option.)
```

For 2d output types, PlotOption specifies which VARIABLE output is to be saved. PlotOption can take on values:

```
'lstar' or 'l*'          - Output of VARIABLE L*, the adiabatic drift shell.
'equator' or 'eq'        - Output calculated VARIABLES on the minimum B surface.
'ionosphere' or 'iono'   - Output calculated VARIABLES in the ionosphere.
'both' or 'all'          - Output calculated VARIABLES at both the ionospher and minimum B surface.
```

Plots are saved in IM/plots. PARTICLE flux, PSD, raidal drift, poloidal drift, precipitation files are saved with extensions `'.fls'`, `'.psd'`, `'.vl'`, `'.vp'`, `'.preci'`, respectively. Lstar VARIABLE output is saved with extension `'.lstar'`. IDL scripts for reading these files can be found in CIMI/tools.

VARIABLE output at the equator or ionosphere have extension `'.outs'` and can be read with SWMF IDL visualization scripts.

DtOutput is required to be read in for ALL output file types. The minimum output is currently set to 60 seconds simulation time.

DoSaveSeparateFiles is a logical check to save individual output files for each time step; otherwise a single appended file is saved. DoSaveSeparateFiles is required to be read for all PARTICLE output types and for Lstar VARIABLE output. Files contain the same information as their appended counterparts and can read with the IDL scripts found in CIMI/tools. Individual PARTICLE files are saved with name format YYYYMMDD_HHMMSS_PARTICLE.EXTENSION where particle currently can be `'{h,o,he,e}'` for hydrogen, oxygen, helium and electrons, respectively; values for EXTENSION are detailed above. Separate files for the Lstar VARIABLE are output as YYYYMMDD_HHMMSS.lstar.

Default is nCIMIPlotType=0 so no plot files are saved.

#VERBOSELSTAR command

```
#VERBOSELSTAR
F                       DoVerboseLstar
```

Command controls for the output of the Lstar calculation to the screen, including information about magnetic island locations (those locations where B is not monotonically decreasing) and the maximum Lstar values for each value of the second adiabatic invariant, K.

Default is DoVerboseLstar=F.

#VERBOSELATGRID command

```
#VERBOSELATGRID
F                       DoVerboseLatGrid
```

Prints to screen latitude and equatorial grid information.

Default is DoVerboseLatGrid=F.

#SAVELOG command

```
#SAVELOG
10                      DtSaveLog
```

When this command is set, a log file for CIMI is written out. The log file saves the change in ring current energy content for each species resulting from each operator. A new entry in the log is written out every DtSaveLog seconds of simulation time.

#TYPEBOUNDARY command

```
#TYPEBOUNDARY
ellipse                 TypeBoundary
```

Determines if the IM outer boundary is an 'ellipse' or 'circle.' Default value is shown.

#SETBOUNDARYPARAMS command

```
#SETBOUNDARYPARAMS
2.0                     DeltaRmax    [Re]
2.0                     DeltaMLTmax [hour]
```

The CIMI grid is based in the ionosphere and we trace the field from those footpoints through the magnetosphere. CIMI then does a number of checks to set its domain. First, we check for open-closed boundary, and the CIMI domain must be inside that. We then check for multiple off equator magnetic field minima. This usually occurs on the dayside under northward IMF and strong pressure. This gives the "Shebansky orbits", which cannot currently be captured by CIMI so those field lines are treated as open as well.

If the spacing between two successive minB points is more than DeltaRmax, the line is considered open to avoid excessive deformation. Range is 1 to 3 Re.

If the MLT of the minB changes more than DeltaMLTmax from the footpoint, the line is considered open to avoid excessive warping. Range is 1 to 4 hours.

Default values are shown.

#MINIMUMPRESSURETOGM command

```
#MINIMUMPRESSURETOGM
1e-2                    MinimumPressureToGM
```

Sets minimum pressure passed to GM.

#DTSATOUT command

```
#DTSATOUT
60.0                    DtSatOut
```

Sets the time cadence, in seconds, that the particle fluxes are output. Default shown. do I print this?

4.4.7 Gridding parameters

#TYPEBOUNDARY command

```
#TYPEBOUNDARY
Ellipse                 TypeBoundary
```

Determines if the IM outer boundary is an 'Ellipse' or 'Circle.' Default shown.

#ENERGYGRID command

```
#ENERGYGRID
  0.10000               MinIonEnergy (in keV)
316.22777               MaxIonEnergy (in keV)
```

DEPRECATED - Command sets the minimum and maximum of the ion energy grid for the output fluxes. MinIonEnergy and MaxIonEnergy are the bounds for the energy array in keV; electron energies are multiplied by 10. Results in a 15 element grid per species that is evenly spaced logarithmically. Parameter included for backwards compatibility. For more direct control over the grid, use #SETENERGYGRID command.

#SETENERGYGRID command

```
#SETENERGYGRID
15                      neng
T                       UseLogEGrid
  0.10000               MinIonEnergy (in keV)
316.22777               MaxIonEnergy (in keV)
```

Command provides user with direct control over the size, spacing, and energy extent of the ion energy grid for the output fluxes. neng is the number of elements in the energy grid per species. UseLogEGrid=T sets logarithmic spacing to the grid; setting UseLogEGrid=F changes it to linearly spaced. MinIonEnergy and MaxIonEnergy are the bounds for the energy array in keV; electron energies are multiplied by 10. The values displayed here result in CIMI's default energy grid regardless of #SETENERGYGRID being specified in PARAM.in.

#RBSPENERGYGRID command

```
#RBSPENERGYGRID
F                       UseRBSPGrid
```

Command sets the output energy grid to exactly the energy centroids of the Van Allen Probes' MagEIS and REPT instruments. If this parameter is set, the #ENERGYGRID and #SETENERGYGRID commands are ignored. Default shown.

#LATITUDINALGRID command

```
#LATITUDINALGRID
T                       DoDefineVarNpower
2.                      varNpower
72.4356255492731975             xlatmax
```

If DoDefineVarNpower is true, calculate ionospheric latitude grid by xlat = acos(1./varL)**(1./varNpower) varL is uniformly spacing from varLmin to varLmax. xlatmin = acos(1./varLmin)**(1./varNpower) xlatmax = acos(1./varLmax)**(1./varNpower) Note. If LATITUDINALGRID is not turned on, default varNpower is 2. parameter xlatmax is in a unit of degrees.

4.4.8 Physics parameters

#STRONGDIFFUSION command

```
#STRONGDIFFUSION
F                       UseStrongDiff
```

Applies very effective exponential decay of ring current and radiation belt electron populations. Diffusion time is first (mu) and second (K) adiabatic invariant dependent. Can result in electrons being lost by up to 50%.

Default is UseStrongDiff is false.

#DIAGONALIZEDDIFFUSION command

```
#DIAGONALIZEDDIFFUSION
T                       UseDiagDiffusion
```

If UseDiagDiffusion is true, use (Q1,Q2) coordinates instead of (a0,E) or (M,K) during diffusion calculation, where Q1 = K and Q2 is defined to be Q2_min = E and Q2 is obtaind from constant Q2 curve (dQ2 = 0) in (a0,E), integrating dE/da0 = DaE/Daa Note. UseWaveDiffusion must be true to use this option.

#DECAY command

```
#DECAY
T                       UseDecay
10 hours                DecayTimescale in seconds
```

If UseDecay is true, adds exponential decay to ring current ion populations, so that if there are no other effects, ion phase space density (PSD) decays proportional to

```
exp(-(deltaT/DecayTimescale))
```

This ad hoc decay can improve agreement with observed recovery after magnetic storms. The default DecayTimescale value of 10 hours as above seems to be close to optimal. The decay term is NOT applied to the electron PSD since both ring current and radiation belt electrons are represented. Rapid loss of electron PSD is controlled with the #STRONGDIFFUSION routine.

The default is UseDecay false.

4.4.9 Testing parameters

#DIAGDIFFUSIONTEST command

```
#DIAGDIFFUSIONTEST
F                       UsePitchAngleDiffusionTest
F                       UseEnergyDiffusionTest
```

If either of the parameters is true, calculate diffusion and compare with analytical solution. NOTE: UseWaveDiffusion and UseDiagDiffusion must be true to use this option. Only UsePitchAngleDiffusionTest OR UseEnergyDiffusionTest can be TRUE at a time; the other must be false.

#COMPOSITION command

```
#COMPOSITION
FIXED                   NameCompModel
0.85                    DensityFraction Hp
0.15                    DensityFraction Op
1.0                     DensityFraction e
```

When CIMI is coupled with a single fluid MHD code, the boundary conditions obtained from GM do not determine the composition of the plasma. This command sets the assumed fraction of H+ and O+. The combined global-inner magnetospheric dynamics strongly depends on these values. NameCompModel selects the model used to set the composition. Currently only "FIXED" is implemented. The fraction of O+, H+ and electrons are given by the next three parameters. The first two should add up to 1, and the electron number density should be 1, so in fact only the first parameter is adjustable, the rest is provided for testing purposes only.

Default values are shown.

#WAVEDIFFUSION command

```
#WAVES
T                           UseWaveDiffusion
T                           UseHiss
T                           UseChorus
D_hiss_UCLA.dat             HissWavesD
D_LBchorus_merge3.dat       ChorusWavesD
T                           UseKpIndex
```

This command defines the wave diffusion in CIMI. By default the waves are off so this command must be added to turn on waves. Other commands include whether to use use plasmaspheric hiss and lower band Chorus. The data files for the wave diffusion coeficients that should be in the rundir/IM/ directory. Note that waves can be driven by KP or AE. When KP is used and GM coupling is true we take the KP passed from GM (note that GEOMAGINDICES should be turned on in BATSRUS for this to work. Note, if not using Kp you should make sure to read an AE file using #KYOTO_AE.

4.5 Input Commands for the Hot Electron Ion Drift Integrator: IM Component

4.5.1 Testing

#STARTTIME command

```
#STARTTIME
2002                    iYear
4                       iMonth
17                      iDay
0                       iHour
0                       iMinute
0                       iSecond
```

The #STARTTIME command sets the initial date and time for the simulation in Greenwich Mean Time (GMT) or Universal Time (UT) in stand alone mode. In the SWMF this command checks start times against the SWMF start time and warns if the difference exceeds 1 millisecond. This time is stored in the BATSRUS restart header file.

The default values are shown above. To be used only in standalone mode within the IM domain.

#STOP command

```
#STOP
3600.0                  tSimulationMax [sec]
```

The tSimulationMax variable contains the simulation time in seconds relative to the initial time set by the #STARTTIME command.

The default value is tSimulationMax=0.

#TIMESTEP command

```
#TIMESTEP
20.                     TimeStep [sec]
```

TimeStep should be multiple of 2, since HEIDI uses Time Splitting scheme. The default value of the time step is dt = 20.

#GRID command

```
#GRID
20                      nRadialGrid
24                      nPhiGrid
42                      nEnergyGrid
71                      nPitchGrid
24                      nPointFieldLine
```

The #GRID command allows to set the grid resolution The Resolution parameter refers to the number of grid points within the domain.

```
nRadialGrid     - number of radial grid points
nPhiGrid        - number of azimuthal grid points
nEnergyGrid     - number of energy grid points
nPitchGrid      - number of pitch angle grid points
nPointFieldLine - number of grid points along the magnetic field line
```

#STORM command

```
#STORM
major                   TypeStorm
```

The TypeStorm parameter allows to set the storm: major, moderate, test.

#INNERBOUNDARY command

```
#INNERBOUNDARY
1e6                     Height [m]
```

The Height parameter sets the altitude of the ionosphere-plasmashere boundary.

#ENERGYSETUP command

```
#ENERGYSETUP
0.1                     EnergyLowerBoundary[keV]
0.006                   LowestEnergyCellWidth [keV]
1.26                    rw
```

The #ENERGYSETUP command allows to set the lower energy boundary (EnergyLowerBoundary), the growth multiplier (GrowthMultiplier) of energy cell width (WE(k+1)=WE(k)*rw), and the width of the lowest energy cell (LowestEnergyCellWidth).

#SPECIES command

```
#SPECIES
T                       UseElectron
T                       UseHPlus
T                       UseHePlus
T                       UseOPlus
T                       UseIon
F                       UsePhotoelElectron
F                       UsePlasmaSheetElectron
```

The #SPECIES command allows to set which species to be included into the simulation. PhotoelElectron and PlasmaSheetElectron allow extra calculation for photo-electrons and plasma sheet electrons runs.

#INDICES command

```
#INDICES
1                       WhichKp
0                       Kp
15                      ApIndex
154                     SunspotAverage
```

The #KpINDEX command allows to set the Kp Index.

```
iKp = 0 for Kp constant.
iKP = 1 for Kp-DKP.
iKP = 2 for Kp read from a table.
iKP = 3 for A = 0.
iKP = 4 Kp read from a file.
```

Or just set Kp to some value. ApIndex is the ApIndex index and SunspotAverage indicates the 13 month sunspot average.

#BOUNDARY command

```
#BOUNDARY
none            TypeBoundary
```

The #BOUNDARY command indicates which boundary conditions to use (to match to initial boundary conditions)

```
0 - Initialized to zero everywhere
1 - Maxwellian distribution
2 - Read FI and NI from file
3 - Read FI and NI from file
4 - Quiet ring current, constant with SunspotAverage, phi and pitch angle.
5 - Read FI from file
6 - Nightside injection
7 - Read from .unff file. Use to reastart. No SOPA data.
9 - Injection everywhere
```

#INITIAL command

```
#INITIAL
none                    TypeInitial
0.0                     MaxwellianScallingFactor
0.03                    CharacteristicEnergy
```

The TypeInitial parameter indicates which initial conditions to use (to match to initial boundary conditions)for each species.

```
0 - Initialized to zero everywhere
1 - Loss Cone Distribution. Maxwellian distribution
2 - Gaussian distribution in azimuth (phi) and SunspotAverage about some location.
3 - Read distribution function from input file.
4 - Quiet ring current, constant with SunspotAverage, phi and pitch angle.
5 - Read distribution function from file (restart.bcf).
6 - Nightside plasma sheet injection.
7 - Read from .unff file. Use to reastart.
```

MaxwellianScallingFactor is the scalling factor for Maxwellian initial distribution. CharacteristicEnergy is the characteristic energy for initial state in (keV).

The default is TypeInitial=none.

#OUTPUT command

```
#OUTPUT
T                       DoSaveDistributionFunctionEverywhere
F                       DoSaveEquatorialDistributionFunction
T                       DoSaveEnergyDeposition
F                       DoSaveTotalPrecipitationFlux
F                       DoSaveDifferentialPrecipitationFlux
F                       DoSaveParticleEnergyLosses
T                       DoSaveThermalPlasmaDensity
F                       DoSaveCflForAdvection
T                       DoSaveDriftVelocities
F                       DoSaveEvsLDistributions
```

```
F                       DoSaveParticleLifetimes
T                       DoSavePressureDensityDst
T                       DoSaveUnformatted
T                       DoSaveContinuousSourcesLosses
T                       DoSaveNightsideBCDistribution
F                       DoSaveDifferentialNumberFlux
```

The #OUTPUT command indicates the output options flag arrays:

```
DoSaveDistributionFunctionEverywhere - F throughout magnetosphere
DoSaveEquatorialDistributionFunction - Equ. trapped F throughout magnetosphere
DoSaveEnergyDeposition               - Flux tube energy deposition
DoSaveTotalPrecipitationFlux         - Total precipitation flux (3 E ranges)
DoSaveDifferentialPrecipitationFlux  - Differential precipitation flux
DoSaveParticleEnergyLosses           - Particle and energy losses
DoSaveThermalPlasmaDensity           - Thermal plasma densities
DoSaveCflForAdvection                - CFL numbers for advection operators
DoSaveDriftVelocities                - Drift velocities for advection
DoSaveEvsLDistributions              - E vs. L distributions at given MLT and PA
DoSaveParticleLifetimes              - Particle lifetimes
DoSavePressureDensityDst             - Pressures, densities, and Dst
DoSaveUnformatted                    - Unformatted output of all Fs
DoSaveContinuousSourcesLosses        - Continuous sources and losses of number/energy
DoSaveNightsideBCDistribution        - Nightside boundary condition distribution

Set DoSaveDifferentialNumberFlux to false to save the distribution function as opposed to the differentia
```

#OUTPUTINFO command

```
#OUTPUTINFO
99266                   NameRun
1800.                   nFrequency [sec]
```

The #OUTPUTFREQUENCY indicates the time interval between printouts. NameRun sets the prefix of the output files. oFrequency represents the frequency the output is saved.

#INJECTIONFREQUENCY command

```
#INJECTIONFREQUENCY
120.                    iFrequency [sec]
```

The #INJECTIONFREQUENCY indicates the time interval between injection BC updates (inject if greater than 0)

#CONVECTION command

```
#CONVECTION
w96                     TypeConvection
```

The TypeConvection parameter indicates how the magnetospheric convection strength is calculated. The default value is shown. Other possibilities are

```
NoConvection               - No convection field
KpVSMaynardChen            - Kp driven V-S with Maynard and Chen activity dependence
MBIVS                      - MBI driven V-S, acitivity from force balance at dusk
McIlwain                   - McIlwain field
McIlwainPlusCPCP           - McIlwain field, Eo driven by Cross Polar Cap Potential
KpVSPlusBurkeWygant        - Kp V-S field plus Burke-Wygant penetration field
McIlwainPlusBurkeWygant    - McIlwain field plus Burke-Wygant penetration field
McIlwainCPCPBurkeWygant    - McIlwain(CPCP) field plus B-W penetration field
KpVSSelfConsintent         - Kp V-S field plus self-consistent penetration field
McIlwainSelfConsistent     - McIlwain field plus self-consistent penetration field
McIlwainCPCPSelfConsistent - McIlwain(CPCP) field plus S-C penetration field
UnshieldedVSwithPen        - Unshielded V-S(CPCP) field plus penetration field
UnshieldedVSnoPen          - Unshielded VS(CPCP) field (no Epen)
W96SC                      - W96 field plus S-C penetration field
W96                        - W96 field everywhere (done in S-C field block)
AMIESC                     - AMIE field plus S-C penetration field
AMIE                       - AMIE field everywhere (done in S-C field block)
ReadFromFile               - Read in electric field from a file
AMIEPot                    - AMIE potentials everywhere
W96Pot                     - Weimer-96 potentials everywhere
Foster                     - Foster potentials everywhere
```

Default value is shown by the example.

#INITIALTHERMALPLASMA command

```
#INITIALTHERMALPLASMA
F                DoReadDGCPM
```

If DoReadDGCPM set to false, is the thermal plasma is initialized by using 48hr of low activity. Needs to be set to T for restart runs and the last _dgcpm file moved to _dgcpm.in.

#SOLARWIND command

```
#SOLARWIND
T                DoReadSolarWind
```

If set to 'True', read solar wind input file.

#PITCHANGLE command

```
#PITCHANGLE
F                        UseConstantStepPA
```

Only one of the options can be true at the same time. UseConstantStepPA sets constant steps in pitch angle.Set to false, uses exact loss cone points.

#INCLUDEWAVES command

```
#INCLUDEWAVES
F                        UseWaves
```

Set to true if self-consistent wave-particle interactions are to be included.

#BFIELD command

```
#BFIELD
analytic                TypeBField
uniform                 TypeBFieldGrid
1.0                     StretchingFactorA
0.0                     StretchingFactorB
```

The TypeBField parameter sets up the magnetic field. If analytic, all numerical integrals are solved using analytical approximations. If numerical, numerical integrals are performed. The TypeBFieldGrid parameter sets up the kind of field line grid. For uniform grid, all point are evenly ditributed along the field line. For non-uniform grid, the grid is more refined in the equatorial region.

StretchingFactorA and StretchingFactorB are two more parameters. Please ask Raluca Ilie for more information.

#SAVERESTART command

```
#SAVERESTART
T                       DoSaveRestart
40.0                    DtSaveRestart
ascii                   TypeFile
```

Default is DoSaveRestart=.true. with DtSaveRestart=-1. This results in the restart file being saved only at the end. An ascii restart file is produced for every DtSaveRestart. Needs to be multiple of the time step. The restart files are overwritten every time a new restart is done. The default directory name is 'restartOUT'.

4.6 Input Commands for the Rice Convection Model 2: IM Component

4.6.1 Testing

#STRICT command

```
#STRICT
T                       UseStrict
```

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

4.6.2 Output

#ASCII command

```
#ASCII
T                       IsAscii
```

The input and output files for RCM can be either ascii or binary. Default value is true for ascii.

#RCMDIR command

```
#RCMDIR
IM                      NameRcmDir
```

The NameRcmDir variable contains the name of the directory to store output files. Default value is "IM". Don't change this unless you know what you are doing.

#SAVEPLOTNAME command

```
#SAVEPLOTNAME
T                       UseEvenPlotName
```

If UseEventPlotName is true, the names of the plot files will contain the event date and time in the YearMoDy_HrMnSc format. If it is false, the simulation time is used in the HourMnSc format.

The default value is false.

#SAVEPLOT command

```
#SAVEPLOT
4                       nFilesPlot
2d min idl              StringPlot
100                     DnSavePlot
-1                      iDtSavePlot [sec]
3d max tec              StringPlot
-1                      DnSavePlot
60                      iDtSavePlot [sec]
2d mc1 idl              StringPlot
100                     DnSavePlot
-1                      iDtSavePlot [sec]
2d mc2 idl              StringPlot
100                     DnSavePlot
-1                      iDtSavePlot [sec]
```

Default is nFilesPlot=0.

StringPlot must contain the following 3 parts (separated with space) in arbitrary order:

```
plotarea   = '2d', '3d'
plotvar    = 'min', 'max', 'mc1', 'mc2'
plotformat = 'tec', 'idl'
```

The plotformat specifies whether to output files for processing in either Tecplot or Idl. Tecplot formatt is ascii, idl is binary (therefore, more compact).

The plotarea string defines the region for the plots, 2d or 3d. Currently the 3D output is implemented for Tecplot format only. 2d means writing variables on the two-dimensional ionospheric grid. 3d refers not to the third spatial dimension but to the energy dependences (writing 3d is essentially writing the distribution function of ring current population).

The plotvar string specifies which variables to write, either a minimum set, or a maximum set (the 'rcm' string is preserved for backwards compatibility only, and it is the same as the 'max' string), or one of the custom sets 'mc1' or 'mc2'. The 'min' value corresponds to the variables 'rho T P rho(MHD) T(MHD) P(MHD' in 2D and to 'eeta veff' in 3D.

The 'max' value corresponds to a lot of variables in 2D. For Tecplot format they include the 'min' variables, but for IDL format the 'max' and 'min' variables are distinct, so you need to specify two IDL plot files to get all the variables. In 3D only the Tecplot format is available, and the 'max' variable set contains 'bndloc Vm —B— V birk alam eeta veff w'.

The values 'mc1' and 'mc2' were originally designed to be used by CCMC. For 2d files, 'mc1' and 'mc2' mean the same thing and define a set of both ionospheric quantities (potential, field-aligned currents, conductances) as well as plasma moments of the ring current species. For 3d, 'mc1' and 'mc2' will cause output of full particle energy spectra (together with some extra variables); for 'mc1', these spectra will be in the form of flux-tube content (proportional to the phase space density), while for 'mc2' it will be differential particle fluxes.

The DnSavePlot and DtSavePlot integers define the plotting frequency in number of time steps and number of seconds, respectively. A negative value -1 means that the frequency parameter is ignored. Note that DtSavePlot must be a multiple of the time step iDtRcm (see the #TIMESTEP command).

4.6.3 Time integration

#RESTART command

```
#RESTART
T                       DoRestart
```

The DoRestart parameter determines if the RCM starts from the restart files saved from a previous run, or from scratch via the standard input files. The default is DoRestart = .false.

#TIMESTEP command

```
#TIMESTEP
5                       iDtRcm
```

The iDtRcm parameter defines the time step in seconds. The default value is 5 seconds.

4.6.4 Physics parameters

#COMPOSITION command

```
#COMPOSITION
```

```
FIXED                   NameCompModel
0.7                     FractionH
0.3                     FractionO

#COMPOSITION
YOUNG                   NameCompModel
128.3                   F107
```

When RCM is coupled with a single fluid MHD code, the boundary conditions obtained from GM do not determine the composition of the plasma. This command sets the assumed fraction of H+ and O+. The combined global-inner magnetospheric dynamics strongly depends on these values. NameCompModel selects the model used to set the composition. If "FIXED" is chosen, the fraction of O+ and H+ must be given as the next two parameters. The two need to add up to 1.0. If NameCompModel is set to "YOUNG", the Young et al., 1982, JGR, Vol. 87 No. A11 empirical relationship is used to set composition dynamically. This relationship sets the composition via Kp index (obtained via coupling with the GM component) and the F10.7 proxy for solar EUV flux. If this model is selected, F10.7 flux must be given as a parameter. Coupling with a multi-ion MHD code removes this adjustable parameter.

Default values are shown by the first example.

#CHARGEEXCHANGE command

```
#CHARGEEXCHANGE
T                       UseChargeExchange
125.                    SunspotNumber
169.                    F107MonthlyMean
90.                     DayOfYear
```

Activate charge exchange in RCM and specify solar conditions. Default values are shown.

#OUTERBOUNDARY command

```
#OUTERBOUNDARY
ellipse                 TypeBoundary
10.0                    xMaxNight
```

Define the outer boundary for RCM. Options are "max" that defines the whole closed field line region and "ellipse" that is fitted inside the closed field line region. For "ellipse" xMaxNight defines the furthest distance in the magnetotail of the ellipse. Default values are shown.

#IONOSPHERE command

```
#IONOSPHERE
IE                      TypeIonosphere
```

Define which ionosphere solver to use. Options are "IE" (SWMF solver) or "RCM" (internal solver). Default is "IE".

#PRECIPITATION command

```
#PRECIPITATION
0.3             LossFactore
0.0             LossFactorH
0.0             LossFactorO
```

Parameter that controls, in a somewhat crude manner, the rate of particle precipitation into the atmosphere (one for each species, total of 3). RCM code calculates the rate of precipitation in the limit of strong pitch-angle scattering, and the factors here reduce this maximum possible rate. In other words, if the max. rate of precipitation for electrons is R_ele, then in the code the rate will be set to R_ele*LossFactore. If LossFactor=0, then there is no loss through precipitaton; if LossFactor=1, the precipitation is maximum possible. This parameter is used in two places in the RCM: one is a loss process for hot magnetospheric population (right-hand side of the advection equation), the other one is to evaluate conductance enhancements in the auroral oval (in the module that computes conductances).

#TEMPERATURE command

```
#TEMPERATURE
3.0             TemperatureFactor
```

Artificially increase the temperature obtained from the magnetosphere model. Default value is 1, i.e. no change.

#DECAY command

```
#DECAY
T                       UseDecay
36000.                  DecayTimescale in seconds
```

If UseDecay is true, add an exponential decay to the ring current, so if there are no other effects, it decays proportional to

```
exp(-(deltaT/DecayTimescale))
```

This ad hoc decay can improve agreement with observed recovery after magnetic storms. The value 10 hours shown by the example seems to be close to optimal.

The default is UseDecay false.

4.7 Input Commands for the FLexible Exascale Kinetic Simulator: PC component

4.7.1 Output

#SAVEPLOT command

```
#SAVEPLOT
6                                nPlot
z=0 var real4 planet year        plotString
-1                               dn
20                               dt
1                                dx
{fluid} ppcS0                    varName
y=0 fluid real4 pic hour         plotString
-100                             dn
5                                dt
-1                               dx
3d fluid amrex planet            plotString
-1                               dn
10                               dt
1                                dx
3d fluid real4 planet compact    plotString
-1                               dn
10                               dt
1                                dx
cut fluid real8 si               plotString
-1                               dn
100                              dt
5                                xMin
10                               xMax
-2                               yMin
2                                yMax
-2                               zMin
2                                zMax
1                                dx
3d particles0 amrex planet       plotString
-1                               dn
200                              dt
1                                dx
```

The first parameter is nPlotFile, which is the number of files to be saved. Each output starts with StringPlot, which specifies the output region, variables, file format and variable units. The plotting frequencies are given by DnOutput and DtOutput.

StringPlot has the format: 'domain variables format unit maxtimeunit'.

The first part 'domain' can be one of the following:

```
x=x0     - a 2D cut of x=x0 plane. 'x0' is the coordinate.
y=y0     - a 2D cut of y=y0 plane. 'y0' is the coordinate.
z=z0     - a 2D cut of z=z0 plane. 'z0' is the coordinate.
3d       - the whole computational domain.
cut      - part of the 3D domain. Need to specify the output range.
```

The value of 'variables' can be:

```
particlesN - location, velocity and weight of species N particles (N=0,1,..)
fluid      - all the fluid variables of the first two species:
             'rhoS0 rhoS1 Bx By Bz Ex Ey Ez uxS0 uyS0 uzS0 uxS1 uyS1 uzS1 pS0 pS1 pXXS0 pYYS0 pZZS0 pXYS0
var        - read from parameter file. The available variables:
             'Bx By Bz Ex Ey Ez ppcS* rhoS* uxS* uyS* uzS* pS* pXXS* pYYS* pZZS* pXYS* pXZS* pYZS* qc div
             where '*' is the number of the species. The variables 'qc' and
             'divEc' are the net charge and the divergence of the electric field
             at the cell center, respectively.
```

The file format can be 'real4', 'real8', 'ascii' 'amrex' or 'hdf5'. The IDL formats 'real4', 'real8' and 'ascii' support the 2D/3D and 'cut' ouput domains for the fields. But they can not save particles. The 'amrex' format supports '3d' and 'cut' domains for saving particles, and '3d' domain for saving fields. 'hdf5' supports '3d' domain for fields, if both AMReX and FLEKS are compiled with a parallel HDF5 library.

Paraview's 'contour' filter does not work well for both 'amrex' and 'hdf5' files. Since Paraview contour each block individually, there are cracks across the block interface. This is a long standing issue of Paraview (https://public.kitware.com/pipermail/paraview/2013-October/029492.html https://discourse.paraview.org/t/amrex-output-contour-plot/4723), and it looks like Paraview developers will not fix it. So a compromised solution is to convert the files to vtm (Paraview) with the scripts in FLEKS/tools.

The 'unit' can be:

```
pic    - normalized CGS units
si     - SI units
planet - planetary unit. (nT, micro-volt/m, km/s, planet radius, amu/cm^3, nPa)
```

The 'maxtimeunit' determines the largest time unit of of the 8-digit file name. The default is 'hour', and 'year' is also available.

```
hour   - HHHHMMSS
year   - YYYYODDD (DDD is the day of the year)
```

No plot file is saved by default.

The idl format files can be visualized by SWMF IDL scripts. The amrex format files can be visualized with Paraview, Visit or yt. Paraview is probably the most user-friendly choice. However, Paraview does not support AMR data well so far (https://gitlab.kitware.com/paraview/paraview/-/issues/20074). To avoid these issues, amrex format files can be converted to either Tecplot ascii (.dat) or vtk files with the scripts (amrex2tec.sh and amrex2vtk.sh) in FLEKS/tools.

#MONITOR command

alias="MONITOR_FLEKS0,MONITOR_FLEKS1,MONITOR_FLEKS2"

```
#MONITOR
10                  dnReport
```

This command controls the frequency of printing simulation information, such as the GMRES convergence history, to STDOUT. The default frequency is every 10 steps.

#SAVELOG command

alias="SAVELOG_FLEKS0,SAVELOG_FLEKS1,SAVELOG_FLEKS2"

```
#SAVELOG
1                           dnSavePic
5                           dnSavePT
```

The frequencies of saving log information for the PIC and particle tracker components, respectively.

#TPSAVE command

alias="TPSAVE_FLEKS0,TPSAVE_FLEKS1,TPSAVE_FLEKS2"

```
#TPSAVE
planet                    unit
10                        dnSave
```

The unit and frequence of saving test particles.

#NOUTFILE command

alias="NOUTFILE_FLEKS0,NOUTFILE_FLEKS1,NOUTFILE_FLEKS2"

```
#NOUTFILE
64                  nFileField
256                 nFileParticle
```

Number of files per AMREX format field or particle output.

4.7.2 Scheme

#PIC command

alias="PIC_FLEKS0,PIC_FLEKS1,PIC_FLEKS2"

```
#PIC
T                   usePIC
```

Turning on/off the PIC module.

#PARTICLETRACKER command

alias="PARTICLETRACKER_FLEKS0,PARTICLETRACKER_FLEKS1,PARTICLETRACKER_FLEKS2"

```
#PARTICLETRACKER
T                   useParticleTracker
```

Turning on/off the test particle module.

#TIMESTEPPING command

alias="TIMESTEPPING_FLEKS0,TIMESTEPPING_FLEKS1,TIMESTEP,TIMESTEP_FLEKS2,TIMESTEP_FLEKS0,TI

```
#TIMESTEPPING
F                   useFixedDt
0.1                 cfl (if useFixedDt is false)

#TIMESTEPPING_FLEKS1
T                   useFixedDt
0.01                dt  (if useFixedDt is true)
```

Setting the CFL or fixed time step. The typical CFL number is 0.1 0.4.

#PARTICLESTAGGERING command

alias="PARTICLESTAGGERING_FLEKS0,PARTICLESTAGGERING_FLEKS1,PARTICLESTAGGERING_FLEKS2"

```
#PARTICLESTAGGERING
T                   isStaggered
```

If isStaggered is true, the primitive particle locations are at $t_{n+1/2}$, otherwise, both locations and velocities are at t_n.

#RANDOMPARTICLESLOCATION command

alias="RANDOMPARTICLESLOCATION_FLEKS0,RANDOMPARTICLESLOCATION_FLEKS1,RANDOMPARTICLESLO

```
#RANDOMPARTICLESLOCATION
T                   isParticleLocationRandom
```

If isParticleLocationRandom is true then particles are initialized with randomaized position, otherwise particles are equally spaced.

#RESAMPLING command

```
#RESAMPLING
T                   doReSampling (rest is read if true)
0.8                 splittingLimit
0.5                 mergingLimit
4                   maxWeightRatio
```

Turning on/off the particle splitting and merging to control the number of particles per cell. If the initial particle number per cell is nPart, splitting (merging) will be triggered once the particle numer per cell becomes lower (higher) than splittingLimit*nPart (mergingLimit*nPart).

If maxWeightRatio is set and it is larger than 1, then the particles with wights larger than maxWeightRatio*averageWeight will be split, where averageWeight is the average weight of the particles in a cell. It is recommended to set maxWeightRatio to 4-8 for OH-PT simulations.

#FASTMERGE command

```
#FASTMERGE
T                     fastMerge
16                  nOld
10                  nNew
3                   nTry
4                     mergeRatioMax
```

When the 'fastMerge' feature is enabled, the merging algorithm aims to merge 'nOld' particles (where 'nOld' is greater than 'nNew') into 'nNew' particles (with 'nNew' being equal to or greater than 5). The goal is to achieve this while ensuring the total mass, momentum, and energy are conserved.

To accomplish this, 'nOld - nNew' particles inside the same 6D-space cell are selected for merging with a Lagrange multiplier solver. This solver minimizes the relative change in the mass of the 'nNew' particles.

The selection process for the 'nOld - nNew' particles chosen for deletion is random, and they are chosen from the 'nOld' particles. However, it's important to note that the Lagrange multiplier solver may fail to find a physically meaningful solution. In such cases, two conditions should be met: (1). The mass of all the new particles must be positive. (2). The change in mass of the new particles should not be excessive, satisfying both 'pNew/pOld .lt. mergeRatioMax' and 'pOld/pNew .lt. mergeRatioMax'. If the solver fails,

the code will make up to 'nTry' attempts to select another set of 'nOld - nNew' particles for deletion until it succeeds.

For simulations involving particle sources, it's recommended to use the following parameters: 'nOld = 16', 'nNew = 10', 'nTry = 3', and '2 .le. mergeRatioMax .le. 4'.

#MERGELIGHT command

```
#MERGELIGHT
T        mergeLight
10       mergePartRatioMax
```

When the 'mergeLight' feature is turned on, the merging algorithm will select the light particles for merging. Among these light particles, the ratio between the heaviest and the lightest particle should be less than 'mergePartRatioMax', otherwise, the merging will not be performed.

#VACUUM command

```
#VACUUM
1E-5            vacuum [amu/cc]
```

If the mass density of a cell is below the 'vacuum' threshold, no macro particles are initially launched from that cell. The particle splitting algorithm also excludes cells with densities less than 'vacuum'. The unit of of 'vacuum' is 'amu/cc'.

#PARTICLELEVRATIO command

```
#PARTICLELEVRATIO
1.2                 pLevRatio
```

For particle splitting and resampling, the base particle number per cell of the level iLev is set to n0*(pLevRatio**iLev), where n0 is the ppc set through #PARTICLES command. The goal is to maintain a larger ppc number for the levels of higher resolution.

#LOADBALANCE command

```
#LOADBALANCE
particle                    loadBalanceStrategy
20                          dn
-1.0                        dt

#LOADBALANCE
cell                        loadBalanceStrategy
-1                          dn
8.0                         dt
```

Load balancing the blocks based on the work load of each cell. The work load is measured by the particle number per cell (ppc) when loadBalanceStrategy is 'particle', which is recommended for OH-PT simulations. If loadBalanceStrategy is 'cell', the work loads of all active cells (do not including cells have been refined) are assumed to be the same.

'dn' and 'dt' control the load balancing frequency.

#DISCRETIZATION command

```
#DISCRETIZATION
0.51                        theta
0.1                         ratioDivC2C
```

The Theta parameter sets θ for the time centering of the electric field, which is calculated at the time $t_{n+\theta}$. When θ is 0.5, the total energy is conserved if ratioDivC2C is 0. However, the exact energy conservation may produce artifical oscillations. Typical values for Theta are from 0.51 to 0.6.

The ratioDivC2C parameter is the fraction of div(E) calculated from an extended stencil in the discretization of the Maxwell equation. Using an extended stencil helps to suppress short-wavelength oscillations.

Default values are shown.

#EFIELDSOLVER command

```
#EFIELDSOLVER
1e-6                    EFieldTol
200                     EFieldIter
```

The tolerance and the maximum iteration number of the electric field GMRES solver. The default values are shown above.

#DIVE command

```
#DIVE
T                        doCorrectDivE
3                        nDivECorrection (read if doCorrectDivE=T)
```

Turning on/off the accurate div(E) cleaning, which corrects the particle locations to satisfy Gauss's law: div(E) = net_charge. Default is true as it is required for accuracy of long simulations. The correction repeats nDivECorrection (default is 3) times. If the divE cleaning is time consuming, reducing nDivECorrection to 1 may also work.

#SMOOTHE command

alias="SMOOTHE_FLEKS0,SMOOTHE_FLEKS1,SMOOTHE_FLEKS2"

```
#SMOOTHE
T                   doSmoothE (rest is read if true)
1                   nSmoothE

#SMOOTHE_FLEKS2
F                   doSmoothE
```

The smoothing algorithm used here is the same as the 'digital filtering'. It seems it is common to apply smoothing tens of times (nSmoothE) for challenging simulations (https://doi.org/10.1016/j.jcp.2011.04.003).

However, smoothing electric field is not always helpful. It adds diffusion to electric field and may artifically increase reconnection rate as it was seen in the island merge simulations. On the other hand, it seems the smoothing may transform the short-wavelength oscillations into long-wavelength oscillations, and the long-wavelength field aligned electric field would accelerate electrons.

#TPRELATIVISTIC command

alias="TPRELATIVISTIC_FLEKS0,TPRELATIVISTIC_FLEKS1,TPRELATIVISTIC_FLEKS2"

```
#TPRELATIVISTIC
T                       isRelativistic
```

If it is true, the test particle velocity update will be relativistic.

4.7.3 Inital and boundary conditions

#GEOMETRY command

```
#GEOMETRY
-1                                  xMin
1                       xMax
-1                                  yMin
1                       yMax
-1                                  zMin
1                       zMax
```

It is used to set the simulation domain size for OH-PT simulations. For MHD-AEPIC simulations, the domain is set from BATSRUS and this command should only exist in the restart file.

#NCELL command

```
#NCELL
32                                      nCellX
64                                      nCellY
48                                      nCellZ
```

It is used to set the domain cell number for OH-PT simulations. For MHD-AEPIC simulations, the cell number is set from BATSRUS and this command should only exist in the restart file.

#REGION command

```
#REGION
region1         regionName
box                       shape
-64.0                   xMin
-32.0                   xMax
-16.0                   yMin
16.0                    yMax
-16.0                   zMin
  0.0                   zMax

#REGION
sphere1         nameRegion
sphere          shape
-10.0                   centerX
 10.0                   centerY
  0.0                   centerZ
 20.0                   radius
```

```
#REGION
shell1    nameRegion
shell     shape
-10.0                   centerX
 10.0                   centerY
  0.0                   centerZ
 20.0                   radiusInner
 20.0                   radiusOuter

#REGION
p1                  nameRegion
paraboloid        shape
0               iAxis
-10.0                           centerX
 10.0                           centerY
  0.0                           centerZ
-10             height
 20.0                           radius1
 10.0                           radius2
```

The regions defined here can be used for setting refinement areas.

For a 'paraboloid', the iAxis is the axis of the paraboloid, and it can be 0 (x-dir), 1 (y-dir), or 2 (z-dir). Its height can be negative, which indicates the orientation direction. The direction of the paraboloid (iAxis) and the directions of radius1 and radius2 form a right-hand system. If iAxis is the x/y/z direction, radius1 is the radius in the y/z/x direction, and radius isin the z/x/y direction.

#REFINEREGION command

```
#REFINEREGION
0                       refineLev
+box1 -sphere1                StringHallRegion

#REFINEREGION
1                       refineLev
+box2 -sphere2                StringHallRegion
```

This command is used to specify the regions of refinement for the level refineLev. The cells inside the '+' shapes but outside the '-' shapes will be refined.

Multiple #REFINEREGION commands can be used for different levels.

#GRIDEFFICIENCY command

```
#GRIDEFFICIENCY
0.7                     gridEfficiency
```

From AMReX online docs: "This threshold value (gridEfficiency), which defaults to 0.7 (or 70

For a block, if the ratio of its cells that are labelled for refinement is more than 'gridEfficiency', the whole block will be refined. Since the block decomposition may be different for the same setup but on different numbers of cores, these simulations may refine grids differently if 'gridEfficiency' is not 1.

#PARTICLES command

alias="PARTICLES_FLEKS0,PARTICLES_FLEKS1,PARTICLES_FLEKS2"

```
#PARTICLES
6                   particles per cell in X
6                   particles per cell in Y
6                   particles per cell in Z

#PARTICLES_FLEKS1
10                  particles per cell in X
10                  particles per cell in Y
1                   particles per cell in Z
```

The command sets particle numbers in each direction for initial condition and ghost cells for a FLEKS region.

#SOURCEPARTICLES command

alias="SOURCEPARTICLES_FLEKS0,SOURCEPARTICLES_FLEKS1,SOURCEPARTICLES_FLEKS2"

```
#SOURCEPARTICLES
1                   particles per cell in X
1                   particles per cell in Y
1                   particles per cell in Z

#SOURCEPARTICLES_FLEKS1
10                  particles per cell in X
10                  particles per cell in Y
1                   particles per cell in Z
```

The command sets particle number per cell for plasma sources.

#ADAPTIVESOURCEPPC command

alias="ADAPTIVESOURCEPPC_FLEKS0,ADAPTIVESOURCEPPC_FLEKS1,ADAPTIVESOURCEPPC_FLEKS2"

```
#ADAPTIVESOURCEPPC
T                   useAdaptiveSourcePPC
```

If it is true, the particle number per cell per step for the sources will be reduced to as low as 1 if the source density is low.

#SUPID command

alias="SUPID_FLEKS0,SUPID_FLEKS1,SUPID_FLEKS2"

```
#SUPID_FLEKS0
3                       nSpecies
1                       supID
1                       supID
1                       supID
```

The starting supID for each species. This command should only exist in a restart file.

#TPREGION command

alias="TPREGION_FLEKS0,TPREGION_FLEKS1,TPREGION_FLEKS2"

```
#TPREGION
boundary           region

#TPREGION_FLEKS0
uniform           region
```

Users can choose generating test particles either in the whole domain (uniform) or only at the boundary.

#TPSTATESI command

alias="TPSTATE_FLEKS0,TPSTATE_FLEKS1,TPSTATE_FLEKS2"

```
#TPSTATESI
1                       nState
1                       iSpecies
0                       vth[m/s]
1e5                     vx[m/s]
2e5                     vy[m/s]
1e6                     vz[m/s]
```

Initializating test particles with user defined thermal velocity (vth) and bulk velocities (vx, vy and vz) for iSpecies. The velocities are in SI unit.

#TPPARTICLES command

alias="TPPARTICLES_FLEKS0,TPPARTICLES_FLEKS1,TPPARTICLES_FLEKS2"

```
#TPPARTICLES
1                  particles per cell in X
1                  particles per cell in Y
1                  particles per cell in Z
```

This command sets the number of test particles per cell.

#TPCELLINTERVAL command

alias="TPCELLINTERVAL_FLEKS0,TPCELLINTERVAL_FLEKS1,TPCELLINTERVAL_FLEKS2"

```
#TPCELLINTERVAL
1                  nIntervalX
1                  nIntervalY
1                  nIntervalZ
```

Generating test particles for every cell may be too much for some applications. This command allows generating test particles for every nInterval cells in the region defined by command #TPREGION

#ELECTRON command

alias="ELECTRON_FLEKS0,ELECTRON_FLEKS1,ELECTRON_FLEKS2"

```
#ELECTRON
-100                    electronChargePerMass

#ELECTRON_FLEKS1
-100                     electronChargePerMass for FLEKS region 1
```

This command sets the charge per mass in normalized unit for electrons. If the _FLEKSX is added, it only sets the value for the X FLEKS region. It is ignored when FLEKS is coupled with BATSRUS configured with the five-moment or six-moment fluid closures that define the electron mass.

Default value is shown.

#TESTCASE command

```
#TESTCASE
TwoStream                       testCase
```

It will set the initial conditions for a specific test case.

#PERIODICITY command

```
#PERIODICITY
F                 isPeriodicX
F                 isPeriodicY
F                 isPeriodicZ
```

We may want to use periodic boundary conditions in some directions even in a coupled simulation, and the command above can be used. Note that if there is only one cell in one direction, periodic boundary conditions will be used in this direction and the parameter above will be ignored. Otherwise, the default value is false.

#MAXBLOCKSIZE command

```
#MAXBLOCKSIZE
8                   nCellX
8                   nCellY
1                   nCellZ
```

This command sets the largest block size that is allowed. All blocks are equal to or smaller than the sizes set here. Users can search "grid information summary" and "load balance report" to obtain the grid information.

8 to 16 cells per direction usually reach a good balance between efficiency and flexibility. 8*8*8 or 16*8*8 are typical for a 3D simulation.

#INITFROMSWMF command

alias="INITFROMSWMF_FLEKS0,INITFROMSWMF_FLEKS1,INITFROMSWMF_FLEKS2"

```
#INITFROMSWMF
T                   initFromSWMF
```

By default, the PIC EM fields and particles are initialized based on the fields received from SWMF. If initFromSWMF is set to false, PIC will be initialized from the #UNIFORMSTATE command.

#NORMALIZATION command

alias="NORMALIZATION_FLEKS0,NORMALIZATION_FLEKS1,NORMALIZATION_FLEKS2"

```
#NORMALIZATION
1.495978707E11          lNorm [m]
3.0e8                   uNorm [m/s]
```

If PIC is not initialized from SWMF (#INITFROMSWMF), length and speed normalization should be set here. Both lNorm and uNorm are in SI units.

#PLASMA command

alias="PLASMA_FLEKS0,PLASMA_FLEKS1,PLASMA_FLEKS2"

```
#PLASMA
1                       nS
1                       mass
0                       charge ! neutral particle

#PLASMA_FLEKS0
2                       nS
1                       mass
1                       charge
0.1                     mass
-1                      charge
```

If PIC is not initialized from SWMF (#INITFROMSWMF), set particle mass and charge here. The units of mass and charge are amu and elementary charge, respectively. If the charge is 0, then it is a neutral species.

#UNIFORMSTATE command

alias="UNIFORMSTATE_FLEKS0,UNIFORMSTATE_FLEKS1,UNIFORMSTATE_FLEKS2"

```
#UNIFORMSTATE
1.8                 rho [amu/cc]  ! Species 1
100                 ux [km/s]
0                   uy [km/s]
0                   uz [km/s]
6000                T  [K]
0.18                rho [amu/cc]  ! Species 2
100                 ux [km/s]
0                   uy [km/s]
0                   uz [km/s]
6000                T  [K]
0.0                 Bx [?]
0.0                 By [?]
0.0                 Bz [?]
0.0                 Ex [?]
0.0                 Ey [?]
0.0                 Ez [?]
```

If PIC is not initialized from SWMF (#INITFROMSWMF), set uniform initial conditions here. This command has only been tested with neutral particles (OH-PT coupling) so far. The units of B and E are SI, and it is likely they will be changed to other units in the future.

#OHION command

```
#OHION
30                          rAnalytic [AU]
T        doGetFromOH

#OHION
30                          rAnalytic [AU]
F         doGetFromOH
4                             rCutoff   [AU]
0.00874         swRho [amu/cc]
1.0e5                     swT   [K]
400                         swU   [km/s]
```

This is only used for OH-PT coupling. The units of the parameters are shown in the example above. Inside the sphere of r .le. rAnalytic, the ion properties are obtained from the OH module if doGetFromOH is true. Otherwise, the following analytic expressions are used for obtainning the ion density, temperature, and velocity: rho(r) = swRho * (rAnalytic/r)**2, T(r) = swT, and u(r) = swU along the ratial direction. Inside r .le. rCutoff, the density is set to rho(rCutoff).

4.7.4 Script commands

#INCLUDE command

```
#INCLUDE
PC/restartIN/FLEKS0_restart.H           NameIncludeFile
```

Include a file. The most useful application is including the restart header file as shown by the example. Including this file helps make sure that the original and restarted runs use consistent settings.

The default is to use a single PARAM.in file.

4.7.5 Restart

#RESTART command

alias="RESTART_FLEKS0,RESTART_FLEKS1,RESTART_FLEKS2"

```
#RESTART
F                  doRestart
```

This command should only exist in the restart file FLEKSX_restart.H to indicate that the FLEKS region is restarted from a file.

In PARAM.in, users should use the command #INCLUDE instead of #RESTART to control the restart.

#RECEIVEICONLY command

alias="RECEIVEICONLY_FLEKS0,RECEIVEICONLY_FLEKS1,RECEIVEICONLY_FLEKS2"

pass

#RESTARTFIONLY command

alias="RESTARTFIONLY_FLEKS0,RESTARTFIONLY_FLEKS1,RESTARTFIONLY_FLEKS2"

```
#RESTARTFIONLY
F                  doRestartFIOnly
```

This command should only exist in the restart file FLEKSX_restart.H. If it is true, only the fluid interface restarts.

#FLUIDVARNAMES command

alias="FLUIDVARNAMES_FLEKS0,FLUIDVARNAMES_FLEKS1,FLUIDVARNAMES_FLEKS2"
PASS

#NSTEP command

alias="NSTEP_FLEKS0,NSTEP_FLEKS1,NSTEP_FLEKS2"

```
#NSTEP
100                 nStep
```

The cycle number of the restart file.
This command should only exist in the restart file FLEKSX_restart.H

#TIMESIMULATION command

alias="TIMESIMULATION_FLEKS0,TIMESIMULATION_FLEKS1,TIMESIMULATION_FLEKS2"

```
#TIMESIMULATION
1.3                 timeSimulation
```

The simulation time of the restart file.
This command should only exist in the restart file FLEKSX_restart.H

#DT command

```
#DT
0.1                 dtSI
0.12                dtNextSI
```

The last and the next time step.
This command should only exist in the restart file FLEKSX_restart.H

4.8 Input Commands for the DGCPM: PS Component

List of PS commands used in the PARAM.in file.

4.8.1 Stand alone mode

Options for configuring the model in stand-alone mode (no SWMF-interface.) Many of these are required if DGCPM is not called from the SWMF, redundant otherwise.

4.8.2 Numerical scheme

Options for configuring the schemes and solvers, boundary conditions, and other numerical options.

#TIMESTEP command

```
#TIMESTEP
10.0    DtStep
```

Set the timestep for the simulation. Default values are shown.

4.8.3 Physical parameters

#KP command

```
#KP
const         NameSourceKp
3.0           ConstKp
```

```
#KP
file          NameSourceKp
kp_data.dat   NameKpFile
```

This command controls the behavior of the built-in Volland-Stern electric field model by setting the value of the Kp index – the only input to VS. NameSourceKp sets the source for Kp, either via an NGDC-formatted file ("file") or as a constant ("const"). If NameSourceKp is "const", then the "ConstKp" value is read and Kp is held constant at that value for all times in the simulation. If NameSourceKp is "file", "NameKpFile" is read and sets the name of the input file.

For an example input file, see DGCPM/Input/kp_test.dat.

#FILLING command

```
#FILLING
3.0     EmptyPeriodClosed
1.0     EmptyPeriodOpen
1.5     FillDays
```

Set parameters that control flux tube refilling and emptying: the time constant for closed flux tube loss ($\tau_{EmptyClosed}$), the time constant for open flux tube loss ($\tau_{EmptyOpen}$), and the time constant for flux tube refilling (τ_{Fill}). All values above should be given in units of days; they are converted to seconds for use in the formulas below.

Default values are shown. If FillDays is set to a negative number, no flux tube refilling will be applied.

In DGCPM, filling and emptying of dayside, closed magnetic field lines works to drive each flux tube towards its saturation density, empirically determined by *Carpenter and Anderson*, JGR, 1992 as,

$$n_{sat} = 10^{-0.3145L+3.9043)} \tag{4.4}$$

given in electrons per cubic centimeter.

On day side closed flux tubes where density is above n_{sat}, the density is dropped back to n_{sat}. If density is less than n_{sat}, a refilling flux is applied:

$$f_{refill} = f_{max} \frac{n_{sat} - n}{n_{sat}} \sin(\phi - \pi/4) \tag{4.5}$$

...where f_{max} is an estimate of the maximum number flux possible (e.g., refilling of an empty flux tube). This is calculated as,

$$f_{max} = \frac{2}{\tau_{Fill}} \frac{V_{min}}{L^3} L_{min} n_{sat,min} \tag{4.6}$$

...where L is the radial distance of the flux tube being refilled, V_{min} and L_{min} are the minimum flux tube volume within the entire computational domain and its radial distance, and $n_{sat,min}$ is the saturation density of the smallest flux tube by volume. The sine factor in the f_{refill} formula scales refilling as a factor of solar zenith angle: flux tubes at local noon refill faster than flux tubes at local dawn or dusk.

On open field lines, density decays:

$$n = n - \frac{1 - \Delta t}{\tau_{EmptyOpen}} \tag{4.7}$$

where Δt is the time step.

Similarly, on night side closed flux tubes, density decays:

$$n = n - \frac{1 - \Delta t}{\tau_{EmptyClosed}} \tag{4.8}$$

4.8.4 Coupling parameters

#GMCOUPLING command

```
#GMCOUPLING
1                       iGmFluidCouple
2.0                     TempPlasma (eV)
```

This command controls the coupling of DGCPM's plasmasphere density and temperature to a GM component in the SWMF. DGCPM will pass its density across its grid to GM. Temperature is converted to thermal pressure based on the density of the plasma. iGmFluidCouple sets the fluid to which density and pressure values are coupled if GM is using a multi-fluid approach. The default behavior is to couple to the first fluid. If GM is in single fluid mode, this parameter has no impact. TempPlasma sets the temperature of the plasmasphere in electron volts.

Default values are shown.

4.8.5 Output parameters

#MLTSLICE command

```
#MLTSLICE
4       nMltSlice
300.0   DtMltSlice
```

Save output extracted at a number (*nMltSlice*) of evenly spaced lines of constant MLT at a cadence of *DtMltSlice* seconds. Each slice will have its own file. For example, if *nMltSlice* is set to 4, output will extracted at 00, 06, 12, and 18 MLT. The current limitation is that no interpolation is made, therefore the total number of azimuthal cells (default is 120) must be evenly divisible by *nMltSlice*. Default values are shown.

#OUTPUT command

```
#OUTPUT
T                       DoWriteStatic
T                       DoWriteDynamic
600.                    OutputInterval
SHORT                   OutputType
DIPOLE                  MagneticType
```

Configure full-domain output files (`dgcpm*.dat` files).

DoWriteStatic turns on static file writing. This is a one-time output file that contains information about the magnetic field configuration, flux tube volume, and grid. It is only relevant for dipole magnetic field configurations. MagneticType sets the field information to be included in the static output file; however, only DIPOLE is implemented at present.

DoWriteDynamic turns on dynamic file writing. These are snapshots of density, velocity, and other variables written at a set interval. OutputInterval sets the frequency of dynamic file writing.

OutputType sets the values to be written to file. All files begin by outputing the grid as theta and phi of the ionospheric footpoints of the field lines (in degrees) and the X, Y coordinates of the field line equatorial crossings (in RE in SM coordinates). Not all units for output values are known. Options include:

- `SHORT` - Density and electric potential (cm^{-3}, V)
- `VELOCITY` - Density, potential, radial velocity, azimuthal velocity (cm^{-3}, V, m/s, $degrees/s$)
- `POTENTIAL` - Density, electric potential (cm^{-3}, V)
- `FLOWS` - Density, radial and azimuthal fluxes (cm^{-3}, ?, ?)
- `OLD` - All variables in an obsolete/basic file format. Not recommended.

Default is to not write `dgcpm*.dat`-type files.

#LOG command

```
#LOG
T                       WriteLogFile
```

This command toggles the writing of the DGCPM log file output. This file writes the cross plasmasphere potential (kV) and Kp index to file every five mintues of simulation time. Default action is to not write the log file.

4.9 Input Commands for the AMPS: PT and PC Components

4.9.1 Amps

#SAVEIDL command

```
#SAVEIDL
3                                   nPlot
z=0 var real4 planet                plotString
-1                                  dn
20                                  dt
1                                   dx
{fluid} numS0                       varName
y=0 fluid real4 pic                 plotString
-100                                dn
5                                   dt
-1                                  dx
cut fluid real8 si                  plotString
-1                                  dn
100                                 dt
5                                   xMin
10                                  xMax
-2                                  yMin
2                                   yMax
-2                                  zMin
2                                   zMax
1                                   dx
```

This command determines the IDL type output from AMPS.

The first parameter is nPlotFile, which is the number of files to be saved. Each output starts with StringPlot, which specify the output region, variables, file format and variable units. The plotting frequencies DnOutput and DtOutput are following. DnOutput is needed and it only works for '3d' type field output.

StringPlot has the format:'domain variables format unit'. 'domain' can be one of the following:

```
 x=x0     - a 2D cut of x=x0 plane. 'x0' is the coordinate. y=y0     - a 2D cut of y=y0 plane. 'y0' is th
3d       - the whole computational domain.
1d       - the whole computational domain. It is essentially the same as '3d'.
cut      - part of the 3D domain. Need to specifiy the output range.
```

The value of 'variables' could be:

```
all                                 - all the PIC field variables of the first two species:
   'qS0 qS1 Bx By Bz Ex Ey Ez jxS0 jyS0 jzS0 jxS1 jyS1 jzS1'
fluid                               - all the FLUID field variables of the first two species:
   'rhoS0 rhoS1 Bx By Bz Ex Ey Ez uxS0 uyS0 uzS0 uxS1 uyS1 uzS1 pS0 pS1 pXXS0 pYYS0 pZZS0 pXYS0 pXZS0 pYZ
var                                 - read from parameter file
```

qS0 and qS1 are charge densities, while rhos0 and rhos1 are mass densities. kXXS0 = sum(vx*vx), where vx is the particle velocity. So kXXS0 includes the effect of bulk velocity. But pXXS0 is true pressure that excludes the influence of the bulk velocity.

The file format could be either 'real4' or 'ascii'. The 'unit' can be:

```
PIC  - normalized PIC unit.
SI   - SI unit. The unit of current and electric field are not well defined.
PLANETARY - planetary unit. B field is in nT, velocity is in km/s, length is in planet radius, density is
```

Note: 1) Available output variables are listed in the function PIC::CPLR::FLUID::get_var in pic_fluid.cpp. 2) DxOutput is only functional for particles and 3d field output now. 3) The position for "cut", "x=", "y="... is in BATSRUS coordinate.

There is no plot file saved by default.

#EFIELDSOLVER command

```
#EFIELDSOLVER
1e-6                    EFieldTol
200                     EFieldIter
```

The tolerance and the maximum iteration number of the electric field GMRES solver. The default values are shown above.

#PARTICLES command

```
#PARTICLES
5                 Particles per cell in X
6                 Particles per cell in Y
1                 Particles per cell in Z
```

The command sets particle numbers in each direction.

#DISCRETIZATION command

```
#DISCRETIZATION
0.51                      theta
0.1                       ratioDivC2C
```

The electric field is calculated at the time $t_{n+\theta}$. When θ is 0.5, the total energy is conserved if ratioDivC2C is 0. However, the exact energy conservation may produce artifical oscillations. 0.51-0.6 are typical values of θ.

'ratioDivC2C' is the ratio of div(E) calculated from extended stencil discretization. It helps to suppress short-wavelength oscillations. Default values are shown. gradRhoRatio and cDiff are not used in AMPS yet.

#DIVE command

```
#DIVE
T                       doCorrectDivE
```

Turning on/off the accurate div(E) cleaning, which corrects the particle locations to satisfy Gauss's law: div(E) = net_charge. Default is true as it is required for accuracy of long simulations.

#RESTART command

```
#RESTART
T                       doCorrectDivE
```

Determine if this is a restart run.

#FIELDLINE command

```
#FIELDLINE
1.5     ROrigin
-175    LonMin
175     LonMax
-85     LatMin
85      LatMax
5       nLon
4       nLat
```

Sets initial points for tracking field lines

#SAMPLING command

```
#SAMPLING
F      SamplingMode
```

if SamplingMode is false, AMPS' sampling procedure is disables. The default value is true

#SAMPLING_LENGTH command

```
#SAMPLING_LENGTH
2                       SamplingLength
```

The number of iteractions that AMPS uses to commect the statistical information before output the numerical solution in a file

#TIMESTEP command

```
#TIMESTEP
2.0                      Dt
```

The command sets the time step that AMPS uses in the simulation

#SAMPLING_OUTPUT_CADENCE command

```
#SAMPLING_OUTPUT_CADENCE
9                      SamplingOutputCadence
```

The cadence between starting sampled procedure for output AMPS' data file. #SAMPLING_OUTPUT_CADENCE is active only when #SAMPLING==false. It activates sampling periodically. Each time sampling is activated, it continues for #SAMPLING_LENGTH iterations, output data file, and disable sampling again.

#TIMESTEPPING command

```
#TIMESTEPPING
F                       useSWMFDt
T                       useFixedDt
1                       fixedDt
```

If useSWMFDt is false, and useFixedDt is true, fixed time step will be used.

#ELECTRON command

```
#ELECTRON
-100                    ElectronChargePerMass
```

This command sets the charge per mass in normalized unit for electrons.
Default value is shown.

#TEST command

```
#TEST
T                       DoTest
```

Switch on testing. Default is false.

4.10 Input Commands for the FLexible Exascale Kinetic Simulator: PC component

4.10.1 Output

#SAVEPLOT command

```
#SAVEPLOT
6                                nPlot
z=0 var real4 planet year        plotString
-1                               dn
20                               dt
1                                dx
{fluid} ppcS0                    varName
y=0 fluid real4 pic hour         plotString
-100                             dn
5                                dt
-1                               dx
3d fluid amrex planet            plotString
-1                               dn
10                               dt
1                                dx
3d fluid real4 planet compact    plotString
-1                               dn
10                               dt
1                                dx
cut fluid real8 si               plotString
-1                               dn
100                              dt
5                                xMin
10                               xMax
-2                               yMin
2                                yMax
-2                               zMin
2                                zMax
1                                dx
3d particles0 amrex planet       plotString
-1                               dn
200                              dt
1                                dx
```

The first parameter is nPlotFile, which is the number of files to be saved. Each output starts with StringPlot, which specifies the output region, variables, file format and variable units. The plotting frequencies are given by DnOutput and DtOutput.

StringPlot has the format: 'domain variables format unit maxtimeunit'.

The first part 'domain' can be one of the following:

```
x=x0     - a 2D cut of x=x0 plane. 'x0' is the coordinate.
y=y0     - a 2D cut of y=y0 plane. 'y0' is the coordinate.
z=z0     - a 2D cut of z=z0 plane. 'z0' is the coordinate.
3d       - the whole computational domain.
cut      - part of the 3D domain. Need to specify the output range.
```

The value of 'variables' can be:

```
particlesN - location, velocity and weight of species N particles (N=0,1,..)
fluid      - all the fluid variables of the first two species:
             'rhoS0 rhoS1 Bx By Bz Ex Ey Ez uxS0 uyS0 uzS0 uxS1 uyS1 uzS1 pS0 pS1 pXXS0 pYYS0 pZZS0 pXY
var        - read from parameter file. The available variables:
             'Bx By Bz Ex Ey Ez ppcS* rhoS* uxS* uyS* uzS* pS* pXXS* pYYS* pZZS* pXYS* pXZS* pYZS* qc di
             where '*' is the number of the species. The variables 'qc' and
             'divEc' are the net charge and the divergence of the electric field
             at the cell center, respectively.
```

The file format can be 'real4', 'real8', 'ascii' 'amrex' or 'hdf5'. The IDL formats 'real4', 'real8' and 'ascii' support the 2D/3D and 'cut' ouput domains for the fields. But they can not save particles. The 'amrex' format supports '3d' and 'cut' domains for saving particles, and '3d' domain for saving fields. 'hdf5' supports '3d' domain for fields, if both AMReX and FLEKS are compiled with a parallel HDF5 library.

Paraview's 'contour' filter does not work well for both 'amrex' and 'hdf5' files. Since Paraview contour each block individually, there are cracks across the block interface. This is a long standing issue of Paraview (https://public.kitware.com/pipermail/paraview/2013-October/029492.html https://discourse.paraview.org/t/amrex-output-contour-plot/4723), and it looks like Paraview developers will not fix it. So a compromised solution is to convert the files to vtm (Paraview) with the scripts in FLEKS/tools.

The 'unit' can be:

```
pic    - normalized CGS units
si     - SI units
planet - planetary unit. (nT, micro-volt/m, km/s, planet radius, amu/cm^3, nPa)
```

The 'maxtimeunit' determines the largest time unit of of the 8-digit file name. The default is 'hour', and 'year' is also available.

```
hour   - HHHHMMSS
year   - YYYYODDD (DDD is the day of the year)
```

No plot file is saved by default.

The idl format files can be visualized by SWMF IDL scripts. The amrex format files can be visualized with Paraview, Visit or yt. Paraview is probably the most user-friendly choice. However, Paraview does not support AMR data well so far (https://gitlab.kitware.com/paraview/paraview/-/issues/20074). To avoid these issues, amrex format files can be converted to either Tecplot ascii (.dat) or vtk files with the scripts (amrex2tec.sh and amrex2vtk.sh) in FLEKS/tools.

#MONITOR command

alias="MONITOR_FLEKS0,MONITOR_FLEKS1,MONITOR_FLEKS2"

```
#MONITOR
10                  dnReport
```

This command controls the frequency of printing simulation information, such as the GMRES convergence history, to STDOUT. The default frequency is every 10 steps.

#SAVELOG command

alias="SAVELOG_FLEKS0,SAVELOG_FLEKS1,SAVELOG_FLEKS2"

```
#SAVELOG
1                           dnSavePic
5                           dnSavePT
```

The frequencies of saving log information for the PIC and particle tracker components, respectively.

#TPSAVE command

alias="TPSAVE_FLEKS0,TPSAVE_FLEKS1,TPSAVE_FLEKS2"

```
#TPSAVE
planet                      unit
10                          dnSave
```

The unit and frequence of saving test particles.

#NOUTFILE command

alias="NOUTFILE_FLEKS0,NOUTFILE_FLEKS1,NOUTFILE_FLEKS2"

```
#NOUTFILE
64                  nFileField
256                 nFileParticle
```

Number of files per AMREX format field or particle output.

4.10.2 Scheme

#PIC command

alias="PIC_FLEKS0,PIC_FLEKS1,PIC_FLEKS2"

```
#PIC
T                   usePIC
```

Turning on/off the PIC module.

#PARTICLETRACKER command

alias="PARTICLETRACKER_FLEKS0,PARTICLETRACKER_FLEKS1,PARTICLETRACKER_FLEKS2"

```
#PARTICLETRACKER
T                   useParticleTracker
```

Turning on/off the test particle module.

#TIMESTEPPING command

alias="TIMESTEPPING_FLEKS0,TIMESTEPPING_FLEKS1,TIMESTEP,TIMESTEP_FLEKS2,TIMESTEP_FLEKS0,TIM

```
#TIMESTEPPING
F                   useFixedDt
0.1                 cfl (if useFixedDt is false)

#TIMESTEPPING_FLEKS1
T                   useFixedDt
0.01                dt  (if useFixedDt is true)
```

Setting the CFL or fixed time step. The typical CFL number is 0.1 0.4.

#PARTICLESTAGGERING command

alias="PARTICLESTAGGERING_FLEKS0,PARTICLESTAGGERING_FLEKS1,PARTICLESTAGGERING_FLEKS2"

```
#PARTICLESTAGGERING
T                   isStaggered
```

If isStaggered is true, the primitive particle locations are at $t_{n+1/2}$, otherwise, both locations and velocities are at t_n.

#RANDOMPARTICLESLOCATION command

alias="RANDOMPARTICLESLOCATION_FLEKS0,RANDOMPARTICLESLOCATION_FLEKS1,RANDOMPARTICLESI

```
#RANDOMPARTICLESLOCATION
T                   isParticleLocationRandom
```

If isParticleLocationRandom is true then particles are initialized with randomaized position, otherwise particles are equally spaced.

#RESAMPLING command

```
#RESAMPLING
T                   doReSampling (rest is read if true)
0.8                 splittingLimit
0.5                 mergingLimit
4                   maxWeightRatio
```

Turning on/off the particle splitting and merging to control the number of particles per cell. If the initial particle number per cell is nPart, splitting (merging) will be triggered once the particle numer per cell becomes lower (higher) than splittingLimit*nPart (mergingLimit*nPart).

If maxWeightRatio is set and it is larger than 1, then the particles with wights larger than maxWeightRatio*averageWeight will be split, where averageWeight is the average weight of the particles in a cell. It is recommended to set maxWeightRatio to 4-8 for OH-PT simulations.

#FASTMERGE command

```
#FASTMERGE
T                     fastMerge
16                  nOld
10                  nNew
3                   nTry
4                     mergeRatioMax
```

When the 'fastMerge' feature is enabled, the merging algorithm aims to merge 'nOld' particles (where 'nOld' is greater than 'nNew') into 'nNew' particles (with 'nNew' being equal to or greater than 5). The goal is to achieve this while ensuring the total mass, momentum, and energy are conserved.

To accomplish this, 'nOld - nNew' particles inside the same 6D-space cell are selected for merging with a Lagrange multiplier solver. This solver minimizes the relative change in the mass of the 'nNew' particles.

The selection process for the 'nOld - nNew' particles chosen for deletion is random, and they are chosen from the 'nOld' particles. However, it's important to note that the Lagrange multiplier solver may fail to find a physically meaningful solution. In such cases, two conditions should be met: (1). The mass of all the new particles must be positive. (2). The change in mass of the new particles should not be excessive, satisfying both 'pNew/pOld .lt. mergeRatioMax' and 'pOld/pNew .lt. mergeRatioMax'. If the solver fails,

the code will make up to 'nTry' attempts to select another set of 'nOld - nNew' particles for deletion until it succeeds.

For simulations involving particle sources, it's recommended to use the following parameters: 'nOld = 16', 'nNew = 10', 'nTry = 3', and '2 .le. mergeRatioMax .le. 4'.

#MERGELIGHT command

```
#MERGELIGHT
T        mergeLight
10       mergePartRatioMax
```

When the 'mergeLight' feature is turned on, the merging algorithm will select the light particles for merging. Among these light particles, the ratio between the heaviest and the lightest particle should be less than 'mergePartRatioMax', otherwise, the merging will not be performed.

#VACUUM command

```
#VACUUM
1E-5            vacuum [amu/cc]
```

If the mass density of a cell is below the 'vacuum' threshold, no macro particles are initially launched from that cell. The particle splitting algorithm also excludes cells with densities less than 'vacuum'. The unit of of 'vacuum' is 'amu/cc'.

#PARTICLELEVRATIO command

```
#PARTICLELEVRATIO
1.2                 pLevRatio
```

For particle splitting and resampling, the base particle number per cell of the level iLev is set to n0*(pLevRatio**iLev), where n0 is the ppc set through #PARTICLES command. The goal is to maintain a larger ppc number for the levels of higher resolution.

#LOADBALANCE command

```
#LOADBALANCE
particle                    loadBalanceStrategy
20                          dn
-1.0                        dt

#LOADBALANCE
cell                        loadBalanceStrategy
-1                          dn
8.0                         dt
```

Load balancing the blocks based on the work load of each cell. The work load is measured by the particle number per cell (ppc) when loadBalanceStrategy is 'particle', which is recommended for OH-PT simulations. If loadBalanceStrategy is 'cell', the work loads of all active cells (do not including cells have been refined) are assumed to be the same.

'dn' and 'dt' control the load balancing frequency.

#DISCRETIZATION command

```
#DISCRETIZATION
0.51                        theta
0.1                         ratioDivC2C
```

The Theta parameter sets θ for the time centering of the electric field, which is calculated at the time $t_{n+\theta}$. When θ is 0.5, the total energy is conserved if ratioDivC2C is 0. However, the exact energy conservation may produce artifical oscillations. Typical values for Theta are from 0.51 to 0.6.

The ratioDivC2C parameter is the fraction of div(E) calculated from an extended stencil in the discretization of the Maxwell equation. Using an extended stencil helps to suppress short-wavelength oscillations.

Default values are shown.

#EFIELDSOLVER command

```
#EFIELDSOLVER
1e-6                      EFieldTol
200                       EFieldIter
```

The tolerance and the maximum iteration number of the electric field GMRES solver. The default values are shown above.

#DIVE command

```
#DIVE
T                          doCorrectDivE
3                          nDivECorrection (read if doCorrectDivE=T)
```

Turning on/off the accurate div(E) cleaning, which corrects the particle locations to satisfy Gauss's law: div(E) = net_charge. Default is true as it is required for accuracy of long simulations. The correction repeats nDivECorrection (default is 3) times. If the divE cleaning is time consuming, reducing nDivECorrection to 1 may also work.

#SMOOTHE command

alias="SMOOTHE_FLEKS0,SMOOTHE_FLEKS1,SMOOTHE_FLEKS2"

```
#SMOOTHE
T                   doSmoothE (rest is read if true)
1                   nSmoothE

#SMOOTHE_FLEKS2
F                   doSmoothE
```

The smoothing algorithm used here is the same as the 'digital filtering'. It seems it is common to apply smoothing tens of times (nSmoothE) for challenging simulations (https://doi.org/10.1016/j.jcp.2011.04.003).

However, smoothing electric field is not always helpful. It adds diffusion to electric field and may artifically increase reconnection rate as it was seen in the island merge simulations. On the other hand, it seems the smoothing may transform the short-wavelength oscillations into long-wavelength oscillations, and the long-wavelength field aligned electric field would accelerate electrons.

#TPRELATIVISTIC command

alias="TPRELATIVISTIC_FLEKS0,TPRELATIVISTIC_FLEKS1,TPRELATIVISTIC_FLEKS2"

```
#TPRELATIVISTIC
T                       isRelativistic
```

If it is true, the test particle velocity update will be relativistic.

4.10.3 Inital and boundary conditions

#GEOMETRY command

```
#GEOMETRY
-1                                  xMin
1                       xMax
-1                                  yMin
1                       yMax
-1                                  zMin
1                       zMax
```

It is used to set the simulation domain size for OH-PT simulations. For MHD-AEPIC simulations, the domain is set from BATSRUS and this command should only exist in the restart file.

#NCELL command

```
#NCELL
32                                      nCellX
64                                      nCellY
48                                      nCellZ
```

It is used to set the domain cell number for OH-PT simulations. For MHD-AEPIC simulations, the cell number is set from BATSRUS and this command should only exist in the restart file.

#REGION command

```
#REGION
region1         regionName
box                       shape
-64.0                   xMin
-32.0                   xMax
-16.0                   yMin
16.0                    yMax
-16.0                   zMin
  0.0                   zMax

#REGION
sphere1         nameRegion
sphere          shape
-10.0                   centerX
 10.0                   centerY
  0.0                   centerZ
 20.0                   radius
```

```
#REGION
shell1    nameRegion
shell     shape
-10.0                  centerX
 10.0                  centerY
  0.0                  centerZ
 20.0                  radiusInner
 20.0                  radiusOuter

#REGION
p1                  nameRegion
paraboloid        shape
0               iAxis
-10.0                          centerX
 10.0                          centerY
  0.0                          centerZ
-10             height
 20.0                          radius1
 10.0                          radius2
```

The regions defined here can be used for setting refinement areas.

For a 'paraboloid', the iAxis is the axis of the paraboloid, and it can be 0 (x-dir), 1 (y-dir), or 2 (z-dir). Its height can be negative, which indicates the orientation direction. The direction of the paraboloid (iAxis) and the directions of radius1 and radius2 form a right-hand system. If iAxis is the x/y/z direction, radius1 is the radius in the y/z/x direction, and radius isin the z/x/y direction.

#REFINEREGION command

```
#REFINEREGION
0                        refineLev
+box1 -sphere1                 StringHallRegion

#REFINEREGION
1                        refineLev
+box2 -sphere2                 StringHallRegion
```

This command is used to specify the regions of refinement for the level refineLev. The cells inside the '+' shapes but outside the '-' shapes will be refined.

Multiple #REFINEREGION commands can be used for different levels.

#GRIDEFFICIENCY command

```
#GRIDEFFICIENCY
0.7                      gridEfficiency
```

From AMReX online docs: "This threshold value (gridEfficiency), which defaults to 0.7 (or 70

For a block, if the ratio of its cells that are labelled for refinement is more than 'gridEfficiency', the whole block will be refined. Since the block decomposition may be different for the same setup but on different numbers of cores, these simulations may refine grids differently if 'gridEfficiency' is not 1.

#PARTICLES command

alias="PARTICLES_FLEKS0,PARTICLES_FLEKS1,PARTICLES_FLEKS2"

```
#PARTICLES
6                  particles per cell in X
6                  particles per cell in Y
6                  particles per cell in Z

#PARTICLES_FLEKS1
10                 particles per cell in X
10                 particles per cell in Y
1                  particles per cell in Z
```

The command sets particle numbers in each direction for initial condition and ghost cells for a FLEKS region.

#SOURCEPARTICLES command

alias="SOURCEPARTICLES_FLEKS0,SOURCEPARTICLES_FLEKS1,SOURCEPARTICLES_FLEKS2"

```
#SOURCEPARTICLES
1                  particles per cell in X
1                  particles per cell in Y
1                  particles per cell in Z

#SOURCEPARTICLES_FLEKS1
10                 particles per cell in X
10                 particles per cell in Y
1                  particles per cell in Z
```

The command sets particle number per cell for plasma sources.

#ADAPTIVESOURCEPPC command

alias="ADAPTIVESOURCEPPC_FLEKS0,ADAPTIVESOURCEPPC_FLEKS1,ADAPTIVESOURCEPPC_FLEKS2"

```
#ADAPTIVESOURCEPPC
T                  useAdaptiveSourcePPC
```

If it is true, the particle number per cell per step for the sources will be reduced to as low as 1 if the source density is low.

#SUPID command

alias="SUPID_FLEKS0,SUPID_FLEKS1,SUPID_FLEKS2"

```
#SUPID_FLEKS0
3                      nSpecies
1                      supID
1                      supID
1                      supID
```

The starting supID for each species. This command should only exist in a restart file.

#TPREGION command

alias="TPREGION_FLEKS0,TPREGION_FLEKS1,TPREGION_FLEKS2"

```
#TPREGION
boundary           region

#TPREGION_FLEKS0
uniform           region
```

Users can choose generating test particles either in the whole domain (uniform) or only at the boundary.

#TPSTATESI command

alias="TPSTATE_FLEKS0,TPSTATE_FLEKS1,TPSTATE_FLEKS2"

```
#TPSTATESI
1                       nState
1                       iSpecies
0                       vth[m/s]
1e5                     vx[m/s]
2e5                     vy[m/s]
1e6                     vz[m/s]
```

Initializating test particles with user defined thermal velocity (vth) and bulk velocities (vx, vy and vz) for iSpecies. The velocities are in SI unit.

#TPPARTICLES command

alias="TPPARTICLES_FLEKS0,TPPARTICLES_FLEKS1,TPPARTICLES_FLEKS2"

```
#TPPARTICLES
1                   particles per cell in X
1                   particles per cell in Y
1                   particles per cell in Z
```

This command sets the number of test particles per cell.

#TPCELLINTERVAL command

alias="TPCELLINTERVAL_FLEKS0,TPCELLINTERVAL_FLEKS1,TPCELLINTERVAL_FLEKS2"

```
#TPCELLINTERVAL
1                   nIntervalX
1                   nIntervalY
1                   nIntervalZ
```

Generating test particles for every cell may be too much for some applications. This command allows generating test particles for every nInterval cells in the region defined by command #TPREGION

#ELECTRON command

alias="ELECTRON_FLEKS0,ELECTRON_FLEKS1,ELECTRON_FLEKS2"

```
#ELECTRON
-100                    electronChargePerMass

#ELECTRON_FLEKS1
-100                     electronChargePerMass for FLEKS region 1
```

This command sets the charge per mass in normalized unit for electrons. If the _FLEKSX is added, it only sets the value for the X FLEKS region. It is ignored when FLEKS is coupled with BATSRUS configured with the five-moment or six-moment fluid closures that define the electron mass.

Default value is shown.

#TESTCASE command

```
#TESTCASE
TwoStream                        testCase
```

It will set the initial conditions for a specific test case.

#PERIODICITY command

```
#PERIODICITY
F                isPeriodicX
F                isPeriodicY
F                isPeriodicZ
```

We may want to use periodic boundary conditions in some directions even in a coupled simulation, and the command above can be used. Note that if there is only one cell in one direction, periodic boundary conditions will be used in this direction and the parameter above will be ignored. Otherwise, the default value is false.

#MAXBLOCKSIZE command

```
#MAXBLOCKSIZE
8                  nCellX
8                  nCellY
1                  nCellZ
```

This command sets the largest block size that is allowed. All blocks are equal to or smaller than the sizes set here. Users can search "grid information summary" and "load balance report" to obtain the grid information.

8 to 16 cells per direction usually reach a good balance between efficiency and flexibility. 8*8*8 or 16*8*8 are typical for a 3D simulation.

#INITFROMSWMF command

alias="INITFROMSWMF_FLEKS0,INITFROMSWMF_FLEKS1,INITFROMSWMF_FLEKS2"

```
#INITFROMSWMF
T                  initFromSWMF
```

By default, the PIC EM fields and particles are initialized based on the fields received from SWMF. If initFromSWMF is set to false, PIC will be initialized from the #UNIFORMSTATE command.

#NORMALIZATION command

alias="NORMALIZATION_FLEKS0,NORMALIZATION_FLEKS1,NORMALIZATION_FLEKS2"

```
#NORMALIZATION
1.495978707E11          lNorm [m]
3.0e8                   uNorm [m/s]
```

If PIC is not initialized from SWMF (#INITFROMSWMF), length and speed normalization should be set here. Both lNorm and uNorm are in SI units.

#PLASMA command

alias="PLASMA_FLEKS0,PLASMA_FLEKS1,PLASMA_FLEKS2"

```
#PLASMA
1                       nS
1                       mass
0                       charge ! neutral particle

#PLASMA_FLEKS0
2                       nS
1                       mass
1                       charge
0.1                     mass
-1                      charge
```

If PIC is not initialized from SWMF (#INITFROMSWMF), set particle mass and charge here. The units of mass and charge are amu and elementary charge, respectively. If the charge is 0, then it is a neutral species.

#UNIFORMSTATE command

alias="UNIFORMSTATE_FLEKS0,UNIFORMSTATE_FLEKS1,UNIFORMSTATE_FLEKS2"

```
#UNIFORMSTATE
1.8                 rho [amu/cc]  ! Species 1
100                 ux [km/s]
0                   uy [km/s]
0                   uz [km/s]
6000                T  [K]
0.18                rho [amu/cc]  ! Species 2
100                 ux [km/s]
0                   uy [km/s]
0                   uz [km/s]
6000                T  [K]
0.0                 Bx [?]
0.0                 By [?]
0.0                 Bz [?]
0.0                 Ex [?]
0.0                 Ey [?]
0.0                 Ez [?]
```

If PIC is not initialized from SWMF (#INITFROMSWMF), set uniform initial conditions here. This command has only been tested with neutral particles (OH-PT coupling) so far. The units of B and E are SI, and it is likely they will be changed to other units in the future.

#OHION command

```
#OHION
30                          rAnalytic [AU]
T        doGetFromOH

#OHION
30                          rAnalytic [AU]
F         doGetFromOH
4                             rCutoff   [AU]
0.00874         swRho [amu/cc]
1.0e5                    swT   [K]
400                         swU   [km/s]
```

This is only used for OH-PT coupling. The units of the parameters are shown in the example above. Inside the sphere of r .le. rAnalytic, the ion properties are obtained from the OH module if doGetFromOH is true. Otherwise, the following analytic expressions are used for obtainning the ion density, temperature, and velocity: rho(r) = swRho * (rAnalytic/r)**2, T(r) = swT, and u(r) = swU along the ratial direction. Inside r .le. rCutoff, the density is set to rho(rCutoff).

4.10.4 Script commands

#INCLUDE command

```
#INCLUDE
PC/restartIN/FLEKS0_restart.H           NameIncludeFile
```

Include a file. The most useful application is including the restart header file as shown by the example. Including this file helps make sure that the original and restarted runs use consistent settings.

The default is to use a single PARAM.in file.

4.10.5 Restart

#RESTART command

alias="RESTART_FLEKS0,RESTART_FLEKS1,RESTART_FLEKS2"

```
#RESTART
F                  doRestart
```

This command should only exist in the restart file FLEKSX_restart.H to indicate that the FLEKS region is restarted from a file.

In PARAM.in, users should use the command #INCLUDE instead of #RESTART to control the restart.

#RECEIVEICONLY command

alias="RECEIVEICONLY_FLEKS0,RECEIVEICONLY_FLEKS1,RECEIVEICONLY_FLEKS2"

pass

#RESTARTFIONLY command

alias="RESTARTFIONLY_FLEKS0,RESTARTFIONLY_FLEKS1,RESTARTFIONLY_FLEKS2"

```
#RESTARTFIONLY
F                  doRestartFIOnly
```

This command should only exist in the restart file FLEKSX_restart.H. If it is true, only the fluid interface restarts.

#FLUIDVARNAMES command

alias="FLUIDVARNAMES_FLEKS0,FLUIDVARNAMES_FLEKS1,FLUIDVARNAMES_FLEKS2"
PASS

#NSTEP command

alias="NSTEP_FLEKS0,NSTEP_FLEKS1,NSTEP_FLEKS2"

```
#NSTEP
100                 nStep
```

The cycle number of the restart file.
This command should only exist in the restart file FLEKSX_restart.H

#TIMESIMULATION command

alias="TIMESIMULATION_FLEKS0,TIMESIMULATION_FLEKS1,TIMESIMULATION_FLEKS2"

```
#TIMESIMULATION
1.3                 timeSimulation
```

The simulation time of the restart file.
This command should only exist in the restart file FLEKSX_restart.H

#DT command

```
#DT
0.1                 dtSI
0.12                dtNextSI
```

The last and the next time step.
This command should only exist in the restart file FLEKSX_restart.H

4.11 Input Commands for the MFLAMPA: SP Component

List of commands used in the PARAM.in file for MFLAMPA configured as the SP component of the SWMF.

4.11.1 Domain

#ORIGIN command

```
#ORIGIN
2.5             ROrigin [Rs]
0.0             LonMin [deg]
-70.0           LatMin [deg]
360.0           LonMax [deg]
70.0            LatMax [deg]
```

Origin points on the surface R=ROrigin in the intervals of longitude, (LonMin,LonMax) and latitude, (LatMin,LatMax). Originating from these points are the magnetic field lines traced down from ROrigin to RScMin and up from ROrigin to RIhMax.

4.11.2 Unit

#PARTICLEENERGYUNIT command

```
#PARTICLEENERGYUNIT
keV             ParticleEnergyUnit
```

The particle energy unit: for example, keV, MeV, GeV, etc. We use keV as the default value.

4.11.3 Stand alone mode

#STARTTIME command

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

This command can only be used in the first session.

The #STARTTIME command sets the date and time in Greenwich Mean Time (GMT) or Universal Time (UT) when the simulation starts.

There is no default value.

#TIMEACCURATE command

```
#TIMEACCURATE
T                       IsTimeAccurate
```

Allows solving steady-state distribution (cosmic rays, seed population) by setting DoTimeAccurate to .false.

#TIMESIMULATION command

```
#TIMESIMULATION
1                       SPTime [sec]
```

The SPTime variable contains the simulation time in seconds relative to the initial time set by the #START-TIME command. The #TIMESIMULATION command and SPTime 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 MFLAMPA time is checked against the global SWMF simulation time.

The default value is SPTime=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.

#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.

#SAVERESTART command

```
#SAVERESTART
F                       DoSaveRestart
-1                      DnSaveRestart
-1.0            DtSaveRestart
```

Determine whether and when to save the restart file (STAND ALONE ONLY) Set default value as: False, -1, and -1.0

#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 MFLAMPA 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 MFLAMPA 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.

4.11.4 Stopping criteria

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

#STOP command

```
#STOP
-1                      nIterMax
600.0           TimeMax [sec]
```

This command is only used in stand alone mode.

The nIterMax 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 TimeMax 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.

The #STOP command must be used in every session.

#CHECKSTOPFILE command

```
#CHECKSTOPFILE
T                       DoCheckStopFile
```

This command is only used in stand alone mode.

If DoCheckStopFile is true then the code checks if the MFLAMPA.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 MFLAMPA.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.

#ECHO command

```
#ECHO
T                       DoEcho
```

This command is only used in stand alone mode.

If DoEcho is true then the code set the parameters to be echoed

The default is DoEcho=.false.

#CPUTIMEMAX command

```
#CPUTIMEMAX
3600        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.

4.11.5 Grid

Parameters of the Lagrangian grid used in MFLAMPA

#CHECKGRIDSIZE command

```
#CHECKGRIDSIZE
1000            nParticleMax
4               nLon
4               nLat
```

Check if the configured parameters of the Lagrangian grid match those assumed in the input files

#GRIDNODE command

```
#GRIDNODE
4               nLon
4               nLat
```

Set the configured parameters of the Lagrangian grid

#TESTPOS command

```
#TESTPOS
1               iNodeTest
99              iParticleTest
1               iPTest
```

Set the test position and momentum

#SPREADGRID command

```
#PREADGRID
6                       nSpreadLon
6                       nSpreadLat
```

Set the size of angular grid in longitude and latitude

#SPREADSIGMA command

```
#SPREADGRID
const           SigmaMode
5.0                     Sigma [deg]
```

Set the sigma changes to the lines: SigmaMode is const: all line have the same characteristic spread SigmaMode is linear-lon: sigma changes linear by lines' longitude index SigmaMode is linear-lat: sigma changes linear by lines' latitude index SigmaMode is bilinear: sigma changes bilinear by lon and lat indexes Then we set the "Sigma" of the lines, namely the changes in degrees.

#SPREADSOLIDANGLE command

```
#SPREADGRID
-1.0            RadiusRef [Rs]
-1.0            Sigma [deg]
```

Set reference radius and value of solid angle associated with lines.

#COORDSYSTEM command

```
#COORDSYSTEM
HGI             TypeCoordSystem
```

Type of heliocentric coordinate system used in MFLAMPA (HGI, HGR, HGC).

#MOMENTUMGRID command

```
#MOMENTUMGRID
1.0             EnergyMin
1000.0          EnergyMax
100             nP
1                       nMu
```

Read energy range, the number of energy bins and the energy unit. Usually read from the restart.H file. Read the pitch angle information, determine whether we take the averaged value over all pitch angles. Also, we need to specify how many intervals in the range of [-1, 1] for pitch angle.

#FLUXINITIAL command

```
#FLUXINITIAL
0.01            FluxInitIo
```

Distribution is initialized to have integral flux: default value is set as 0.01 [pfu].

#FLUXCHANNEL command

```
#FLUXCHANNEL
6                       nFluxChannel
```

Flux channel numbers: use the one on GOES by default, 6 channels.

4.11.6 Endbc

#UPPERENDBC command

```
#UPPERENDBC
F           UseUpperEndBc
lism        TypeUpperEndBc
default     TypeLisBc
F           UseModulationPot
600.0       ModulationPot
```

To determine whether we set the upper boundary condition. If so, we turn UseUpperEndBc and have UpperEndSpectrum_I in diffusion. At this time, we need to determine the type of UpperEndBc, LISM is the nominal one for use. For LISM, there are various models fitting or simulating the spectrum, read as TypeLisBc. Here we set Eq.(2) in Usoskin et al. 2005 as default. Then, we also need to determine whether we use LISM or the GCR spectrum at 1 AU. For the latter, we also need the modulation potential (denoted as "phi" in Usoskin et al. 2005), which is temporal and spatial (and even energy) dependent. We stick to the one derived by Oulu neutron monitor and published in the "Dataset 1" of https://cosmicrays.oulu.fi/phi/phi.html.

4.11.7 Advance

#INJECTION command

```
#INJECTION
5.0         SpectralIndex
0.25        Efficency
```

Set up parameters of the power-law spectrum of suprathermal particles at sqrt(2 mp Ti) lt p lt Pinj:
F Efficiency * (Spectral index - 3) / p**SpectralIndex

The injection efficiency is the ratio of suprathermal proton density to the total proton density. Should not exceed one, however, somewhat higher than one values may be of use.

#CFL command

```
#CFL
0.99            Cfl
```

Please send an email to Igor Sokolov for more information.

#POISSONBRACKET command

```
#POISSONBRACKET
F               UsePoissonBracket
```

If UsePoissonBracket is True, the advection scheme is conservative Poisson bracket scheme; otherwise, it is non-conservative.

The default value is false.

4.11.8 Action

#RESTART command

```
#RESTART
T               DoRestart
```

The default value is true.

#DORUN command

```
#DORUN
T                       DoRun
```

If DoRun is False, only magnetic field lines are extracted, but the particle distribution function is not solved.

The default value is true.

#TRACESHOCK command

```
#TRACESHOCK
T               DoTraceShock
```

If DoTraceShock is False, the shock wave is not traced and the density profile is not sharpened around it. The default value is true.

#USEFIXEDMFPUPSTREAM command

```
#USEFIXEDMFPUPSTREAM
F          UseFixedMeanFreePathUpstream
1.0        MeanFreePath0InAu
```

Determine whether we set the mean free path, see Li et al. (2003), doi:10.1029/2002JA009666. The default value for UseFixedMeanFreePathUpstream is set as false, and that for MeanFreePath0InAu is 1.0

#DIFFUSION command

```
#DIFFUSION
T           UseDiffusion
```

Used for testing, to disable diffusion, if UseDiffusion is false. The default value is true.

4.11.9 Turbulence

#SCALETURBULENCE command

```
#SCALETURBULENCE
const           ScaleTurbulenceType
0.24            ScaleTurbulence [AU] at 1 AU
```

cut-off wavenumber of turbulence spectrum: k0 = 2*cPi/Scale The default value is set as const (or constant), i.e., the value (e.g., 0.1) of "ScaleTurbulence [AU] at 1 AU".

#TURBULENTSPECTRUM command

```
#TURBULENTSPECTRUM
F          UseTurbulentSpectrum
```

Determine whether we use the turbulence spectrum Set the default value is false.

4.11.10 Input/output

#USEDATETIME command

```
#USEDATETIME
T          UseDateTime
```

Set the format for the output file names like _eYYYYMMDD_HHMMSS tagging the file with real date and UT tume instant. When unset (default) the time tag looks like _tHHMMSS, where HHMMSS is the simulation time in hours- minutes-seconds passed after the initial time (set by the command #STARTTIME)

#TIMING command

```
#TIMING
T           UseTiming
-2                  nTiming
-1                  nDepthTiming
cumm        TypeTimingReport
```

Set the timing format in SP component, but as default, we set UseTiming as False; if UseTiming is True, one should specify nTiming, nDepthTiming, and TypeTimingReport.

#SAVEINITIAL command

```
#SAVEINITIAL
T           DoSaveInitial
```

As the default (DoSaveInitial=T) the MFLAMPA saves datasets (at least the MHD information) starting from the initial state prior to the run. However, once MFLAMPA restarts, it is useful to set DoSaveInitial to F, in order to avoid rewriting the output files and eliminate duplicate tags in the file tag list, becuase the same output files and tags were saved at the end of previous run.

#READMHDATA command

```
#READMHDATA
T               DoReadMhData
MH_data         NameInputDir
```

Sets the name of input directory with MHD data.

#MHDATA command

```
#MHDATA
idl             TypeFile
120             nFileRead
MH_data.lst     NameTagFile
```

Sets type, number and names of the MHD input files. While running MFLAMPA in cinjunction with the MHD code, these parameters are usually saved into the header file, which is then included into the MFLAMPA parameter file

#NTAG command

```
#NTAG
120             nTag
```

The number of different time-line tags of the MH_data files.

#SAVEPLOT command

```
#SAVEPLOT
3                           nFileOut
mh1d flux ascii             StringPlot
mh2d flux spread ascii      StringPlot
215.0                       Radius [Rs] (read if mh2d/mhtime/flux2d/fluxtime)
```

```
fluxtime flux ascii          StringPlot
215.0                        Radius [Rs] (read if mh2d/mhtime/flux2d/fluxtime)
30.0                         Longitude [deg] (red if fluxtime)
46.0                         Latitude [deg] (red if fluxtime)
```

Sets the number and type of the plotfiles to save. For several types of plot the radius of heliocentric spherical surface on which to plot may be also set. As an example, "mh1d" means to save 1D plot (for each magnetic field line) including coordinates and MHD variables; "R" means radial coordinate (R) plus fluxes; "idl" means to save the data in the idl format. In StringPlot, the first identifier is always a type of plot, the last identifier is a type of file, and the string variables in between denote the list of plot variables extra to the MHD ones.

Please send an email to Igor Sokolov for more information on this command.

#NOUTPUT command

```
#NOUTPUT
10                  nOutput
```

Frequency to plot (effective for IsSteadyState only)

4.12 Input Commands for the PWOM: PW Component

List of PW commands used in the PARAM.in file

4.12.1 Numerical scheme

#SCHEME command

```
#SCHEME
Godunov                 TypeSolver
Godunov                 TypeFlux
0.05                    DtVertical
F                       IsFullyImplicit
F                       IsPointImplicit
F                       IsPointImplicitAll
```

TypeSolver sets the type of solver which is to be used. Currently only two options are available: Godunov and Rusanov. These are actually both Godunov type schemes, but with a first order exact Riemann solver or a second order approximate Riemann solver. TypeFlux sets the intercell flux for the approximate Riemann solver and hence only matters when the Rusanov option is selected for the TypeSolver. The three choices are LaxFriedrichs, Rusanov, and HLL. DtVertical sets the time step for plasma propagating along the field line. IsFullyImplicit determines whether the fully implicit time stepping is used. If not, the last two parameters are also read. IsPointImplicit determines if the point implicit scheme is used or not for the collision terms. IsPointImplicitAll is true if all terms are point implicit.

The default values are shown.

#LIMITER command

```
#LIMITER
1.5                     LimiterBeta
```

Sets the beta parameter in the Rusanov scheme.

#VERTICALGRID command

```
#VERTICALGRID
390                     nPoints
2e6                     DeltaR
```

Sets the number of grid points in the vertical direction and the grid spacing.

#REGRID command

```
#REGRID
T                       DoRegrid (rest of parameters read if true)
T                       DtRegrid
F                       DoSavePoints
```

If DoRegrid is true then regrid the field footpoints every DtRegrid seconds. If DoSavePoints is true, save the footpoints and the triangulation before the regridding is done. The goal is to maintain a relatively uniform coverage of the polar region.

Default is DoRegrid false.

#TIMEACCURATE command

```
#TIMEACCURATE
T               IsTimeAccurate
```

DoTimAccurate determines if the field lines are solved in time accurate mode or in steady state mode.

#TIMESTEP command

```
#TIMESTEP
50              DtHorizontal
```

DtHorizontal is the time step for horizontal motion of the field line. Default value is shown.

#VARIABLEDT command

```
#VARIABLEDT
F                       IsVariableDt
```

When IsVariableDt=T then the vertical time step is variable based on the change in pressure.

#MOTION command

```
#MOTION
T                       DoMoveLine
```

This command determines which to move the field lines as determined by the horizontal convection, or to hold them in their initial locations. The default value is shown.

#ROTATION command

```
#ROTATION
T               UseCentrifugalForce
```

This command determines whether centrifugal forces should be included. The default is shown.

#FIELDLINE command

```
#FIELDLINE
1               nTotalLine
```

nTotalLine sets the number of field lines included in the simulation. The default is shown.

4.12.2 Input/output

#TEST command

```
#TEST
PW_move_line            StringTest
0                       iProcTest
2                       iLineTest
```

Set test parameters. The subroutines to be tested are listed in StringTest, the tested processor is iProcTest, and the tested field line is iLineTest. If iLineTest is 0, all field lines produce test output.

Default is an empty StringTest, i.e. no test output is produced.

#RESTART command

```
#RESTART
T                       IsRestart
```

If the IsRestart variable is true, then the PWOM uses a restart file. Otherwise, the PWOM uses a cold start routine. The default is shown.

#TYPEPLOT command

```
#TYPEPLOT
real4                   TypePlot (ascii/real4/real8)
```

The type (format) of the plot files written. The value "ascii" produces a simple text file, "real4" is single precision binary and "real8" is double precision binary format.

Default value is "ascii".

#SAVEPLOT command

```
#SAVEPLOT
10.0                    DtSavePlot
-1                      DnSavePlot
T                       DoSaveFirst
F                       DoAppendPlot
```

The frequency which plot files are written out are defined here. DoSaveFirst determines whether or not to save a plot on the first call. DoAppendPlot sets appending or not to previous plot file. Default values are shown.

#SAVEPLOTELECTRODYNAMICS command

```
#SAVEPLOTELECTRODYNAMICS
F                       DoPlotElectrodynamics
10                      DtPlotElectrodynamics
```

DoPlotElectrodynamics determines whether the electrodynmics plot information is saved. For instance the field aligned currents, and the polar cap potental can be saved with this command. The frequency of output is determined by the DtPlotElectrodynamics parameter.

#PLOTENERGYGRID command

```
#PLOTENERGYGRID
log                     TypeSpecGrid
0.1                     EminSpecGrid
100.0                   EmaxSpecGrid
25                      nSpecGrid
```

These are parameters for the flux spectrum particle output grid. It defines the type of grid (linear or log), the minimum energy, the maximum energy, and the number of grid points. Defaults are shown. Note that this is just the output grid and does not affect the calculation itself.

#NGDC_INDICES command

```
#NGDC_INDICES
ApFile.dat                      NameApIndexFile
f107.txt                        NameF107File
```

Read the Ap index and F10.7 from two files and calculate the appropriate array of AP indices to feed to MSIS.

#NOAAHPI_INDICES command

```
#NGDCHPI_INDICES
HpiFile.dat                     NameHpiFile
```

Read the HPI from a file. Used to get the Auroral ionization and heating

#HPI command

```
#HPI
0                       HemisphericPower
```

Set the HPI to a constant value. Used to get the Auroral ionization and heating

#AURORA command

```
#AURORA
F                       UseAurora
```

Set whether or not to include auroral ionization and bulk heating.

#SOLARWIND command

```
#SOLARWIND
0                       Bx
0                       By
0                       Bz
400                     Vx
```

This command sets the solar wind parameters for the Weimer model. The solar wind parameters are constant if this command is used

#MHD_INDICES command

```
#MHD_INDICES
IMF.dat                 NameUpstreamFile
```

Read the IMF from a file. Unlike the SOLARWIND command, the IMF can be time dependant in this case.

4.12.3 Control

#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.

#STARTTIME command

```
#STARTTIME
2006                    iYear
7                       iMonth
19                      iDay
0                       iHour
0                       iMinute
0                       iSecond
```

The STARTTIME command sets the integer year, month, day, hour, minute and second at which the simulation begins. This command is only used in standalone mode.

#STOP command

```
#STOP
10                      Tmax
-1                      MaxStep
```

The #STOP command sets the stopping condition for the PWOM during standalone execution. Tmax sets the stop time in timeaccurate mode. and MaxStep sets the maximum number of iterations for non-timeaccurate mode.

4.12.4 Physics

#FLUIDPWI command

```
#FLUIDPWI
T               UseFluidPWI
```

Use wave-particle interaction for the fluid version of PWOM. Default is false.

#FAC command

```
#FAC
F               UseJr
```

UseJr determines whether to use field aligned currents to affect the ion outflow. The default is shown.

#STATICATMOSPHERE command

```
#STATICATMOSPHERE
T                       UseStaticAtmosphere
```

To keep the MSIS atmosphere constant set UseStaticAtmosphere to True.

#MSISPARAM command

```
#MSISPARAM
60                      F107
60                      F107A
4                       AP1
4                       AP2
4                       AP3
```

```
4                       AP4
4                       AP5
4                       AP6
4                       AP7
```

This command sets the MSIS parameters for the the neutral atmosphere. The defaults are shown.

#SE command

```
#SE
T               DoCoupleSE (rest is read if true)
T               UseFeedbackFromSE
F               IsVerboseSE
120             DtGetSe
```

#SE commands DoCoupleSE and UseFeedbackFromSE tells if you should include SE coupling (works for Earth and Jupiter) and if feedback should be included. IsVerboseSE option tells if SE should write lots of output (good for debugging one line) or run quietly (good for lots of lines). DtGetSe tells how frequently SE should be called.

#SETPRECIP command

```
#SETPRECIP
T                       UseFixedPrecip (rest is read if true)
100.0                   PrecipEnergyMin[eV]
1000.0                  PreciptEnergyMax[eV]
400.0                   PrecipEnergyMean[eV]
2.0                     PrecipEnergyFlux[ergs/cm2/s]
```

Sets electron precipitation from a given distribution at the top of the simulated magnetic field line. In PWOM stand alone mode UseFixedPrecip define if we include precipitation, for coupled runs the precipitations is given by IE module. In the case of coupled runs you can force all field lines to have the same precipitations with UseFixedPrecip. The precipitation is defined as a Maxwell energy spectrum covering energies between PrecipEnergyMin and PreciptEnergyMax with mean energy of PrecipEnergyMean. The inflow precipitation flux at the top of the field line is given by PrecipEnergyFlux. See also #POLARRAIN.

Default is UseFixedPrecip false.

#POLARRAIN command

```
#POLARRAIN
T           UsePolarRain (rest is read if true)
80.0        PolarRainEMin[eV]
300.0       PolarRainEMax[eV]
100.0       PolarRainEMean[eV]
1.0e-2      PolarRainEFlux[ergs/cm2/s]
```

Sets a constant electron precipitation from a given distribution at the top of the simulated magnetic field line Applied to all field lines. The precipitation is defined as a Maxwellian energy spectrum covering energies between PolarRainEMin and PolarRainEMax with mean energy of MeanPolarRainEMean. The inflow precipitation flux at the top of the field line is given by PolarRainEFlux. See also #SETPRECIP.

Default is UsePolarRain false.

4.13 Input Commands for the RBE: RB Component

List of RB commands used in the PARAM.in file

4.13.1 Numerical scheme

#SPLITING command

```
#SPLITING
F               UseSplitting
F               UseCentralDiff
```

The SPLITING command determines whether the drift equations are solved using a dimensionally split or unsplit version of the solver. Also whether or not central differencing is used is set here.

The default values are shown.

#TIMESTEP command

```
#TIMESTEP
3.0             Dt
```

Dt is the timestep of the calculation.

Default value is shown.

#LIMITER command

```
#LIMITER
F                       UseMcLimiter
2                       BetaLimiter
```

Set whether or not the MC limiter is used. If it is not, the super bee limiter is used. Also set the Beta parameter for the MC limiter. The default value is shown.

#TIMESIMULATION command

```
#TIMESIMULATION
0.0             TimeSimulation
```

This command specifies the simulation time.

#SPECIES command

```
#SPECIES
e               NameSpecies
```

Determine whether electrons or protons are solved for in RBE. The default is shown.

#BMODEL command

```
#BMODEL
MHD               NameModel
F                 UseFixedB
```

The type of magnetic field is used (t96, t04, or MHD).

#IEMODEL command

```
#IEMODEL
1               iConvect
```

The model for determining convection is set. 1 for Weimer and 2 for MHD

#PLASMASPHERE command

```
#PLASMASPHERE
F               UsePlasmaSphere
```

Command determines whether the plasmasphere is used.

#STARTUPTIME command

```
#STARTUPTIME
0.0             tStartup
```

Startup time for RBE model.

4.13.2 Input/output

#INPUTDATA command

```
#INPUTDATA
2000f223                NameStorm
```

The name of the input data associated with the storm

#SAVEPLOT command

```
#SAVEPLOT
2               DtSavePlot
F               UseSeparatePlotFiles (read if DtSavePlot is positive)
2000f223        OutName (read if UseSeparatePlotFiles is false)
```

Define frequency and output name of plots. If DtSavePlot is negative, no plots are saved. If zero, the final plot is saved only. For positive values, the frequency of saves is given. The frequency has to be a multiple of the time step given in #TIMESTEP command.

The default value is zero, so the final plot is saved only.

#PLOTELECTRODYNAMICS command

```
#PLOTELECTRODYNAMICS
F               DoSaveIe
```

Determine whether or not to save IE output. The frequency is defined in the SAVEPLOT command.

#RESTART command

```
#RESTART
F               DoRestart
```

Determine whether or not to continue a previous run. Default is false.

#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.

4.14 Input Commands for the MFLAMPA: SP Component

List of commands used in the PARAM.in file for MFLAMPA configured as the SP component of the SWMF.

4.14.1 Domain

#ORIGIN command

```
#ORIGIN
2.5             ROrigin [Rs]
0.0             LonMin [deg]
-70.0           LatMin [deg]
360.0           LonMax [deg]
70.0            LatMax [deg]
```

Origin points on the surface R=ROrigin in the intervals of longitude, (LonMin,LonMax) and latitude, (LatMin,LatMax). Originating from these points are the magnetic field lines traced down from ROrigin to RScMin and up from ROrigin to RIhMax.

4.14.2 Unit

#PARTICLEENERGYUNIT command

```
#PARTICLEENERGYUNIT
keV             ParticleEnergyUnit
```

The particle energy unit: for example, keV, MeV, GeV, etc. We use keV as the default value.

4.14.3 Stand alone mode

#STARTTIME command

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

This command can only be used in the first session.

The #STARTTIME command sets the date and time in Greenwich Mean Time (GMT) or Universal Time (UT) when the simulation starts.

There is no default value.

#TIMEACCURATE command

```
#TIMEACCURATE
T                       IsTimeAccurate
```

Allows solving steady-state distribution (cosmic rays, seed population) by setting DoTimeAccurate to .false.

#TIMESIMULATION command

```
#TIMESIMULATION
1                       SPTime [sec]
```

The SPTime variable contains the simulation time in seconds relative to the initial time set by the #START-TIME command. The #TIMESIMULATION command and SPTime 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 MFLAMPA time is checked against the global SWMF simulation time.

The default value is SPTime=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.

#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.

#SAVERESTART command

```
#SAVERESTART
F                       DoSaveRestart
-1                      DnSaveRestart
-1.0            DtSaveRestart
```

Determine whether and when to save the restart file (STAND ALONE ONLY) Set default value as: False, -1, and -1.0

#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 MFLAMPA 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 MFLAMPA 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.

4.14.4 Stopping criteria

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

#STOP command

```
#STOP
-1                      nIterMax
600.0           TimeMax [sec]
```

This command is only used in stand alone mode.

The nIterMax 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 TimeMax 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.

The #STOP command must be used in every session.

#CHECKSTOPFILE command

```
#CHECKSTOPFILE
T                       DoCheckStopFile
```

This command is only used in stand alone mode.

If DoCheckStopFile is true then the code checks if the MFLAMPA.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 MFLAMPA.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.

#ECHO command

```
#ECHO
T                       DoEcho
```

This command is only used in stand alone mode.

If DoEcho is true then the code set the parameters to be echoed

The default is DoEcho=.false.

#CPUTIMEMAX command

```
#CPUTIMEMAX
3600        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.

4.14.5 Grid

Parameters of the Lagrangian grid used in MFLAMPA

#CHECKGRIDSIZE command

```
#CHECKGRIDSIZE
1000            nParticleMax
4               nLon
4               nLat
```

Check if the configured parameters of the Lagrangian grid match those assumed in the input files

#GRIDNODE command

```
#GRIDNODE
4               nLon
4               nLat
```

Set the configured parameters of the Lagrangian grid

#TESTPOS command

```
#TESTPOS
1               iNodeTest
99              iParticleTest
1               iPTest
```

Set the test position and momentum

#SPREADGRID command

```
#PREADGRID
6                       nSpreadLon
6                       nSpreadLat
```

Set the size of angular grid in longitude and latitude

#SPREADSIGMA command

```
#SPREADGRID
const           SigmaMode
5.0                     Sigma [deg]
```

Set the sigma changes to the lines: SigmaMode is const: all line have the same characteristic spread SigmaMode is linear-lon: sigma changes linear by lines' longitude index SigmaMode is linear-lat: sigma changes linear by lines' latitude index SigmaMode is bilinear: sigma changes bilinear by lon and lat indexes Then we set the "Sigma" of the lines, namely the changes in degrees.

#SPREADSOLIDANGLE command

```
#SPREADGRID
-1.0            RadiusRef [Rs]
-1.0            Sigma [deg]
```

Set reference radius and value of solid angle associated with lines.

#COORDSYSTEM command

```
#COORDSYSTEM
HGI             TypeCoordSystem
```

Type of heliocentric coordinate system used in MFLAMPA (HGI, HGR, HGC).

#MOMENTUMGRID command

```
#MOMENTUMGRID
1.0             EnergyMin
1000.0          EnergyMax
100             nP
1                       nMu
```

Read energy range, the number of energy bins and the energy unit. Usually read from the restart.H file. Read the pitch angle information, determine whether we take the averaged value over all pitch angles. Also, we need to specify how many intervals in the range of [-1, 1] for pitch angle.

#FLUXINITIAL command

```
#FLUXINITIAL
0.01            FluxInitIo
```

Distribution is initialized to have integral flux: default value is set as 0.01 [pfu].

#FLUXCHANNEL command

```
#FLUXCHANNEL
6                       nFluxChannel
```

Flux channel numbers: use the one on GOES by default, 6 channels.

4.14.6 Endbc

#UPPERENDBC command

```
#UPPERENDBC
F           UseUpperEndBc
lism        TypeUpperEndBc
default     TypeLisBc
F           UseModulationPot
600.0       ModulationPot
```

To determine whether we set the upper boundary condition. If so, we turn UseUpperEndBc and have UpperEndSpectrum_I in diffusion. At this time, we need to determine the type of UpperEndBc, LISM is the nominal one for use. For LISM, there are various models fitting or simulating the spectrum, read as TypeLisBc. Here we set Eq.(2) in Usoskin et al. 2005 as default. Then, we also need to determine whether we use LISM or the GCR spectrum at 1 AU. For the latter, we also need the modulation potential (denoted as "phi" in Usoskin et al. 2005), which is temporal and spatial (and even energy) dependent. We stick to the one derived by Oulu neutron monitor and published in the "Dataset 1" of https://cosmicrays.oulu.fi/phi/phi.html.

4.14.7 Advance

#INJECTION command

```
#INJECTION
1.0         Efficiency
5.0         SpectralIndex
```

Set up parameters of the power-law spectrum of suprathermal particles at sqrt(2 mp Ti) lt p lt Pinj:
F Efficiency * (Spectral index - 3) / p**SpectralIndex

The injection efficiency is the ratio of suprathermal proton density to the total proton density. Should not exceed one, however, somewhat higher than one values may be of use. Default distribution is f(p) 1.0 * 2 / p**5.

#CFL command

```
#CFL
0.99            Cfl
```

Please send an email to Igor Sokolov for more information.

#POISSONBRACKET command

```
#POISSONBRACKET
F               UsePoissonBracket
```

If UsePoissonBracket is True, the advection scheme is conservative Poisson bracket scheme; otherwise, it is non-conservative.

The default value is false.

4.14.8 Action

#RESTART command

```
#RESTART
T               DoRestart
```

The default value is true.

#DORUN command

```
#DORUN
T                       DoRun
```

If DoRun is False, only magnetic field lines are extracted, but the particle distribution function is not solved.

The default value is true.

#TRACESHOCK command

```
#TRACESHOCK
T               DoTraceShock
```

If DoTraceShock is False, the shock wave is not traced and the density profile is not sharpened around it. The default value is true.

#USEFIXEDMFPUPSTREAM command

```
#USEFIXEDMFPUPSTREAM
F          UseFixedMeanFreePathUpstream
1.0        MeanFreePath0InAu
```

Determine whether we set the mean free path, see Li et al. (2003), doi:10.1029/2002JA009666. The default value for UseFixedMeanFreePathUpstream is set as false, and that for MeanFreePath0InAu is 1.0

#DIFFUSION command

```
#DIFFUSION
T           UseDiffusion
```

Used for testing, to disable diffusion, if UseDiffusion is false. The default value is true.

4.14.9 Turbulence

#SCALETURBULENCE command

```
#SCALETURBULENCE
const           ScaleTurbulenceType
0.24            ScaleTurbulence [AU] at 1 AU
```

cut-off wavenumber of turbulence spectrum: k0 = 2*cPi/Scale The default value is set as const (or constant), i.e., the value (e.g., 0.1) of "ScaleTurbulence [AU] at 1 AU".

#TURBULENTSPECTRUM command

```
#TURBULENTSPECTRUM
F          UseTurbulentSpectrum
```

Determine whether we use the turbulence spectrum Set the default value is false.

4.14.10 Input/output

#USEDATETIME command

```
#USEDATETIME
T          UseDateTime
```

Set the format for the output file names like _eYYYYMMDD_HHMMSS tagging the file with real date and UT tume instant. When unset (default) the time tag looks like _tHHMMSS, where HHMMSS is the simulation time in hours- minutes-seconds passed after the initial time (set by the command #STARTTIME)

#TIMING command

```
#TIMING
T           UseTiming
-2                  nTiming
-1                  nDepthTiming
cumm        TypeTimingReport
```

Set the timing format in SP component, but as default, we set UseTiming as False; if UseTiming is True, one should specify nTiming, nDepthTiming, and TypeTimingReport.

#SAVEINITIAL command

```
#SAVEINITIAL
T           DoSaveInitial
```

As the default (DoSaveInitial=T) the MFLAMPA saves datasets (at least the MHD information) starting from the initial state prior to the run. However, once MFLAMPA restarts, it is useful to set DoSaveInitial to F, in order to avoid rewriting the output files and eliminate duplicate tags in the file tag list, becuase the same output files and tags were saved at the end of previous run.

#READMHDATA command

```
#READMHDATA
T               DoReadMhData
MH_data         NameInputDir
```

Sets the name of input directory with MHD data.

#MHDATA command

```
#MHDATA
idl             TypeFile
120             nFileRead
MH_data.lst     NameTagFile
```

Sets type, number and names of the MHD input files. While running MFLAMPA in cinjunction with the MHD code, these parameters are usually saved into the header file, which is then included into the MFLAMPA parameter file

#NTAG command

```
#NTAG
120             nTag
```

The number of different time-line tags of the MH_data files.

#SAVEPLOT command

```
#SAVEPLOT
3                           nFileOut
mh1d flux ascii             StringPlot
mh2d flux spread ascii      StringPlot
215.0                       Radius [Rs] (read if mh2d/mhtime/flux2d/fluxtime)
```

```
fluxtime flux ascii          StringPlot
215.0                        Radius [Rs] (read if mh2d/mhtime/flux2d/fluxtime)
30.0                         Longitude [deg] (red if fluxtime)
46.0                         Latitude [deg] (red if fluxtime)
```

Sets the number and type of the plotfiles to save. For several types of plot the radius of heliocentric spherical surface on which to plot may be also set. As an example, "mh1d" means to save 1D plot (for each magnetic field line) including coordinates and MHD variables; "R" means radial coordinate (R) plus fluxes; "idl" means to save the data in the idl format. In StringPlot, the first identifier is always a type of plot, the last identifier is a type of file, and the string variables in between denote the list of plot variables extra to the MHD ones.

Please send an email to Igor Sokolov for more information on this command.

#NOUTPUT command

```
#NOUTPUT
10                  nOutput
```

Frequency to plot (effective for IsSteadyState only)

4.15 Input Commands for the Global Ionosphere Thermosphere Model 2: UA Component

GITM is typically run with a `UAM.in` file, which resides in the directory that you are running from. In the framework it obtains parameters from the PARAM.in file.

4.15.1 Time variables

In order to run GITM, a starting time and an ending time must be specified. These are specified using the following commands:

#TIMESTART command

```
#TIMESTART
1999                    iYear
03                      iMonth
18                      iDay
00                      iHour
00                      iMinute
00                      iSecond
```

This command is only used in the stand alone mode.

The #STARTTIME command sets the initial date and time for the simulation.

#TIMEEND command

```
#TIMEEND
1999                    iYear
03                      iMonth
25                      iDay
00                      iHour
00                      iMinute
00                      iDay
```

This command is only used in the stand alone mode.

The #TIMEEND command sets the final date and time for the simulation.

#RESTART command

```
#RESTART
F                       DoRestart
```

There are two commands that are typically not input by a user, but are specified in the restart header file that is read in the exact same way as the input file. It is possible to set these variables, though.

#ISTEP command

```
#ISTEP
1
```

This sets the current iStep to the read in value instead of starting at iteration 1.

#TSIMULATION command

```
#TSIMULATION
0.0
```

This offsets the current start time by the read in amount. It is simply added to the #STARTTIME input time.

#CPUTIMEMAX command

```
#CPUTIMEMAX
7200.0                  CPUTimeMax (seconds)
```

When you are running on a queue-based system, you can use this command to set the exact amount of time that the code should run, then stop and write a restart file. It is good to give a small buffer so the code has time to write files before the queue stops. This buffer time is quite dependent upon the system. On fast I/O machines, I typically give a buffer of only a couple of minutes. On some systems, I sometimes give a full half hour, just to make absolutely sure the code will write all of the correct files and exit before the queue is up.

4.15.2 Initial conditions

#INITIAL command

```
#INITIAL
F                       UseMSIS
T                       UseIRI
200.0                   TempBottom
1200.0                  TempTop
5.0e17                  NDensity1
7.0e18                  NDensity2
3.0e19                  NDensity3
```

On the Earth, empirical models exist which can be used to derive a background atmosphere and ionosphere. These are MSIS (thermosphere and IRI (ionosphere). If MSIS is used, then the all of the species densities are set using MSIS. There are 2 species which MSIS does not include: [NO] and [N(^{2}D)]. We have made up some formula for setting these two species. The neutral temperature is also set using MSIS if UseMSIS = T.

If UseMSIS = F, then GITM reads in the temperature at the bottom and top of the atmosphere (for the initial condition), and the number density at the bottom of the atmosphere for all of the major species (nSpecies, which is set in ModPlanet.f90).

It UseIRI = T, the number densities of the ion species are set by IRI. If it is .false., then the initial densities are set to some very, very small value and the ionosphere is grown out of the chemistry with the neutral atmosphere.

The variables TempMin, etc., are only read in if `UseMSIS = F`.

#APEX command

```
#APEX
T                       UseApex
```

A model of the magnetic field of the Earth can also be used. This variable sets whether to use a realistic magnetic field (T) or a dipole (F). In the current framework only the dipole works.

The default value is false.

4.15.3 Indices

#F107 command

```
#F107
150.0                   f107
150.0                   f107a
```

The $f10.7$ is a proxy for how bright the Sun is in a given set of wavelengths. For a low value (70), the temperature of the atmosphere will be low, and the ionospheric density will be small. For a high value (300), the temperature will be above 1500 K, and ionospheric electron densities will be well above $10^{12}/m^3$. This is used in the routine `calc_euv.f90` to determine the solar flux at the top of the atmosphere.

#HPI command

```
#HPI
10.0            HPI
```

The hemispheric power index (HPI) describes how much power is in the hemispherically summed precipitating electrons (in Gigawatts). This is a number that is typically in the 1-10 range, but during active times can reach 300. This is used in the `get_potential.f90` routine to get the auroral inputs at the top of the atmosphere.

#KP command

```
#KP
1.0                     kp
```

KP is a 3 hour index that summarizes the general activity level of the magnetosphere. It has a range from 0-9. Currently, KP is not used in GITM.

#SOLARWIND command

```
#SOLARWIND
0.0                     Bx
0.0                     By
-2.0                    Bz
400.0                   Vx
```

The interplanetary magnetic field (IMF) and solar wind velocity are used by a number of empirical models of the ionospheric potential. The IMF components typically range between $\pm 10nT$ at Earth. The fields have reached values as high as 75 nT. The B_z component is typically the most geoeffective at Earth, such that negative B_z can cause large ionospheric potentials. The velocity typically ranges between 350-600 km/s, although it has been known to go upwards of 1700 km/s.

4.15.4 Index files

Conversely, you can input time-dependent values of the solar wind and IMF, HPI, Kp, f10.7, etc. There are currently three methods for inputing these quantities.

It is quite easy to incorporate other methods. These three methods are located in the `srcIO` directory with appropriate file names. You can simply copy one of the files, rename the subroutine, modify it to read in the appropriate data, add it to the `Makefile`, add a flag in the `set_inputs.f90` (in both the `src` and `srcIO` directories), then compile it and debug it.

#AMIEFILES command

```
#AMIEFILES
b19980504n
b19980504s
```

Instead of using empirical models of the ionospheric potential and auroral precipitation, you can use model results from the assimilative mapping of ionospheric electrodynamics (AMIE) technique. If you do, you have to specify a Northern hemisphere and Southern hemisphere file.

#MHD_INDICES command

```
#MHD_INDICES
imf.dat
```

The first method only inputs the solar wind velocity, density, temperature and IMF.

#NGDC_INDICES command

```
#NGDC_INDICES
spidr.dat
```

The second method takes data from the NOAA SPIDR interface. You can download almost all of the parameters in this format.

#NOAAHPI_INDICES command

```
#NOAAHPI_INDICES
power_1998.txt
```

The third method only accepts HPI data from the NOAA satellites.

4.15.5 Information

#DEBUG command

```
#DEBUG
1                       iDebugLevel
8                       iDebugProc
60.0                    DtReport
F                       UseBarriers
```

Sometimes debugging can be a real pain. This command makes it slightly easier by allowing you to output more stuff. The `iDebugLevel` variable controls the amount of information output, with 0 outputting only a time-step and a message when output files are written, and 10 being a torrent of so much information you can't read it all. You can also choose which CPU is outputting the information - remember that MPI counts from 0 (not from 1, as most people do). The `DtReport` variable says how often the time-report is given. The `UseBarriers` variable is supposed to stop the code fairly often to make sure all of the processors are on the same page, but there is a bug in this is the `#SATELLITES` are used (don't ask).

4.15.6 The control of nature

The GITM code development has been aimed toward making the code quite versatile. This means that most fundamental parameters have flags so you can turn them off and on. Most of the time, these should be left on, since all of the being `T` means that you are running the "physically correct" condition. But, if you want to turn something off to experiment with the physics, you can do this.

Some of these are not really physically consistent yet. For example, the variable `UseDiffusion` turns off both the Eddy and Molecular diffusion in the neutral density calculations, but leaves the Eddy diffusion on in the temperature equation. Also, if you turn off `UseConduction`, the Eddy diffusion in the temperature equation is turned off. So, things need to be fixed a little bit. Most of the options really only turn off one thing, though.

This is for the neutral temperature equations and **not** for the electron and ion equations.

#THERMO command

```
#THERMO
T                       UseSolarHeating
T                       UseJouleHeating
T                       UseAuroralHeating
T                       UseNOCooling
T                       UseOCooling
T                       UseConduction
```

#DIFFUSION command

```
#DIFFUSION
T
```

This only applies to the neutral densities, and includes both Eddy and Molecular diffusion. It should be separated shortly.

#FORCING command

```
#FORCING
T               UsePressureGradient
T               UseIonDrag
T               UseViscosity
T               UseCoriolis
T               UseGravity
```

The `UsePressureGradient` variable is ignored in this version of GITM, since pressure solved for self-consistently within the solver. Everything else works as a source term (in `calc_sources.f90`, except if `UseGravity = F`, gravity is zeroed in `initialize.f90`.

#IONFORCING command

```
#IONFORCING
T               UseExB
T               UseIonPressure
T               UseGravity
T               UseNeutralDrag
```

All of these variables are used within `calc_ion_v.f90`.

#CHEMISTRY command

```
#CHEMISTRY
T               UseIonChemistry
T               UseIonAdvection
T               UseNeutralChemistry
```

You can turn off the chemistry and the ion advection with these terms.

#ELECTRODYNAMICS command

```
#ELECTRODYNAMICS
60.0                    DtPotential [s]
60.0                    DtAurora    [s]
```

The electric potential and aurora are two of the most expensive routines to run. In addition, they typically don't change on a global-scale on more than a 1-minute cadence. So, you can set these values to something on the order of 60 seconds. If you are using higher temporal resolution IMF parameters, you can set them as low as you want.

4.15.7 Controlling the grid

#GRID command

```
#GRID
8                       nBlocksLon
4                       nBlocksLat
-90.0                   LatStart
90.0                    LatEnd
180.0                   LonStart
```

If LatStart and LatEnd are set to less than -90 and greater than 90, respectively, then GITM does a whole sphere. If not, it models between the two. If you want to do 1-D, set nLons=1, nLats=1 in ModSize.f90, then recompile, then set LatStart and LonStart to the point on the Globe you want to model.

#STRETCH command

```
#STRETCH
65.0                    ConcentrationLatitude
0.0                     StretchingPercentage
1.0                     StretchingFactor
```

The stretched grid is concentrated around the ConcentrationLatitude. The stretching is controlled by StretchingPercentage: 0 means no stretching, 1.0 means a lot. The StretchingFactor provides further control: greater than 1 means stretch less, less than 1 means stretch more.

#ALTITUDE command

```
#ALTITUDE
95.0                    AltMin (km)
600.0                   AltMax (km)
T                       Stretched grid in altitude
```

4.15.8 Output

#SAVEPLOT command

```
#SAVEPLOT
3600.0                  DtRestart [s]
1                       nPlotFile
3DALL                   TypePlotFile
900.0                   DtPlotFile [s]
```

The DtRestart determines the frequency of saving restart files. If negative, no restart files are saved by GITM itself (but the SWMF can still make GITM save its restart files). The number of plot files is given by nPlotFile. For each plot file the type and the saving frequency are given by the parameters TypePlotFile and DtPlotFile, respectively.

The default is DtRestart=-1 and nPlotFile=0

#SATELLITES command

```
#SATELLITES
2
guvi.2002041623.in
15.0
stfd.fpi.in
60.0
```

Index